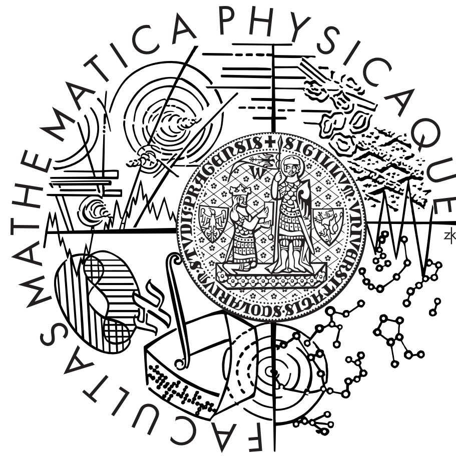
Charles University in Prague
Faculty of Mathematics and Physics

**DOCTORAL THESIS**




Viliam Šimko

# From textual specification to formal verification

Department of Distributed and Dependable Systems

| | |
|---|---|
| Supervisor of the doctoral thesis: | RNDr. Petr Hnětynka, Ph.D |
| Study programme: | Computer Science |
| Specialization: | 4I-2 – Software Systems |

Prague 2013

I would like to thank all those who supported me in my doctoral study and the work on my thesis. I very appreciate the help and counseling received from Petr Hnětynka, Tomáš Bureš, František Plášil and Petr Kroha.

I would also like to thank all my colleagues at the Department of Distributed and Dependable Systems who have contributed valuable feedback, ideas and advice. To avoid any possible bias, the following list is ordered randomly[1]: Petr Tůma, Michal Malohlava, Tomáš Pop, Pavel Jančík, Andrej Podzimek, Alena Koubková, František Plášil, Pavel Parízek, Lubomír Bulej, Jan Kofroň, Petr Hnětynka, Pavel Ježek, Michal Kit, Martin Děcký, Tomáš Kalibera, David Hauzar, Ilias Gerostathopoulos, Jaroslav Keznikl, Petr Kroha, Eva Mládková, Rima Al Ali, Tomáš Bureš, Vojtech Horký, Petra Novotná, Lukáš Marek, Peter Libič.

Regarding the statistical and NLP methods, I would not have been able to employ them in my thesis without the guidance of Martin Holeňa, Zdeněk Žabokrtský, Alena Koubková and the brilliant Stanford on-line courses on Machine Learning and NLP that are freely available at http://opencourseonline.com/playlist.

Regarding the FOAM method, I appreciate the help of Jiří Vinárek and David Hauzar and also the two other members of the REPROTOOL team – Ondřej Fiala and Rudo Tomori. All the prototyping and brainstorming sessions made FOAM a usable tool.

My thanks also go to the institutions that provided financial support for my research work. Through my doctoral study, my work was partially supported by the Charles University institutional funding and by the Grant Agency of the Czech Republic.

Last but not least, I would like to thank my wife, L'ubka, for her support. I could not have completed this work without her tolerance and patience.

[1] Using a **true** random number generator that utilizes quantum vacuum fluctuations [105]. A block of random samples, downloaded from http://photonics.anu.edu.au/qoptics/Research/qrng.php, was used in the following way: `# sort -R -random-source=qrsamples.txt people.txt`

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague, May 31, 2013 Viliam Šimko

**Název práce:** Z textové specifikace k formální verifikaci

**Autor:** Viliam Šimko

**Katedra:** Katedra distribuovaných a spolehlivých systémů

**Vedoucí práce:** RNDr. Petr Hnětynka, Ph.D

**Abstrakt:** Běžný způsob popisu funkčních požadavků při vývoji softwaru je tvorba textových případů použití (use-cases). Jejich úlohou v úvodních fázích projektu je zachytit formou přirozeného jazyka způsob fungování systému z pohledu koncového uživatele. Protože jde o text psaný v přirozeném jazyce, není možné správnost textových případů použití přímo formálně ověřovat. Obdobně významným artefaktem při vývoji software je doménový model. Jde o popis nejdůležitějších konceptů a vztahů, které jsou pro vyvíjenou aplikaci důležité. Tvorba doménového modelu běžně probíhá iterativně od prvního prototypu z textu až po výsledný formální model. Tato práce se zabývá dvěma souvisejícími tématy – formální ověřování případů použití a odvozování doménového modelu z textu. První část je věnovaná metodě FOAM, která umožňuje pomocí jednoduchých anotací vložených do textu případů použití formálně ověřovat jejich správnost (model-checking). Anotace umožňují zachytit větvení kroků v případech použití a uživatel má možnost vyjádřit časové závislosti mezi různými částmi specifikace, zároveň je však zachovaná srozumitelnost původního textu. Druhá část práce popisuje tzv. Prediction Framework, který pomocí lingvistické analýzy textu a statistických klasifikátorů (log-linear Maximum Entropy models) umožňuje predikování doménového model z textu.

**Klíčová slova:** Verifikace, Požadavky, Formální metody, Modelování

**Title:** From textual specification to formal verification

**Author:** Viliam Šimko

**Department:** Department of Distributed and Dependable Systems

**Supervisor:** RNDr. Petr Hnětynka, Ph.D

**Abstract:** Textual use-cases have been traditionally used at the design stage of the software development process to describe software functionality from the user's perspective. Because use-cases typically rely on natural language, they cannot be directly subject to formal verification. Another important artefact is the domain model, a high-level overview of the most important concepts in the problem space. A domain model is usually not constructed en bloc, yet it undergoes refinement starting from the first prototype elicited from text. This thesis covers two closely related topics – formal verification of use-cases and elicitation of a domain model from text. The former is a method (called FOAM) that features simple user-definable annotations inserted into a use-case to make it suitable for verification. A model-checking tool is employed to verify temporal invariants associated with the annotations while still keeping the use-cases understandable for non-experts. The latter is a method (titled Prediction Framework) that features an in-depth linguistic analysis of text and a sequence of statistical classifiers (log-linear Maximum Entropy models) to predict the domain model.

**Keywords:** Verification, Requirements, Formal Methods, Modeling

# Contents


**1 Introduction** **1**
- 1.1 Specification of functional requirements . . . . . . . . 1
  - 1.1.1 Problem of sequencing demonstrated on an example . . . . . 2
- 1.2 Textual use-cases . . . . . . . . 3
- 1.3 Problem statement and goals . . . . . . . . 4
- 1.4 Summary of contribution and publications . . . . . . . . 5
- 1.5 Structure of the thesis . . . . . . . . 8
- 1.6 Note on conventions used . . . . . . . . 8

**2 History of the FOAM method** **9**
- 2.1 The Procasor tool . . . . . . . . 9
- 2.2 Generating code from use-case specifications . . . . . . . . 9
- 2.3 Generic component model from use-cases . . . . . . . . 10
- 2.4 The REPROTOOL project . . . . . . . . 12
- 2.5 Further development . . . . . . . . 13

**3 Verification of use-cases** **15**
- 3.1 Overview of the FOAM method . . . . . . . . 16
  - 3.1.1 Flow annotations . . . . . . . . 16
  - 3.1.2 Temporal annotations . . . . . . . . 18
- 3.2 Construction of labeled transition systems . . . . . . . . 19
  - 3.2.1 Formalizing the Input Use-Case Model . . . . . . . . 21
  - 3.2.2 Formalizing the Use-Case Behavior Automaton . . . . . . . . 24
  - 3.2.3 Building Use-Case Behavior Automaton – step #1 . . . . . . 24
  - 3.2.4 Building Use-Case Behavior Automaton – step #2 . . . . . . 28
  - 3.2.5 Temporal properties . . . . . . . . 29
- 3.3 Verification using NuSMV . . . . . . . . 30
- 3.4 Expressiveness of FOAM . . . . . . . . 32
- 3.5 Evaluation of scalability . . . . . . . . 34
  - 3.5.1 FOAM scalability experiment 1 . . . . . . . . 36
  - 3.5.2 FOAM Scalability Experiment 2 . . . . . . . . 37
  - 3.5.3 FOAM Scalability Experiment 3 . . . . . . . . 37
  - 3.5.4 FOAM Scalability Experiment 4 . . . . . . . . 37
  - 3.5.5 Summary of the Experimental Results . . . . . . . . 38
- 3.6 Evaluation of learning curve . . . . . . . . 38
  - 3.6.1 Selection of use-cases for the test . . . . . . . . 39
  - 3.6.2 Method applied by independent testers . . . . . . . . 40

3.6.3 Feedback from testers . . . . . . . . . . . . . . . . . . . . 40
Adding flow annotations . . . . . . . . . . . . . . . . . . . 40
Adding temporal annotations . . . . . . . . . . . . . . . . . 41
Summary . . . . . . . . . . . . . . . . . . . . . . . . . 41
3.7 Evaluation of the FOAM tool . . . . . . . . . . . . . . . . . . . . 42
3.8 Implemented FOAM tool . . . . . . . . . . . . . . . . . . . . . . 43
3.9 Summary of Chapter 3 . . . . . . . . . . . . . . . . . . . . . . . 44

**4 Domain model elicitation** **47**
4.1 Domain modeling . . . . . . . . . . . . . . . . . . . . . . . . . . 47
4.1.1 Iterative development and refinement . . . . . . . . . . . . . 48
4.1.2 Grammatical Inspection . . . . . . . . . . . . . . . . . . . . 48
4.2 Natural language processing techniques . . . . . . . . . . . . . . . 49
4.2.1 Linguistic pipeline and common analysis structure . . . . . . 49
4.2.2 Tokenization . . . . . . . . . . . . . . . . . . . . . . . . . 49
4.2.3 Part-of-speech tagging . . . . . . . . . . . . . . . . . . . . 50
4.2.4 Lemmatization . . . . . . . . . . . . . . . . . . . . . . . . 50
4.2.5 Sentence detection . . . . . . . . . . . . . . . . . . . . . . 50
4.2.6 Named entity recognition . . . . . . . . . . . . . . . . . . . 51
4.2.7 Hand-written rules and patterns . . . . . . . . . . . . . . . . 51
4.2.8 Parsing : constituency . . . . . . . . . . . . . . . . . . . . 51
4.2.9 Parsing : dependency . . . . . . . . . . . . . . . . . . . . . 52
4.2.10 Coreference resolution . . . . . . . . . . . . . . . . . . . . 52
4.2.11 Sentence analysis example . . . . . . . . . . . . . . . . . . 52
4.3 Statistical classification related to our method . . . . . . . . . . . . 54
4.3.1 Features . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
4.3.2 Feature extractors and context generators . . . . . . . . . . . 54
4.3.3 Statistical classifier . . . . . . . . . . . . . . . . . . . . . . 55
4.3.4 Training samples . . . . . . . . . . . . . . . . . . . . . . . 55
4.3.5 Training samples in our method . . . . . . . . . . . . . . . . 55
4.3.6 Maximum entropy models for classification . . . . . . . . . . 56
4.3.7 Maximum entropy Markov models . . . . . . . . . . . . . . . 57
4.4 From text to domain model in 4 phases . . . . . . . . . . . . . . . . 57
4.4.1 Preprocessing phase . . . . . . . . . . . . . . . . . . . . . 58
4.4.2 Feature selection phase . . . . . . . . . . . . . . . . . . . . 59
4.4.3 Training phase . . . . . . . . . . . . . . . . . . . . . . . . 60
4.4.4 Domain model elicitation phase . . . . . . . . . . . . . . . . 61
**Step:** Identifying words forming a domain entity . . . . . . . 61
**Step:** Identifying multi-word entities . . . . . . . . . . . . . 61
**Step:** Deriving names for entity links . . . . . . . . . . . . . 61
**Step:** Creating classes in the domain model . . . . . . . . . . 62
**Step:** Merging duplicate classes in the domain model . . . . . 62
**Step:** Predicting relations . . . . . . . . . . . . . . . . . . . 62
4.5 Evaluation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
4.5.1 Training vs. testing data . . . . . . . . . . . . . . . . . . . 63
4.5.2 Cross-validation . . . . . . . . . . . . . . . . . . . . . . . 63
4.5.3 Evaluation metrics in the experiment . . . . . . . . . . . . . 63

4.5.4 Data used in the experiment . . . . . . . . . . . . . . . . . . . 64
4.5.5 Classification in the experiment . . . . . . . . . . . . . . . . . 65
4.5.6 Results of the experiment . . . . . . . . . . . . . . . . . . . . 66
4.6 Summary of Chapter 4 . . . . . . . . . . . . . . . . . . . . . . . . . 68

**5 Related work** **71**
5.1 Systematic reviews . . . . . . . . . . . . . . . . . . . . . . . . . . . 71
5.2 NLP in requirements engineering . . . . . . . . . . . . . . . . . . . 72
5.3 Controlled natural languages . . . . . . . . . . . . . . . . . . . . . 72
5.4 Use-case templates . . . . . . . . . . . . . . . . . . . . . . . . . . . 73
5.5 Extended use-case models . . . . . . . . . . . . . . . . . . . . . . . 74
5.6 Modeling static structures from requirements . . . . . . . . . . . . . 75
5.7 Modeling dynamic structures from requirements . . . . . . . . . . . . 76
5.8 Formal semantics of requirements specification . . . . . . . . . . . . 77
5.9 Consistency of computational models . . . . . . . . . . . . . . . . . 78
5.10 Use of ontologies . . . . . . . . . . . . . . . . . . . . . . . . . . . . 79

**6 Conclusions** **81**
6.1 Future work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 82

**A FOAM case study** **95**
A.1 FOAM case study : Answers . . . . . . . . . . . . . . . . . . . . . . 96

**B Measured prediction performance** **101**

**C Training data used for the evaluation** **105**

# Chapter 1

# Introduction

Formal methods in the recent years have slowly become applicable to industry-size scenarios. More powerful hardware, more sophisticated software made model-checking feasible. However, the main challenge remaining is the involvement of less skilled users. In practice, analysts and software engineers usually avoid advanced techniques in favor of simpler approaches. Widely adopted is the specification of requirements using just natural language, even though it is ambiguous and difficult to be verified for errors. For many year, researches in the field of requirements engineering have tried to tackle challenges of natural language, yet no silver bullet has been found and (probably) never will be.

The simple grammatical inspection approaches from the early days of requirements engineering gradually evolved into sophisticated statistical methods. The advances in computational linguistics played a great role, because it allowed researchers to apply more accurate linguistic taggers, parsers, classifiers and other useful tools that sometimes even surpass the performance of humans as demonstrated by the intriguing IBM Watson project. However, even without a supercomputer at hand, a natural language specification can be semi-automatically transformed into formal models with the aim of minimizing ambiguity and inconsistency of natural language.

## 1.1 Specification of functional requirements

Specification of functional requirements using textual use-cases is a well-established technique in requirements engineering [17]. Neill et. al. [78] reported that over 50% of projects use scenarios or use-cases for requirements elicitation (see Figure 1.1).

A use-case captures a particular functionality of the system textually, as a scenario of actions and responses written in a natural language. The use of the natural language makes textual use-cases an ideal approach for consulting the intended behavior of a developed system, i.e. System Under Discussion (SuD), with the users/stakeholders. Natural language is actually the most widely used form of specification, as can be seen in Figure 1.2. The aforementioned survey [78] also reported that only 7% of requirements are specified formally. The majority of them are rather captured informally (51%) or semi-formally (27%), i.e. in natural language. Another supporting claim for this comes from the survey [64]. Here the authors reported that natural language make 79% of the specifications, 16% is semi-formal or structured language, such as

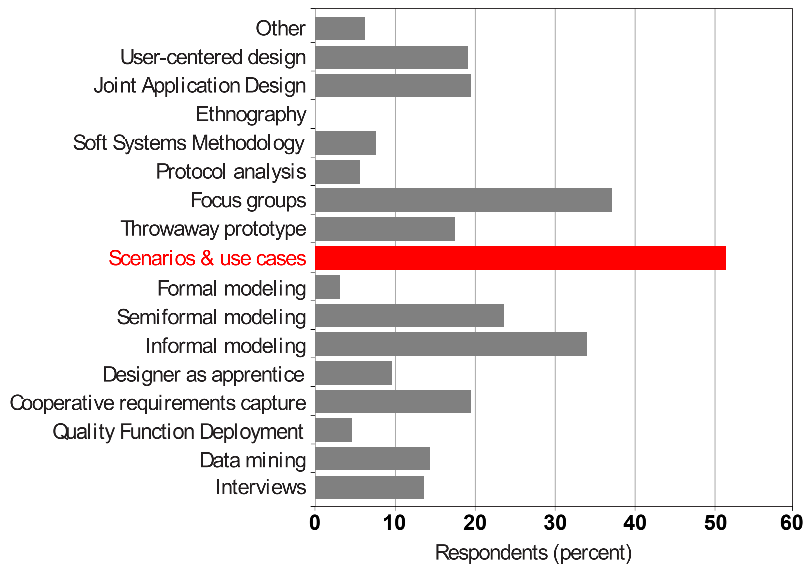
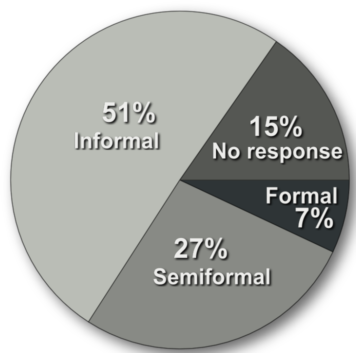
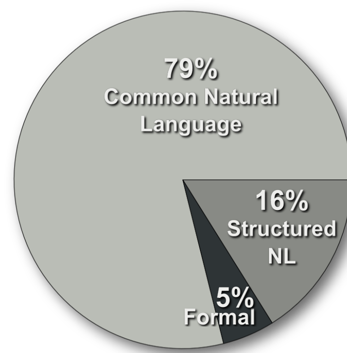



**Figure 1.1:** Techniques used for requirements elicitation. Courtesy of Neill et al. [78].

templates or forms, and only 5% of the requirements are formal.




**According to Neill et al. [78]**




**According to Luisa et al. [64]**

**Figure 1.2:** Requirements formality of modeling notation.

However, the natural language brings the risk of ambiguity or contradiction in specification documents which can negatively impact later phases of the system development. With the increasing complexity of a use-case specification it becomes hard to ensure its validity. Additionally, in a changing environment, the original specification can easily get out-of-sync with the implementation artefacts. Thus a formalization and automated validation of use-cases is desirable. In particular, correct sequencing of use-case actions is a universally relevant property.

### 1.1.1 Problem of sequencing demonstrated on an example

The problem of correct sequencing of actions is demonstrated by the use-cases given in Figure 1.3. The use-cases $u_1$, $u_2$ and $u_3$ operate with the "location", which in this case

is relevant for the sequencing of their actions. In $u_2$ (step 2), the "location" is provided by the server and later being referred in $u_3$ (step 3). Later, during evolution of the specification, new branching conditions can be added changing the normal execution-flow, e.g. by terminating a use-case. In our example (Figure 1.4), the branch Variation $2a$ in $u_2$, leading to an abort, has been introduced in a later project phase. However, in a case this branch executed, the "location" is not provided and it would result in problem inside the use-case $u_3$. Thus, in order to avoid the inconsistency the Extension $1a$ has been added into $u_3$ (Figure 1.5). Even from such a small example, it is obvious that manual detection of such inconsistencies is almost impossible (especially in projects where the use-cases are prepared by a group of analysts). Therefore, an automated verification mechanism is in fact a necessity.

**UseCase**:$\mathbf{u}_1$ Select city on map
1. The user opens the map web page.
2. The system generates a map with available cities.
3. The user selects a city on the map.

---

**UseCase**:$\mathbf{u}_2$ Generate city
**Preceding**:$\mathbf{u}_1$ "Select city on map"
1. The system asks MapServer to provide city information.
2. MapServer provides the requested information. (providing the location)
3. The system generates the map with default zoom settings.
4. User adjusts zoom settings.

---

**UseCase**:$\mathbf{u}_3$ Generate restaurant map for city
**Preceding**:$\mathbf{u}_1$ "Select city on map"
1. Include use−case "Generate city".
2. System validates the zoom settings.
3. System asks RestaurantServer for restaurants. (near the given location)
4. RestaurantServer generates the restaurant layer information.
5. System generates restaurant map.

**Figure 1.3:** Example of use-cases sharing an artefact relevant for the sequencing of actions.

## 1.2 Textual use-cases

A use-case describes a particular functionality of a system in natural language. This makes use-cases a very advantageous asset for communicating software specification with customers as well as with developers, however the use of natural language is also the main obstacle in automated processing and verification of use-cases. This is not just because of the intricacies interpreting the text in the natural language, but also because, to date, there is no standardized form of use-cases. To overcome the latter, we

**UseCase**:$\mathbf{u}_2$ Generate city
**Preceding**:$\mathbf{u}_1$ "Select city on map"
1. The system asks MapServer to provide city information.
2. MapServer provides the requested information. (providing the location)
3. The system generates the map with default zoom settings.
4. User adjusts zoom settings.

Variation: 2a. MapServer error occurred.
2a1. Use-case aborts.

**Figure 1.4:** An inconsistency introducing variation added to the specification.

**UseCase**:$\mathbf{u}_3$ Generate restaurant map for city
**Preceding**:$\mathbf{u}_1$ "Select city on map"
1. Include use−case "Generate city".
2. System validates the zoom settings.
3. System asks RestaurantServer for restaurants. (near the given location)
4. RestaurantServer generates the restaurant layer information.
5. System generates restaurant map.

Extension: 1a. There was an abort in "Generate city".
1a1. Use-case aborts.

**Figure 1.5:** An inconsistency resolving extension added to the specification.

adhere to the widely accepted format proposed in [17]. The former is however much more serious problem, which we tackle in our method by introduction of annotations as explained further in the text. For the reader's convenience, we briefly summarize this use-case format below.

Typically the system under discussion is specified as a set of use-cases (further denoted as UCM, i.e. Use Case Model). A single use-case always specifies the *main scenario* and a (potentially empty) set of *branching scenarios*. Each scenario comprises a sequence of *use-case steps*. A use-case step, written as a simple sentence in a natural language (English in our case), expresses an interaction between SuD and actors (typically users and stakeholders). A use-case step is identified by its sequence number. The main scenario (also called success scenario) defines the sequence of interactions for achieving the goal of the use-case (e.g. steps 1-3 in Figure 1.5). A branching scenario is either *variation* or *extension* of a particular use-case step. An extension enhances specification of the particular step while a variation is an alternative to the step's specification. The correspondence of a variation or an extension to a step is given by referring to the step's sequence number (e.g. variation $2a$ is an alternative to step 2 in the use-case $u_2$).

Use-cases can also be involved in a *precedence relation* [11, 3, 29], which constraints their sequencing (e.g. in Figure 1.5, before the use-cases $u_2$ or $u_3$ can be executed the use-case $u_1$ has to be executed first).

## 1.3 Problem statement and goals

Requirements engineering is about managing artefacts during the software development process and to facilitate change requests. Artefacts such as textual specification, models or code, are interconnected. If we capture these relations in a consistent and accessible way, we can easily extract valuable statistical information to predict new artefacts. If we build appropriate formal models out of these artefact, we can also verify certain consistency properties.

The focus of this thesis is to propose methods that leverage existing formal methods (verification of behavior of a system) to requirements engineering (in particular use-cases and domain models) by using existing linguistic tools (text analysis, coreference resolution, relation extraction). The emphasis is on practical usability of the designed method. The main goals are:

**G1** *Formalize behavior of textual use-cases.*
Design a lightweight method for capturing behavior of use-cases and expressing temporal constraints in their control flow. The method should integrate well with specifications written in natural language. Even users not familiar with temporal logic should be able to describe temporal constraints among use-case steps in an intuitive way, while at the same time, the formalism should be extensible enough so that advanced users can define their own consistency constraints using temporal logic.

**G2** *Design a method for Verification of use-cases.*
Define mapping from the formalism mentioned above to a model which can be verified by existing model-checkers. The method should be also evaluated from the scalability and usability point of view.

**G3** *Combine linguistic and software engineering artefacts.*
Design a model and method for: (i) automated linguistic analysis of textual specifications, (ii) combining the extracted linguistic information with other software engineering artefacts, (iii) automated extraction of features, training a statistical classifier and evaluation of the prediction performance. (iv) use the classifier to derive a prototype domain model from natural language. This thesis is focused only on specification texts written in natural language, domain models (similar to class diagrams) and textual use-cases.

## 1.4 Summary of contribution and publications

All topics discussed within this thesis are supported by peer-reviewed publications published on international conferences or submitted to a journal/conference. In all of the papers listed below, I carried out most of the work covering the initial design, experiments, coding and case studies with the help of the co-authors. Part of the implementation was also carried out by students in the REPROTOOL [89] software project, which I supervised. Moreover, I also supervised the master thesis of Jiri Vinarek, who substantially contributed to the implementation of the *FOAM tool*.

My very first motivation for verification of use-cases was formed during my participation in projects OASIS[1] and GAMA[2] [32, 63, 28]. These EU-funded projects were focused on the establishment of a central platform to enable multilingual, facilitated and user-orientated access to a significant number of media art archives and their digitalised contents. The architecture was basically a network of independent database adapters connected through a central repository which aggregates metadata from the archives into a common schema. The projects followed an iterative approach – requirements specification, coding, testing and other phases of the project's lifecycle were revisited multiple times in order to support an open user-centric design. Interaction among the components was specified as scenarios. However, no formal approach

[1] http://oasis-archive.eu/
[2] http://gama-gateway.eu

was employed to verify the correctness of such a specification. Although we made a great effort to inspect the scenarios visually, we always felt unsure about the correctness of the system's behavior. An example could be a process of populating the central repository with fresh data from the remote databases. The process involved cooperation among multiple components described using scenarios. However, we could not tell whether the specification permits any unforeseen/undesirable states.

After successfully finishing the GAMA project, I started the Ph.D. studies focusing on verification of use-cases. First, the theoretical concepts of the FOAM method were drafted in the papers [92, 91]. Later, based on the experience with the tool implementation, I decided to redefine the original UCM→LTS transformation. The new transformation was published in [93]. However, the approach still needed a proper evaluation of scalability and a case study which was finally carried out in [94]. Moreover, the implementation of the *FOAM tool* was finished and refactored into cleanly separated modules. As a next step, the approach was extended to the realm of machine learning and information extraction. The paper [96] describes a method for deriving prototype domain model from textual specification using statistical classification approach. A substantial part of the work was the implementation of a framework for automated evaluation of the tool's classification performance.

## Reviewed publications:

**[93]** V. Simko, P. Hnetynka, T. Bures, and F. Plasil. "FOAM: A Lightweight Method for Verification of Use-Cases". In: *Software Engineering and Advanced Applications (SEAA), 38th EUROMICRO Conference on*. 2012, pp. 228–232. DOI: 10.1109/SEAA.2012.15

**[91]** V. Simko, D. Hauzar, T. Bures, P. Hnetynka, and F. Plasil. "Verifying Temporal Properties of Use-Cases in Natural Language". In: *Postproc. of FACS'2011*. LNCS. Springer, 2011. DOI: 10.1007/978-3-642-35743-5_21

**[92]** V. Simko, P. Hnetynka, and T. Bures. "From Textual Use-Cases to Component-Based Applications". In: *Proc. of SNPD'10*. Vol. 295. Studies in Computational Intelligence. Springer, 2010, pp. 23–37. ISBN: 978-3-642-13264-3. DOI: 10.1007/978-3-642-13265-0

**[32]** A. Glowacz, M. Grega, M. Leszczuk, Z. Papir, P. Romaniak, P. Fornalski, M. Lutwin, J. Enge, T. Lurk, and V. Simko. "Open internet gateways to archives of media art". In: *Multimedia Tools and Applications* (Mar. 2011), pp. 1–24. ISSN: 1380-7501. DOI: 10.1007/s11042-011-0784-3

**[63]** A. Ludtke, B. Gottfried, O. Herzog, G. Ioannidis, M. Leszczuk, and V. Simko. "Accessing Libraries of Media Art through Metadata". In: *2009 20th International Workshop on Database and Expert Systems Application*. IEEE, 2009, pp. 269–273. ISBN: 978-0-7695-3763-4. DOI: 10.1109/DEXA.2009.93

**[28]** J. Enge, A. Glowacz, M. Grega, M. Leszczuk, Z. Papir, P. Romaniak, and V. Simko. "OASIS Archive – Open Archiving System with Internet Sharing". In: *Future Multimedia Networking*. Vol. 5630. LNCS. Springer, 2009, pp. 254–259. ISBN: 978-3-642-02471-9. DOI: 10.1007/978-3-642-02472-6_28

## Technical reports:

**[96]** V. Simko, P. Kroha, and P. Hnetynka. *Implemented Domain Model Generation*. Tech. rep. 2013/3. D3S, Charles University in Prague, Apr. 2013. URL: http://d3s.mff.cuni.cz/publications/download/D3S-TR-2013-03.pdf
**A paper based on this work was recently submitted to a conference.**

**[94]** V. Simko, P. Hnetynka, T. Bures, and F. Plasil. *Formal Verification of Annotated Use-Cases (FOAM Method)*. Tech. rep. 2012/2. D3S, Charles University in Prague, 2012. URL: http://d3s.mff.cuni.cz/publications/download/D3S-TR-2012-02.pdf
**A paper based on this work was recently submitted to a journal.**

**[95]** V. Simko, P. Kroha, and P. Hnetynka. *Domain Model Generation With the Help of Supervised Machine Learning*. Tech. rep. 2012/6. D3S, Charles University in Prague, 2012. URL: http://d3s.mff.cuni.cz/publications/download/D3S-TR-2012-06.pdf

**[12]** T. Bures, P. Hnetynka, P. Kroha, and V. Simko. *Requirement Specifications Using Natural Languages*. Tech. rep. 2012/5. D3S, Charles University in Prague, 2012. URL: http://d3s.mff.cuni.cz/publications/download/D3S-TR-2012-05.pdf

**[90]** V. Simko. *Patterns In Specification Documents*. Tech. rep. 2011/6. D3S, Charles University in Prague, 2011. URL: http://d3s.mff.cuni.cz/publications/download/D3S-TR-2011-06.pdf

## Implemented tools:

**FOAM tool:** This is a command-line tool that performs verification of use-case models annotated with FOAM annotations. The architecture of the tool is modular and extensible. Each transformation phase is clearly separated with well-defined meta-models describing its inputs and outputs. The demo application shows how a valid specification and also an invalid specification is verified. At the time of writing this text, the tool is available for download at:

`http://vlx.matfyz.cz/Projects/FoamTool`

**Prediction Framework:** The framework is a headless Eclipse-based product composed of multiple OSGi bundles. Some of the bundles contain linguistic and statistical models. The application demonstrates 4 phases: (1) preprocessing, (2) feature selection, (3) training, and (4) elicitation. At the time of writing this text, the implementation is available for download at:

`http://vlx.matfyz.cz/Projects/PredictionTool`

## 1.5 Structure of the thesis

The next **Chapter 2** summarizes the history of the FOAM method presented in the rest of the text. Afterwards, the text continues by the core **Chapters 3 and 4**.

**Chapter 3** is based on publications [91, 93, 94]. It focuses on verification of textual use-cases using the *FOAM tool*. The chapter starts by explaining what kind of textual use-cases we consider for verification (Section 1.2). Then, in Sections 3.1 and 3.2, the theoretical background of FOAM method is explained. In Section 3.4 we provide proofs about expressiveness and correctness of the proposed method. In Sections 3.5, 3.6 and 3.7 discuss the scalability, learning curve and applicability in practice.

**Chapter 4** is based on publications [12, 90, 92, 95, 96]. It elaborates on the approach to predicting software engineering artefacts using a Maximum Entropy (MaxEnt) statistical classifier. In particular, we demonstrate the idea on a method for predicting a prototype domain model from English text. Sections 4.2 and 4.3 provide an overview of the theory and tools regarding NLP and statistical classification. Section 4.4 describes all phases: (1) Preprocessing, (2) Feature selection, (3) Training, (4) Elicitation. Section 4.5 summarizes the experimental results of 3 classification models selected for the demonstration.

The thesis then continues by **Chapter 5**, which summarizes the related work for both main topics together. Finally, **Chapter 6** concludes the thesis.

It should be noted that the reviewed papers [28, 32, 63] mentioned above are not included to the following chapters because they merely motivated the research around FOAM.

## 1.6 Note on conventions used

In order to distinguish between a text included verbatim from the aforementioned publications, a colored vertical bar on the right or left side of the text is used. Where appropriate, I have slightly modified and unified the original text (in a way not changing the meaning) to make the thesis coherent and easy to read.

Here is an example of how paragraphs are marked to indicate a verbatim copy. It means that from this point, the text appeared in the papers [X] and [Y].

# Chapter 2

# History of the FOAM method

## 2.1 The Procasor tool

The *Procasor tool* (elaborated in [73, 25, 74]) was previously developed in our research group. It was an attempt to support the analyst while specifying functional requirements in the form of textual use-cases. Procasor takes use-cases in plain English and transforms them into a formal behavior specification called *procases* [82]. The procase is a variant of a *behavior protocol* [81] which is a process algebra precisely describing the behavior of a system or components.

The central part of the Procasor tool is an interactive use-case editor [74] integrated with linguistic tools to give an immediate feedback to the writer. The tool can generate two types of formal specification – *behavior protocols* and *UML state machines*. A typical textual use-case analyzed by Procasor together with its corresponding procase is depicted in Figure 2.2.

When writing use-case steps, each sentence is linguistically analysed. The tool acquires information about the action expressed by the sentence together with the communicating parties involved in the action. To do that, Procasor needs a *domain model* to be specified manually beforehand. Domain model describes entities appearing in the system being developed, in Procasor it is captured as a simple list of entities.

The linguistic analysis employed in Procasor uses COLLINS statistical parser [18] developed by M. Collins at the University of Pennsylvania in 1996. The parser produces constituency parse trees that break the sentence into phrases. An example of such a parse tree is depicted in Figure 2.1. Procasor expects a uniform sentence structure as suggested by well known use-case approaches [17], [55], [58]. It uses a fixed set of rules for matching patterns in the generated parse tree.

## 2.2 Generating code from use-case specifications

In the work [30], the Procasor tool has been further extended by a generator that can produce an executable application from use-cases. Input to the generator were *procases* from Procasor and *domain model* of the designed system, prepared manually, encoded as a UML class diagram (see Figure 2.3). The domain model consists of: (i) conceptual classes, (ii) attributes of conceptual classes, and (iii) associations among conceptual classes.

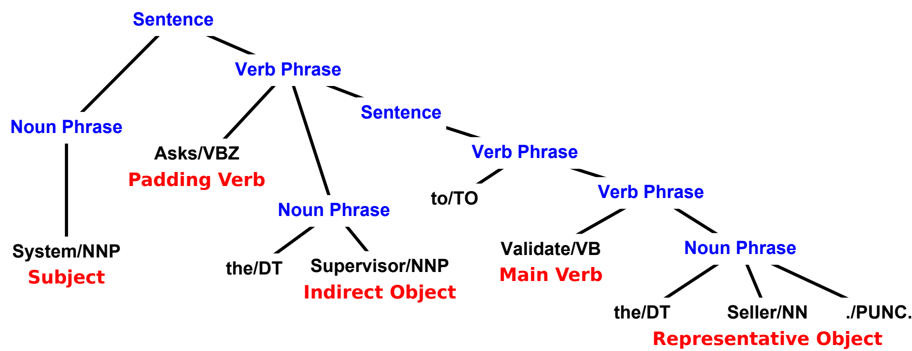



**Figure 2.1:** Example parse tree for the sentence *"System asks the Supervisor to validate the Seller."* as used by the Procasor tool. It is a constituency parse tree, where non-terminals are the phrase types (such as Noun Phrase, Verb Phrase) and terminals are individual words.

Outputs from the generator were[1] (i) objects capturing the work-flow of each use-case (Controller), (ii) web pages acting as GUI to the user (View), and (iii) objects representing the business objects (Model). The intended use of the generated implementation was to let users test the traces expressed by the use-cases. Only a secondary goal was to generate code skeletons for the actual implementation.

The created application can be used for three purposes as depicted in Figure 2.4: (i) to immediate evaluation of completeness of the use-cases, (ii) to obtain first impression by application users, and (iii) as a skeleton to be extended into the final application.

In the MDD terminology, the use-cases and domain model serve as a platform independent model (PIM), which are transformed via intermediate models into an executable code, i.e. platform specific model (PSM).

## 2.3 Generic component model from use-cases

[92]

The aforementioned generator targeted Java Enterprise Edition (JEE) applications only. Therefore, we aimed at solving this limitation in the paper [92]. We proposed a transformation pipeline (Figure 2.5) which generates a generic component-based architecture from use-cases. In our approach, we proposed an intermediate component-model, independent of any particular component framework, from which multiple back-ends could generate platform specific code. Our pipeline had two extension points. (i) The input filters in the front-end that allow for a different input format, (ii) the transformations in the back-end in order to add support of different programming languages and platforms, in which the final application can be produced.

The front-end consisted of the Procasor tool, however, modified in order to accept different formats. The following transformation steps were executed based on the generated procases and the domain model:

1. Procedures were identified in the generated procases. A procedure is a sequence of actions, which starts with a request receive action and continues with other

[1] Using the terminology of the Model-View-Controller design pattern

```
UseCase: Clerk submits an offer on behalf of a Seller

Scope: Marketplace
SuD: Computer System
Primary actor: Clerk
Supporting actor: Trade Commission
Supporting actor: Supervisor
Supporting actor: Seller

Main success scenario specification:
    1. Clerk submits information describing an item
    2. System validates the description
    3. Clerk adjusts/enters price and enters seller's
         contact and billing information
    4. System validates the seller's contact information
    5. System asks the Supervisor to validate the seller
    6. Supervisor permits the seller to operate on the
         marketplace
    7. System validates the whole offer with the Trade
         Commission
    8. System lists the offer in published offers
    9. System responds with an uniquely identified
         authorization number
Extensions:
    2a. Validation performed by the system fails
        2a1. Use case aborted
    7a. Trade commission rejects the offer
        7a1. Use case aborted
Sub−variations:
    2b. Price assessment available
        2b1. System provides the seller with a price
               assessment
```

```
PROCASE: Clerk submits an offer on behalf of a Seller

?CL.submitItemDescription;
#validateDescription;
#validationPerformedSystemFails;
%ABORT
+
(
    ?CL.submitItemDescription;
    #priceAssessmentAvailable;
    !SI.providePriceAssessment
    +
    ?CL.submitItemDescription;
    #validateDescription
);
?CL.enterPriceContactBillingInformation;
#validateContactInformation;
!SU.validateSeller;
?SU.permitSeller;
!TC.validateOffer;
(
    #listOffer;
    !SI.respondUniqAuthNumber
    +
    #tradeCommissionRejectsOffer;
    %ABORT
)
```

**Figure 2.2:** Example input and output format used by the Procasor tool. Input is a textual use-case. Output is a behavior protocol describing the use-case formally. Both of them were taken from the Marketplace System case study by Plasil et al. [82])

action than request receive. In the final code, these procedures correspond to the methods of generated objects.

2. Arguments of identified procedures and data-flow among them were inferred from the domain model. These arguments eventually result into methods arguments in the final code and also into objects' allocations.

3. The generic component model was generated.

4. Finally, executable code is generated from the generic component model.

**The generic component model consists of:**

**Entity Data Objects** representing entities from the domain model. Because the domain model and the use-cases do not provide enough information to generate the internal logic of the domain entities (e.g. validation of an item), we generate only skeletons that log their activity. This means that the generated application can be launched without manual modification and by itself provides traces, that give valuable feedback.

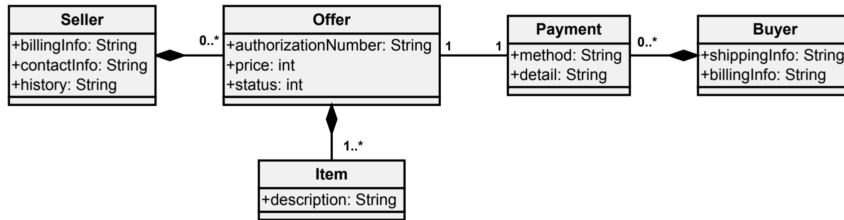



**Figure 2.3:** Example domain model used as an input to the JEE generator, taken from [30]. The domain model had to be created manually.

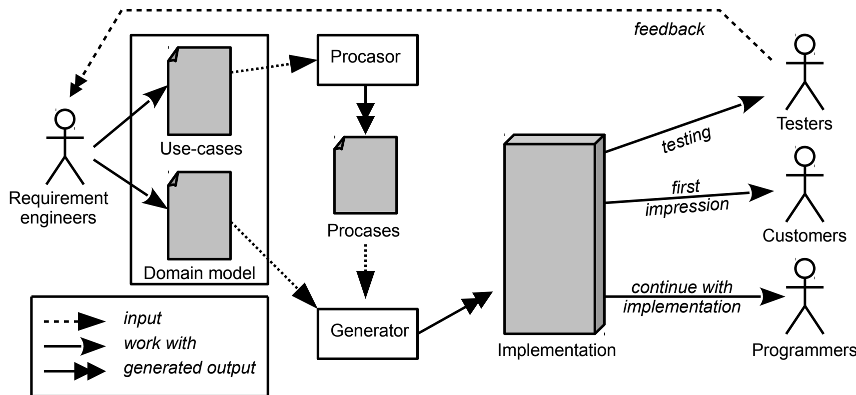



**Figure 2.4:** System usage overview of the J2EE generator.

**Use-Case Objects** that contain the business logic of the corresponding use-cases, i.e. the sequence of actions in the use-case success scenario and all possible alternatives.

**User Interface Objects** that represent the interactive part of the final application. The users can interact via them with the application, set required inputs, and obtain computed results.

**Procedure Signatures** that represent the interface to Entity Data Objects.

## 2.4 The REPROTOOL project

In 2010, we decided to improve the Procasor tool in the following ways:

1. We wanted to refactor the implementation into separate components featuring explicitly-defined interfaces. We decided to build the new framework on Eclipse, where each component would become an OSGi bundle (Eclipse plug-in).

2. We wanted to refresh the Procasor's old linguistic tool-set. Each linguistic task would be implemented by a separate component. We also wanted to develop multiple components implementing the same functionality, such as Part-of-speech tagging. Users would be able to arbitrarily switch between alternative implementations.

3. We aimed at improving the use-case editor.

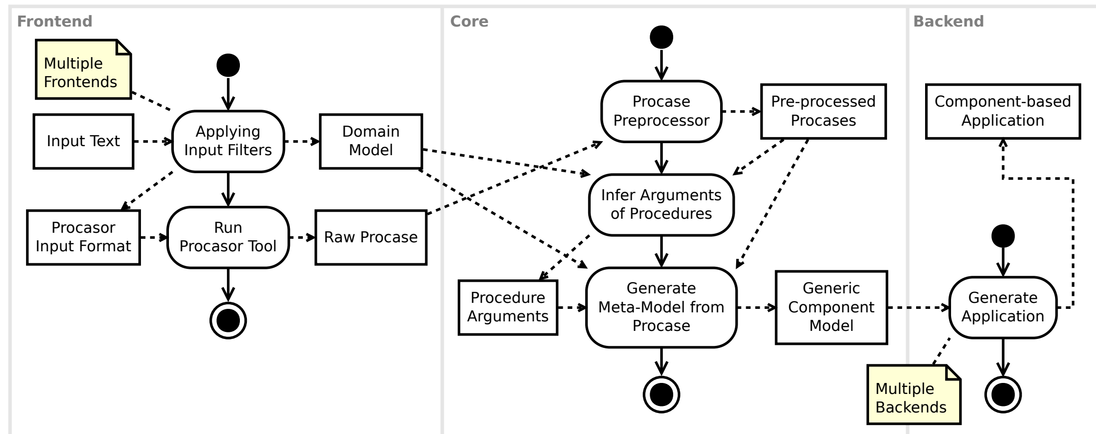



**Figure 2.5:** Architecture of the uc2comp generator. The architecture supported multiple front-ends and back-ends.

4. Finally, we wanted to integrate verification with the editor. For this task, we chose the NuSMV model checker.

These ideas were implemented within the REPROTOOL [89] software project[2]. The developed tool covered almost all the points above. However, it was only usable as a proof-of-concept that model-checking can be successfully integrated with textual use-cases. The strongest part of the developed application was the verification aspect which integrates NuSMV model-checker. The weakest part was the graphical use-case editor. Eventually, it turned out that users preferred their favourite textual editor for writing use-cases instead of the developed graphical editor. The main ideas employed in REPROTOOL were outlined in our paper [91].

While the main focus in Procasor was on the transformation of natural language into a formal model, in our new method, we focused more on the verification aspect. Even so, we still wanted to preserve the advantages of natural language specification. Therefore, we introduced annotations that can be unobtrusively inserted into the text to capture the behavior expressed by sentences of a use-case. Another important concept that we introduced, and which Procasor did not consider, were the *precedence* and *include* relations among use-cases. Correct sequencing of actions suddenly became a non-trivial problem; naturally a task for a model-checker.

## 2.5 Further development

Based on the experience with REPROTOOL implementation, we decided to concentrate our effort on verification, which, as demonstrated in the following chapter, paid off later in the FOAM project. We have further improved the transformation pipeline from annotated textual use-cases to the formal model. The new transformation was published in [93]. Recently, after finishing the implementation of the FOAM tool, we also conducted an evaluation of scalability, learning curve and applicability of the method. The results were published in [94].

[2]"Software project" is a mandatory requirement for undergraduate students at our university. They have to implement a working application collaboratively in teams of 4-6 people.

The last remaining part was to reintroduce the linguistic aspect to the pipeline. Our FOAM method presumes that the input text is already annotated. From the FOAM's point of view, all the information is encoded in the annotations and the text acts only as a carrier of annotations.

The next logical step was to design a mechanism that would semi-automatically make predictions from the text, such as suggesting where the annotations should be placed. Also remember that Procasor and the JEE generator mentioned above required a domain model, usually prepared manually by the user. To tackle these issues, we designed a framework for predicting software engineering artefacts using statistical classification. Since this is a rather broad task, we decided to focus on the prediction of a domain model from text, which is summarized in the paper [96]. However, the developed framework can be easily extended in order to predict other artefacts, such as FOAM annotations.

The next two chapters describe in detail both the FOAM method (Chapter 3) and the elicitation of a domain model (Chapter 4).

# Chapter 3

# Verification of use-cases

This chapter elaborates on a method allowing for verification of correct sequencing of actions in use-cases (The FOAM method – Formal Verification of Annotated Use-Case Models). The FOAM method works with use-cases in their natural language form requiring only a few basic annotations to be inserted in the use-cases to capture the semantics.

In [91], we have introduced the use of *annotations* to formalize the use-case semantics and to select properties that are subject to verification. The approach was further elaborated in [93], where we included user-definable properties (as opposed to pre-defined set of properties that could have been verified by [91]). The majority of the text within this chapter is taken from [94]. We extended the ideas and corrected former drawbacks in the following way:

1. All transformations are now formally defined which is necessary for a correct implementation. The transformation is defined as a set of inference rules, independent on any particular implementation – Section 3.2.
2. We proved that the method has a sufficient expressive power to encode a sufficiently general Kripke structure and a related temporal logic formula – Section 3.4.
3. We improved the scalability with respect to real-life use-cases Section 3.2.1. Now the method scales linearly with the number of use-cases.
4. We evaluated the scalability of FOAM and explored its theoretical limits with respect to the NuSMV model-checker – Section 3.5.
5. We evaluated the usability of FOAM in a case study – Section 3.7.
6. And finally, we finished the implementation of the FOAM tool[1] – Section 3.8.

[1]Example output from the tool: http://foam-tool.appspot.com/overview/overview.html

[94]
[93]

## 3.1 Overview of the FOAM method

As discussed above, ensuring the correctness of an evolving specifications is hard. It would be a mistake to understand requirements documents as final and unchangeable as emphasized in [58]. Automation is required especially for large and complex specifications where manual reviewing becomes tedious. As a first step in the automation process, it is necessary to extract the control flow from the use-cases. Next, we need to formally define conditions to be verified. All this information is available in the use-cases but it is "hidden" in the natural language of the text. To overcome this, we propose to enhance the use-cases with *annotations*, which are short tags appended to a particular use-case step sentence (blue tags **#($a$:$s$)** in Figure 3.2).

**These annotations can be divided into two groups:**

(1) *flow annotations* expressing control flow of use-cases, and

(2) *temporal annotation* expressing conditions to be verified.

In an ideal case, these annotations are also automatically added to the use-case, but, for now, our method expects that they are manually added by the analysts and automation of this step is left for future work.

The verification itself then takes the annotated use-cases and automatically transforms them into Labeled Transition System (LTS). This LTS (i.e., an automaton) encodes the execution of all scenarios expressed by the use-cases. The process further continues by transforming LTS into the input for NuSMV model checker [15], which verifies the use-cases. Transformations are transparent to the user; the potential errors reported by NuSMV are presented in a natural language by translating the counter-example to the steps of the flawed use-case. The whole process is sketched in Figure 3.1. In the rest of this section we describe the used annotations in detail.

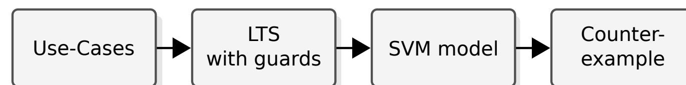



**Figure 3.1:** Overview of the verification method

### 3.1.1 Flow annotations

Execution of a use-case starts with the first step of its main scenario and then continues till the end possibly visiting optional branches. However, the control flow of the execution can be further altered by: (1) *aborts* which prematurely end the scenario – typically as a reaction to an error; (2) *includes* which incorporate (inline) another use-case in the place of a particular step, (3) *jumps* which move execution to a specified use-case step, and (4) *conditions* of extensions and variations.

All these constructs are written in a natural language. FOAM considers them the core concepts influencing the control flow and captures them formally using annotations of the following form:

**#(abort)** : This annotation expresses abort of the scenario.

**UseCase**:$\mathbf{u}_1$ Select city on map
1. The user opens the map web page.
2. The system generates a map with available cities.
3. The user selects a city on the map. **#(create:city)**

**Variation**: 2a. No cities available.
2a1. System displays an empty map with message.
2a2. Use−case aborts. **#(abort)**

---

**UseCase**:$\mathbf{u}_2$ Generate city
**Preceding**:$\mathbf{u}_1$ "Select city on map"
1. The system asks MapServer to provide city information. **#(use:city)**
2. MapServer provides the requested information. **#(create:location)**
3. The system generates the map with default zoom settings.
4. User adjusts zoom settings.

**Variation**: 2a. MapServer error occurred.
2a1. Use−case aborts. **#(abort)**

---

**UseCase**:$\mathbf{u}_3$ Generate restaurant map for city
**Preceding**:$\mathbf{u}_1$ "Select city on map"
1. Include use−case "Generate city". **#(include:GenerateCity)**
2. System validates the zoom settings.
3. System asks RestaurantServer for restaurants. **#(use:city)**, **#(use:location)**
4. RestaurantServer generates the restaurant layer information.
5. System generates restaurant map.

**Extension**: 1a. There was an abort in "Generate city". **#(guard:abort)**
1a1. Use−case aborts. **#(abort)**

**Extension**: 2a. Zoom settings are invalid.
2a1. System display an error message to the user.
2a2. System uses the default zoom settings.

---

The elements denoted as **#(a:s)** are examples of annotations in FOAM.
– "$a$" is the name of the annotation
– "$s$" is the qualifier of the annotation

**Figure 3.2:** Example of a consistent specification using annotated use-cases.

**#(goto:$s$)** : This annotation represents a jump within the use-case. The parameter $s$ indicates the target use-case step of the jump.

**#(include:$u$)** : This annotation specifies inclusion (inlining) of another use-case $u$.

The following annotations **#(mark)** and **#(guard)** assume existence of boolean variables $b_1, \ldots, b_n$ initialized to $false$ (globally accessible in UCM).

**#(mark:$b_i$)** : This annotation sets $b_i$ to $true$.

**#(guard:$f(b_1, \ldots, b_k)$)** : The $f$ parameter of this annotation is a propositional logic formula over the boolean variables $b_1, \ldots, b_k$. The annotation serves as a guard for extensions and variations. That is, a variation/extension can be followed only if the formula is true. Notice how guard annotations are used in Figure 3.2. As a syntactic sugar, we support the definition of **#(guard:P)**, where the pattern $P$ is used for matching other annotations in the specification (including wildcards). In our example the **#(guard:abort)** annotation implies that **#(mark)** is automatically added to step $2a2$ in $u_1$, step $2a1$ in $u_2$ and step $1a1$ in $u_3$.

### 3.1.2 Temporal annotations

Temporal annotations allow expressing temporal invariants among use-case steps in the whole Use-Case Model (UCM) without requiring an in-depth knowledge of the underlying temporal logic (CTL or LTL). In Fig. 5 these annotations are: **#(create:city)**, **#(use:city)**, **#(create:zoom)**, **#(use:zoom)**. FOAM allows these annotation to be user-defined. In particular, it distinguishes two types of users:

*(a) experts in temporal logic* who prepare templates of annotations in the FOAM's Temporal Annotation Definition Language (TADL), i.e. our language for template definition (for an example of it see Figure 3.3),

*(b) domain engineers* who refer to the names of these templates when associating use-case steps with annotations (Figure 3.2). For this activity detailed knowledge of temporal logic is not necessary.

Specifically, when an annotation **#(x:y)** appears in a specification, the TADL definition for $x$ is used to convert it into a set of temporal formulae (where $x$ is substituted by $x_y$). The transformation is described in detail in Section 3.2.5.

TADL defines a group of related temporal annotations along with their semantics expressed as a set of temporal logic formulae in CTL, LTL or PLTL, which is a convenient LTL extension related to the past rather than future. These formulae can be connected with usual propositional logic operators such as $\neg\varphi$ (Negation), $\alpha \mid \beta$ (OR), $\alpha \,\&\, \beta$ (AND), $\alpha \rightarrow \beta$ (Implication). Temporal constraint expressed by the formula is also written down in a human-readable form for error reporting (when showing a counter-example to the user). The formulae need to be transformed into a representation supported by the particular model-checking back-end, in our case to NuSMV. (see Section 3.3) Here is the complete list of all the temporal operators supported by the FOAM verification tool:

[94]

**Supported CTL operators:**

$AX(\varphi)$ : $\varphi$ holds on all paths in the next state.

$AG(\varphi)$ : $\varphi$ holds on all paths in all states.

$AF(\varphi)$ : On all paths there is some state in future where $\varphi$ holds.

$A[\alpha\ U\ \beta]$ : On all paths, at some point $\beta$ holds while in the meantime $\alpha$ holds.

$EX(\varphi)$ : There is a path on which $\varphi$ holds in the next state.

$EG(\varphi)$ : There is a path on which $\varphi$ holds globally.

$EF(\varphi)$ : There is a path with a state in future where $\varphi$ holds.

$E[\alpha\ U\ \beta]$ : There is a path where at some point $\beta$ holds while in the meantime $\alpha$ holds.

**Supported LTL operators (related to future):**

$X(\varphi)$ : "Next", i.e. $\varphi$ holds in the next state.

$G(\varphi)$ : "Globally", i.e. $\varphi$ holds in all states.

$F(\varphi)$ : "Future", i.e. there is a state in future where $\varphi$ holds.

$[\alpha\ U\ \beta]$ : "Until", i.e. at some point $\beta$ holds while in the meantime $\alpha$ holds.

$[\alpha\ R\ \beta]$ : "Release", is equivalent to $\neg[\neg\alpha\ U\ \neg\beta]$.

**Supported PLTL operators (related to past):**

$Y(\varphi)$ : "Yesterday", i.e. $\varphi$ holds in the previous state.

$H(\varphi)$ : "Historically", i.e. $\varphi$ holds on all states in the past.

$O(\varphi)$ : "Once", i.e. $\varphi$ holds in some state in the past.

$\alpha\ S\ \beta$ : "Since" is a temporal dual of U (until), so that $\beta$ holds somewhere in the past and $\alpha$ is true from then up to now.

$\alpha\ T\ \beta$ : "Triggered" is the temporal dual for R, which means that: $[\alpha\ T\ \beta] \equiv \neg[\neg\alpha\ S\ \neg\beta]$

Let us now examine the example in Figure 3.3 in more detail. There are three temporal annotation groups defined here. The "create, use" annotations allow expressing constraints on ordering the use-case steps. For instance, in Figure 3.2, the step 1 of the use-case $u_2$ annotated with **#(use:city)** should be executed only if there was a previously executed step with the **#(create:city)** annotation. Additionally, there should not be an execution with several **#(create:city)** annotations. The transformation of these annotations would result in the following set of formulae:

```
LTL G(use_city → O(create_city)) "There must be create before use"
CTL AG( create_city → EF(use_city) ) "At least one branch with use required after create"
CTL AG( create_city → AX(AG(¬create_city)) ) "Only one create"
```

The templates "open, close" and "init, process, release" in Figure 3.3 illustrate how more complex annotations can be defined in TADL (a strict ordering of 2 phases and then of 3 phases with the possibility to be extended to N phases).

## 3.2 Construction of labeled transition systems

In this section we explain details of the transformations from annotated textual use-cases into formal structures that can be automatically verified. The whole process is depicted in Figure 3.4. The input is the Use-Case Model (UCM) – a collection of annotated textual use-cases.

Some use-cases are selected by the analyst as "primary", which means that they can be directly executed. An example would be the use-case MOD1_UC1 (Figure A.2) from our case-study describing how users register to the system. The behavior of this use-case is self-contained and it makes perfect sense to execute it as a separate scenario.

**Annotations**: create, use
**LTL G**(use → **O**(create)) "There must be create before use"
**CTL AG**( create → **EF**(use) ) "At least one branch with use required after create"
**CTL AG**( create → **AX**(**AG**(¬create)) ) "Only one create"

**Annotations**: open, close *−− strict ordering of 2 phases*
**LTL G**(close → **O**(open)) "There must be open before close"
**CTL AG**(open → **AF**(close)) "After open, close is required on all branches"
**CTL AG**(open → **AX**(**A**[¬open **U** close])) "No multi−open without close"
**CTL AG**(close → **AX**(¬**E**[¬open **U** (close & ¬open)])) "No multi−close without open"

**Annotations**: init, process, release *−− strict ordering of 3 phases*
*−− init → process*
**LTL G**(process → **O**(init)) "There must be init before process"
**CTL AG**(init → **AF**(process)) "After init, process is required on all branches"
**CTL AG**(init → **AX**(**A**[¬init **U** process])) "No multi−init without process"
*−− process → release*
**LTL G**(release → **O**(process)) "There must be process before release"
**CTL AG**(process → **AF**(release)) "After process, release is required on all branches"
**CTL AG**(process → **AX**(**A**[¬process **U** release])) "No multi−process without release"
**CTL AG**(release → **AX**(¬**E**[¬init **U** (release & ¬init)])) "No multi−release without init"

**Annotations**: $phase_1$, $phase_2$, ..., $phase_N$ *−− strict ordering of N phases*
*−− $phase_i$ → $phase_{i+1}$ for i=1, 3, ..., N*
**LTL G**($phase_{i+1}$ → **O**($phase_i$))
**CTL AG**($phase_i$ → **AF**($phase_{i+1}$))
**CTL AG**($phase_i$ → **AX**(**A**[¬$phase_i$ **U** $phase_{i+1}$]))
*−− $phase_N$ → $phase_1$*
**CTL AG**($phase_N$ → **AX**(¬**E**[¬$phase_1$ **U** ($phase_N$ & ¬$phase_1$)]))

**Figure 3.3:** Examples of custom annotations (templates) defined in TADL.

However, UCM may also contain non-primary use-cases intended only for inclusion from other use-cases. As an example, see the use-case MOD2_UC12 (Figure A.9) which describes a shared behavior just included by other use-cases in the system – it cannot be executed separately.

Another important assumption in FOAM is that use-cases **not related** by precedence relation are considered **independent**. It is up to the analyst to progressively define the precedence relation where needed, e.g. in case of data-dependency encoded as a **#(create)** – **#(use)** annotation pair.

Each use-case is specified as a set of steps, variations and extensions. Additionally, UCM specifies precedence constraints among use-cases in the model. Based on precedence and inclusion, we define for each primary use-case $u$ a restricted version of UCM (here denoted as **r**UCM) containing only such use-cases that can influence $u$. This decomposition is possible due to our "independence" assumption mentioned in the previous paragraph. For each **r**UCM we build a non-deterministic automaton – called Use-Case Behavior Automaton (UCBA). All UCBAs together represent the overall behavior of the whole UCM.

UCBA is essentially an LTS with guards over boolean variables; thus it can be straightforwardly encoded in specification languages of modern model-checkers (we discuss such an encoding for the NuSMV model-checker in Section 3.3). The verification of UCBA is performed with respect to temporal logic formulae coming from the definition of temporal annotations.

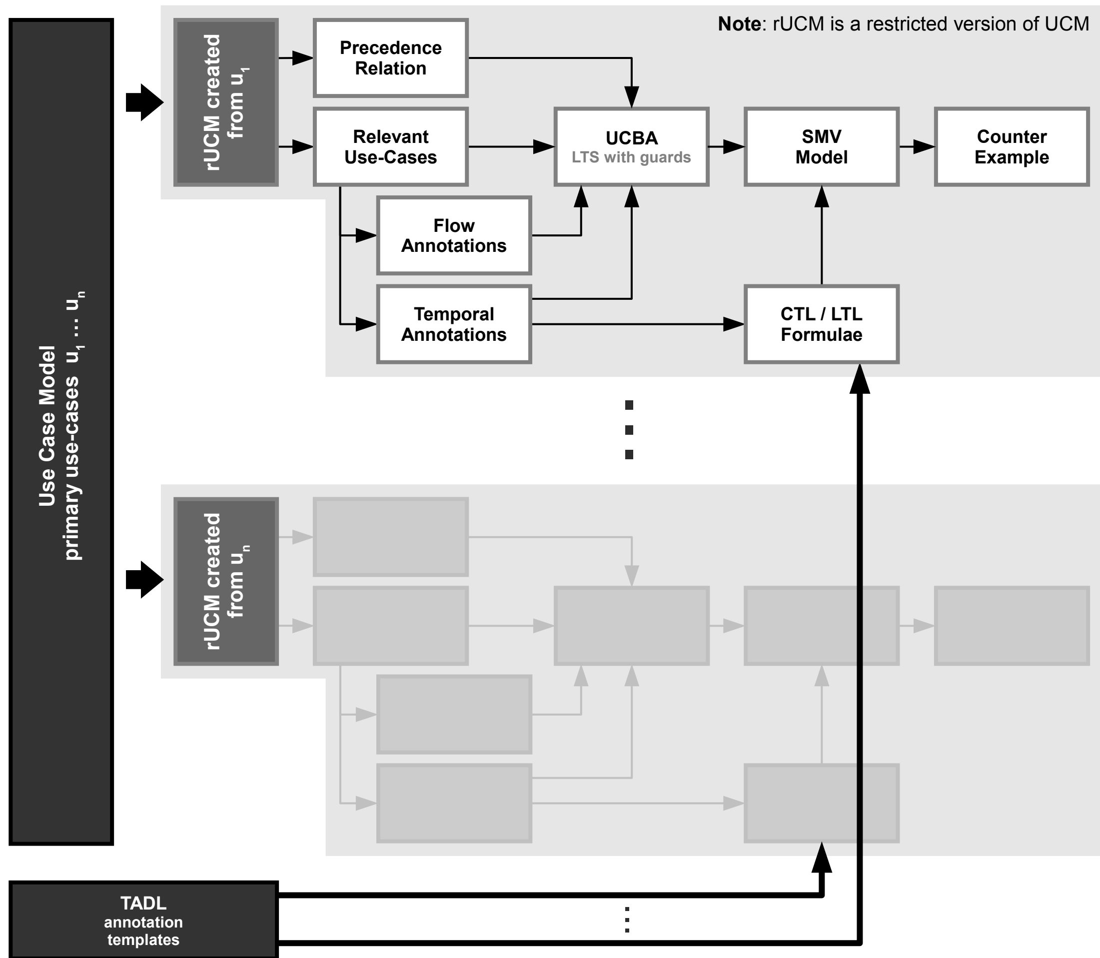



**Figure 3.4:** Verification method in detail

### 3.2.1 Formalizing the Input Use-Case Model

We start the formalization with definition of an annotated textual use-case. This structure represents a use-case as close as possible to the way it is usually written down (e.g. as in Figure 3.2). This means that we explicitly capture use-case steps (along with annotations attached to them), extensions and variations.

**Def.1 (Annotated textual use-case).** An annotated textual use-case is a tuple:

$$u = (S_u, W_u, w_u^m, Ext_u, Var_u, Flow_u, Temp_u)$$

where:

- $S_u$ is a set of all steps (sentences written in English);
- $W_u = \{w | w \subseteq S_u\}$ is a set of all scenarios of $u$ where each scenario is a linearly ordered set with its total order $\leq_w$ such that scenarios do not share steps, i.e. $\forall_{w,w' \in W_u}(w' \neq w) \Rightarrow (w \cap w' = \emptyset)$.
- $w_u^m \in W_u$ is the main scenario;
- $Ext_u : W_u \mapsto S_u$ is a mapping function which assigns extensions to steps, i.e. $w' \in W_u$ is an extension of $w \in W_u$ from step $s \in w$ if $Ext_u(w') = s$;
- $Var_u : W_u \mapsto S_u$ is a mapping function which assigns variations to steps, i.e. $w' \in W_u$ is a variation of $w \in W_u$ from step $s \in w$ if $Var_u(w') = s$;
- $Flow_u : S_u \mapsto 2^{\mathbb{F}}$ is a function that assigns a set of flow annotations to each step ($\mathbb{F}$ denotes a set of all flow annotations);
- $Temp_u : S_u \mapsto 2^{\mathbb{T}}$ is a function that assigns a set of temporal annotations to each step ($\mathbb{T}$ denotes a set of all temporal annotations).

Further, we say that a use-case is **well-formed** if the following structural constraints below are not violated. These rules follow the common practice of writing use-cases to help keep use-cases well-separated, comprehensible and of well understood semantics.

1. The annotations **#(abort)** and **#(goto)** can only be attached to the last step of a variation or extension.
2. The annotation **#(guard)** is attached only to the first step of an extension or variation.
3. The main scenario of a primary use-case does not contain any **#(goto)**, **#(abort)** or **#(guard)** annotations.

Now, we define UCM as a collection of use-cases accompanied with a precedence relation over use-cases. UCM thus represents the textually specified overall behavior of a system.

**Def.2 (Use-Case Model).** A Use-Case Model (UCM) is a tuple:

$$\mathbb{M} = (U_{\mathbb{M}}, U_{\mathbb{M}}^p, Prec_{\mathbb{M}})$$

where:

- $U_{\mathbb{M}}$ is a set of use-cases;
- $U_{\mathbb{M}}^p \subseteq U_{\mathbb{M}}$ is a set of primary use-cases;
- $Prec_{\mathbb{M}} : U_{\mathbb{M}} \times U_{\mathbb{M}}$ is a precedence relation on use-cases.

In the rest of the text, we assume only UCMs with well-formed use-cases.

**Def.3 (Restricted Use-Case Model for a use case to be verified).**

Let $Prec^*_{\mathbb{M}} : U_{\mathbb{M}} \times U_{\mathbb{M}}$ be a transitive closure of $Prec$.

Let $Inc_{\mathbb{M}} : U_{\mathbb{M}} \times U_{\mathbb{M}}$ be defined as:

$$(u_1, u_2) \in Inc \iff \exists s \in S_{u_1}, \textbf{\#(include: } u_2\textbf{)} \in Flow_{u_1}(s)$$

Let $Inc^*_{\mathbb{M}} : U_{\mathbb{M}} \times U_{\mathbb{M}}$ be a transitive closure of $Inc$.

The Restricted Use-Case Model for a use case to be verified (**r**UCM) $M$ for the particular use-case $u \in \mathbb{M}$ is defined as follows:

$$M = (U_M, U^p_M, Prec_M)$$

where:

- $U_M = U^p_M \cup \{Inc^*_{\mathbb{M}}(u_p), \exists u_p \in U^p_M\}$;
- $U^p_M = Prec^*_{\mathbb{M}}(u) \cup \{u\}$;
- $Prec_M = Prec_{\mathbb{M}}/U^p_M$.

Each **r**UCM $M$ is used for verification of a single use-case $u$. It is a restricted version of the UCM $\mathbb{M}$, which (i) contains only primary use-cases and use-cases that are (transitively) included in these primary use-cases, (ii) the set of primary use-cases $U^p_M$ contains the use-case $u$ and those use-cases which (transitively) precede the use-case $u$, all orderings of the use-cases from $U^p_M$ will be explored during the verification, (iii) **r**UCM has a restricted version of the precedence relation in UCM $\mathbb{M}$.

As explained above, use-cases not related by precedence or inclusion do not have data dependencies and can be verified independently. Moreover, we require that if a use-case $u_i$ is included in a use-case $u$, the use-case $u$ must satisfy all dependencies of the use-case $u_i$. That is, the set of use-cases that must precede the use-case $u_i$ must be a subset of the set of use-cases that must precede the use-case $u$. Thus precedences of included use-cases need not to be taken into the account explicitly.

The notion of **r**UCM is needed for better scalability of FOAM. Instead of building a single automaton from the whole UCM (as presented in [91] and [93]), we construct multiple restricted versions of UCM for each primary use-case. Without this restriction, the constructed automaton would take into account also ordering of use-cases which do not depend on each other. These orderings are redundant for the verification and would slow down the verification. As our tests show (see Section 3.5), the verification complexity grows exponentially with the number of use-cases. Unlike UCM, each **r**UCM is small for real-life specifications as it grows only with the number of preceding use-cases, which typically does not go over 4 (see the example 3.8) Thus the method scales linearly with the size of UCM. Moreover, the verification can run in parallel.

[94]
[93]

### 3.2.2 Formalizing the Use-Case Behavior Automaton

We can now focus on the verification of a single **r**UCM. In FOAM, we transform **r**UCM into UCBA, which has well-defined semantics and can be rather directly used as an input to standard model-checkers. UCBA is defined as follows:

**Def.4 (Use-Case Behavior Automaton).** An Use-Case Behavior Automaton (UCBA) is a tuple:

$$A = (V, init_0, \tau, B, AP, Val, Lab, Guards)$$

where:

- $V$ is a set of states.
- $init_0 \in V$ is the initial state.
- $\tau \subseteq V \times V$ are transitions.
- $B$ is a set of boolean variables.
- $AP$ is a set of atomic propositions.
- $Val : \tau \mapsto 2^{(B \times \{true, false\})}$ are actions (valuations) on transitions which assign values to boolean variables in $B$.
- $Lab : V \mapsto 2^{AP}$ is labeling of states by temporal properties.
- $Guards : \tau \mapsto 2^{\mathcal{L}}$ are guards on transitions (a guard $g \in \mathcal{L}$ is a propositional logic formula with variables from $B$).

The semantics of UCBA is the following:

- the execution starts in state $init_0$,
- the transition to another state is by non-deterministic choice among outgoing transitions, whose all guards are satisfied,
- upon the transition, the boolean variables of the automaton are updated based on the actions associated with the transition,
- for the sake of model-checking, the function $Lab$ gives the atomic propositions that hold in a particular state.

### 3.2.3 Building Use-Case Behavior Automaton – step #1

Having provided the definition of **r**UCM and UCBA, we now show UCBA construction from **r**UCM. This process is performed in two steps. In the first step below, we describe the automaton with the help of inference rules (in the form $\frac{premise}{conclusion}$). The rules put logical constraints on UCBA based on the input **r**UCM. In other words, the inference rules provide a logical theory, the model of which is UCBA. In FOAM, we take the minimal model (with respect to inclusion) as the resulting UCBA.

The basic UCBA structure constructed from use-cases $u_1, \ldots, u_n$ is depicted in Figure 3.5. There is an initial state $init_0$ with branches to particular sub-automatons,

each corresponding to one of the use-cases $u_1, \ldots, u_n$. The transitions to the sub-automatons are guarded by formulae that reflect the precedence constraints. This way, UCBA captures the non-determinism in sequencing the use-cases. After the sequence is completed, UCBA proceeds to the final state $succ_0$, where a cycle is formed to generate infinite traces as typically required by model-checkers.

In the inference rules, we use for brevity reasons the notation $s \to s'$ to denote the existence of states $s$ and $s'$ and the existence of transition between them, i.e. $s, s' \in V \wedge (s, s') \in \tau$. Additionally we use the notation $s \xrightarrow{[G]} s'$ to additionally state that the $G \in 2^{\mathcal{L}}$ is a subset of guards on transition $t = (s, s') \in \tau$, i.e. that $G \subseteq Guards(t)$; and we use the notation $s \xrightarrow{\{V\}} s'$ to additionally state that the $V \in 2^{(B \times \{true, false\})}$ is subset of actions on $t$, i.e. that $V \subseteq Val(t)$.

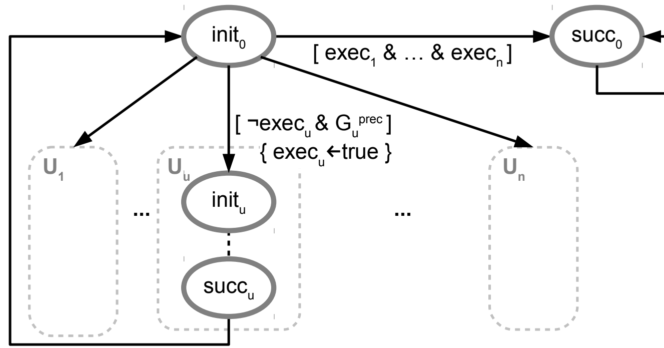



**Figure 3.5:** UCBA constructed from use-cases $u_1, \ldots, u_n$

[94]

The rules are as follows; each of them accompanied with an explanation.

**(Rule 1) Representing steps:** Every use-case step $x$ is represented in the automaton $A$ as a fixed number of states connected with transitions:

- $x^{\mathsf{in}}$ represents the state before $x$ has been executed.
- $x^{\mathsf{var}}$ is the source state of all variations attached to $x$.
- $x^{\mathsf{jump}}$ is the target state of a **#(goto:$x$)** annotation.
- $x^{\mathsf{ext}}$ is the source state of all extensions attached to $x$.
- $x^{\mathsf{out}}$ represents the state after $x$ and all its branching scenarios have been executed. Therefore it is the target state when continuing the execution from extensions and variations.

$$\frac{u \in U_M, x \in S_u}{x^{\mathsf{in}} \to x^{\mathsf{var}} \to x^{\mathsf{jump}} \to x^{\mathsf{ext}} \to x^{\mathsf{out}}}$$

**(Rule 2) Representing scenarios:** Let $w \in W_u$ be a scenario containing steps $x_1 \leq_w \ldots \leq_w x_n$ linearly ordered using the total order $\leq_w \in Ord_u$. Then in $A$ we connect the individual steps according to the order imposed by the $\leq_w$ relation.

$$\frac{u \in U_M, w \in W_u, x_1 \leq_w \ldots \leq_w x_n}{(x_1^{\mathsf{out}} \to x_2^{\mathsf{in}}), \ldots, (x_{n-1}^{\mathsf{out}} \to x_n^{\mathsf{in}})}$$

**(Rule 3) Connecting variations to parents:** Let $w \in W_u$ be a variation from step $x$ which contains steps $y_1, \ldots, y_n$. Then we connect $w$ (state $y_1^{\mathsf{in}}$) to its parent (state $x^{\mathsf{var}}$). If the variation is conditioned by a **#(guard)** annotation, we add this as a guard.

$$\frac{u \in U_M, w \in W_u, w = \{y_1, \ldots, y_n\}, Var_u(w) = x, \quad G_V = \{g \,|\, \textbf{\#(guard:}g\textbf{)} \in Flow_u(y_1)\}}{x^{\mathsf{var}} \xrightarrow{[G_V]} y_1^{\mathsf{in}}}$$

**(Rule 4) Connecting extensions to parents:** Let $w \in W_u$ be an extension from step $x$ which contains steps $y_1, \ldots, y_n$. Then we connect $w$ (state $y_1^{\mathsf{in}}$) to its parent (state $x^{\mathsf{ext}}$). If the extension is conditioned by a **#(guard)** annotation, we add this as a guard.

$$\frac{u \in U_M, w \in W_u, w = \{y_1, \ldots, y_n\}, Ext_u(w) = x, \quad G_E = \{g \,|\, \textbf{\#(guard:}g\textbf{)} \in Flow_u(y_1)\}}{x^{\mathsf{ext}} \xrightarrow{[G_E]} y_1^{\mathsf{in}}}$$

**(Rule 5) Continuation from scenarios:** Let $w$ be a branching scenario (variation/extension) from step $x$, which continues the execution in its parent scenario. Then we connect $y_n$ (state $y_n^{\mathsf{out}}$) – the last step of $w$ – to $x$ (state $x^{\mathsf{out}}$).

$$\frac{u \in U_M, w \in W_u, x = Var_u(w) \vee x = Ext_u(w), \quad w = \{y_1, \ldots, y_n\}, \textbf{\#(abort)} \notin Flow_u(y_n), \quad \forall_{s \in S_u} \textbf{\#(goto:}s\textbf{)} \notin Flow_u(y_n)}{y_n^{\mathsf{out}} \rightarrow x^{\mathsf{out}}}$$

**(Rule 6) Handling goto annotations:** Let $x$ be a use-case step annotated with an annotation **#(goto:$y$)**, where $y$ is another step in the same use-case. We handle the annotation by jumping to the $y^{\mathsf{jump}}$ location. This means that variations of the step $y$ will be skipped, however extensions are still executed.

$$\frac{u \in U_M, x \in S_u, \textbf{\#(goto:}y\textbf{)} \in Flow_u(x)}{x^{\mathsf{out}} \rightarrow y^{\mathsf{jump}}}$$

**(Rule 7) Handling abort annotations:** For each use-case step annotated with an annotation **#(abort)**, we add a transition which introduces an infinite loop. The loop is skipped only when returning from an included use-case (see Rule 10).

$$\frac{u \in U_M, x \in S_u, \textbf{\#(abort)} \in Flow_u(x)}{x^{\mathsf{out}} \rightarrow x^{\mathsf{out}}}$$

**(Rule 8) Handling mark annotations:** For each use-case step annotated with an annotation **#(mark:$b_i$)**, we add an action on the transition $x^{\mathsf{jmp}} \rightarrow x^{\mathsf{ext}}$ that assigns $true$ to the variable $b_i$ which is a boolean variable accessible globally within UCM. These boolean variables are used in guards on transitions within the automaton.

$$\frac{u \in U_M, x \in S_u, \textbf{\#(mark:}b_i\textbf{)} \in Flow_u(x)}{x^{\mathsf{jmp}} \xrightarrow{\{b_i \leftarrow true\}} x^{\mathsf{ext}}}$$

**(Rule 9) Resolution of includes (calling a procedure):** In FOAM, the *include* relationship among use-cases is expressed using **#(include)** annotations. In order to implement include operations within UCBA, we add a set of boolean variables $incl_{x,c}$, where $incl_{x,c} = true$ if a use-case $c$ has been called directly from a step $x$ of a use-case $u$.

Let $x$ be a use-case step annotated with **#(include:**$c$**)**. We disable execution of the transition $x^{\mathsf{jump}} \rightarrow x^{\mathsf{ext}}$ by adding a $[false]$ guard (because the use-case $c$ will be called instead). Then we connect $x^{\mathsf{jump}}$ to the initial state $y_1^{\mathsf{in}}$ of the use-case $c$ using a new transition, which sets the variable $incl_{x,c}$ to $true$.

$$\frac{u, c \in U_M, x \in S_u, \textbf{\#(include:}c\textbf{)} \in Flow_u(x), \quad w_c^m = \{y_1, \ldots, y_n\}}{x^{\mathsf{jump}} \xrightarrow{\{incl_{x,c} \leftarrow true\}} y_1^{\mathsf{in}}, x^{\mathsf{jump}} \xrightarrow{[false]} x^{\mathsf{ext}}}$$

**(Rule 10) Resolution of includes (return):** The last step of the included use-case $c$ is connected back to the calling use-case $u$, assuming that $c$ does not end by looping (i.e. does not contain the **#(goto)** annotation). Also all the abort-states have to be connected back to $u$. These newly added returning transitions revert the variable $incl_{x,c}$ back to $false$. The same use-case $c$ can be included into multiple use-cases (even multiple times into the same use-case). Therefore we use the guard $[incl_{x,c}]$ on each returning transition, so that the transition is enabled just within the scope of a particular inclusion. We also need to add $[\neg incl_{x,c}]$ guards for all the existing unguarded transitions that start in the same state as the newly added returning transitions.

$$\frac{\begin{array}{c} u, c \in U_M, x \in S_u, \textbf{\#(include:}c\textbf{)} \in Flow_u(x), \\ w \in W_c, w = \{y_1, \ldots, y_n\}, \\ (w = w_c^m \wedge \textbf{\#(goto)} \notin Flow_c(y_n)) \vee \textbf{\#(abort)} \in Flow_c(y_n), \\ G_1 = Guards(y_n^{\mathsf{out}} \rightarrow z_1), \ldots, G_k = Guards(y_n^{\mathsf{out}} \rightarrow z_k), \end{array}}{\begin{array}{c} \forall i \in \{1, \ldots, k\} : y_n^{\mathsf{out}} \xrightarrow{G_i \,\cup\, [\neg incl_{x,c}]} z_k, \\ y_n^{\mathsf{out}} \xrightarrow{[incl_{x,c}], \{incl_{x,c} \leftarrow false\}} x^{\mathsf{ext}} \end{array}}$$

**(Rule 11) Scheduling of use-cases:** The execution of use-cases may be arbitrarily sequenced with respect to the precedence relation $Prec_M$. In this rule, we create a mechanism that non-deterministically executes each use-case exactly once, while obeying the precedence relation. To do so, we introduce a boolean variable $exec_u$ for each $u \in U_M$, where $exec_u = true$ indicates that $u$ has already been executed. Furthermore, we introduce a global initial state $init_0$ with a transition to the initial state and from the final state of each primary use-case. Each transition to the initial state is guarded by a predicate over $exec_u$ variables, which reflects the precedence relation (a conjunction of $exec$ variables). The variable $exec_u$ is set to $true$ when entering the use-case, i.e. on the first transition $x_1^{\mathsf{in}} \rightarrow x_1^{\mathsf{var}}$ within the first step of the use-case $u$. (The variable is also set to $true$ when the use-case is called by inclusion from other use-case.)

$$\frac{u \in U_M^P, w_u^m = \{x_1, \ldots, x_n\}, \quad G_u^{\mathsf{prec}} = \{exec_v | \exists_{v \in U_M^P}(v, u) \in Prec_M\}}{init_0 \xrightarrow{[G_u^{\mathsf{prec}}, \neg exec_u]} x_1^{\mathsf{in}} \xrightarrow{\{exec_u \leftarrow true\}} x_1^{\mathsf{var}}, x_n^{\mathsf{out}} \rightarrow init_0}$$

**(Rule 12) Final state:** After the sequence of all primary use-cases has been executed successfully (indicated by $\forall_{u \in U_M^P} exec_u = true$), UCBA ends up in its final state $succ_0$ in an infinite cycle.

$$\frac{G = \{exec_u | u \in U_M^P\}}{init_0 \xrightarrow{[G]} succ_0 \rightarrow succ_0}$$

**(Rule 13) Atomic propositions:** Temporal annotations attached to use-case steps are translated to UCBA as atomic propositions attached to a corresponding $x^{\mathsf{jump}}$ state. Note that $x^{\mathsf{jump}}$ is a state that is always visited when a step in the use-case is taken (it is circumvented only when variation is used instead of the default step).

$$\frac{x \in S_u, u \in U_M}{Lab(x^{\mathsf{jump}}) = Temp_u(x)}$$

### 3.2.4 Building Use-Case Behavior Automaton – step #2

[94]
[93]

In this step, we address an issue in semantics of guards on variations and extensions. The typical interpretation, which we also stick to, is the following:

(i) A non-deterministic choice is assumed among the default step and its unguarded branches (i.e. variations and extensions without guards).

(ii) A non-deterministic choice is assumed among guarded branches that contain non-disjunctive guards.

(iii) Mutual exclusivity is assumed among the default step and unguarded branches on one hand and the guarded branches on the other hand.

The UCBA constructed in step #1 follows the semantics in terms of (i) and (ii), but not of (iii). To address (iii), we need to additionally introduce the guards for the default step and the unguarded variations and extensions so as the mutual exclusivity holds. This is done in the following way:

[94]

For each step $x$, we compute the union of guarding formulae on variations as:

$$G_V^x = \bigcup_{\forall t = (x^{\mathsf{var}}, y) \in \tau} \bigwedge Guards(t)$$

If $G_V^x$ is a non-empty set, we add to each unguarded transition from $x^{\mathsf{var}}$ a guard computed as:

$$\neg \bigwedge G_V^x$$

We apply a similar addition for unguarded extensions by computing:

$$G_E^x = \bigcup_{\forall t=(x^{\mathsf{ext}},y)\in\tau} \bigwedge Guards(t)$$

if $G_E^x \neq \emptyset$, we add to each unguarded transition from $x^{\mathsf{ext}}$ a guard:

$$\neg \bigwedge G_E^x$$

It should be noted that after applying this operation, there is no non-deterministic branching in the automaton which would mix guarded and unguarded transitions. Either there are no guards or all transitions have a properly defined guard.

### 3.2.5 Temporal properties

[94]
[93]

Now we show the instantiation of temporal logic formulae based on the **r**UCM and TADL (user-defined temporal annotations). Each temporal annotation used in **r**UCM has the form **#($a$:$s$)**, where "$a$" is the name of the annotation and "$s$" is the qualifier of the annotation in the use-case. Let $tadl$ be a TADL definition for the annotation name $a$. Such annotation therefore contributes a set of formulae $F_{\#(a:s)} = \bigcup_{i=1}^{n} F_i^{tadl}[_/\#(a{:}s)]$, where $F_i^{tadl}$ is the i-th logical formula defined in the template $tadl$ and where $[_/\#(_{:}s)]$ denotes renaming of each variable (represented by placeholder _) in the formula to the form "_:$s$".

In other words, whenever an annotation "$a$" appears in the text with the parameter "$s$", we need to instantiate all the corresponding formulae from the $tadl$ template by replacing the template variables with new variables containing $s$ as a qualifier.

The temporal properties to be verified by the model-checker are obtained as union over all the sets $F_{\#(a:s)}$ contributed by annotations used in **r**UCM.

[94]

**Example:** Assuming the following TADL template with annotations **#(a)** and **#(b)**.

```
Annotations: a, b
 LTL G(b → O(a))
 CTL AG( a → EF(b) )
```

Now, if the textual specification contains annotations **#(a:x)**, **#(a:y)** and **#(b:y)** the set of formulae will be constructed as a union of:

```
LTL G(b_x → O(a_x))
CTL AG( a_x → EF(b_x) )
LTL G(b_y → O(a_y))
CTL AG( a_y → EF(b_y) )
LTL G(b_y → O(a_y))
CTL AG( a_y → EF(b_y) )
```

. . . which yields the formulae:

```
LTL G(b_x → O(a_x))
CTL AG( a_x → EF(b_x) )
LTL G(b_y → O(a_y))
CTL AG( a_y → EF(b_y) )
```

[94]
[93]

## 3.3 Verification using NuSMV

We have implemented a verification of UCBA using the NuSMV model checker [15] (as UCBA is defined as an LTS structure, it is easy to employ any other state-of-the-art model checker for this task). NuSMV supports analysis of synchronous and asynchronous systems using Computational Tree Logic (CTL) and Linear Temporal Logic (LTL), thus we allow for both in defining temporal annotations.

Transformation of UCBA into the NuSMV input language is straightforward (Figure 3.7). There is a NuSMV variable $state$, which corresponds to the current state. Transitions of UCBA are reflected as NuSMV rules setting the $state$ variable based on the source state and guarding formulae.

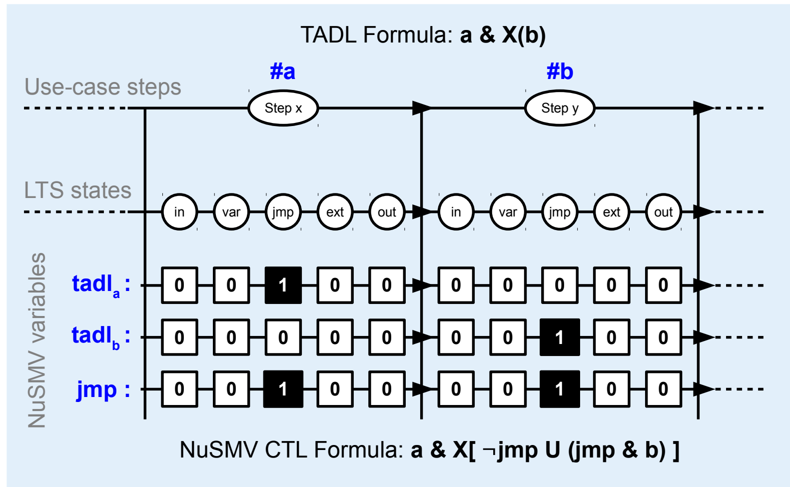



**Figure 3.6:** Transformation of TADL Formulas to NuSVM LTL/CTL specification. This demonstrates the increase in granularity when moving from a use-case to an LTS. Each step in a use-case corresponds to multiple LTS states. Model-checking variables that represent temporal annotations are only set to 1 in *jmp* states. Moreover, the temporal formulae have to also translated.

[94]

**Boolean variables:** The transformation from LTS to NuSMV generates variables that represent: (i) temporal annotations (set to $true$ at some point and immediately to $false$ in the next state), (ii) variables controlling inclusion of use-cases ($incl_{x,c}$) and scheduling ($exec_u$), (iii) **#(mark)** annotations (set to $true$ at some point, remaining $true$ until the final state).

**Non-deterministic guards:** NuSMV does not support a guarded non-deterministic choice between rules. However, an unguarded non-deterministic choice is supported (e.g. $state = s0 : \{s1, s2, s3\}$) Therefore we emulate non-deterministic guards in the following way:

1. Target state is non-deterministically chosen among all target states (regardless of the guards).
2. Transition to the selected target state is taken if the guard holds, if it does not, transition back to the source is taken and the process is repeated.

| TADL | | NuSMV LTL/PLTL/CTL |
|---|---|---|
| $X(\varphi)$ | $\equiv$ | $X[\neg jmp\, U\, (jmp \,\&\, \varphi)]$ |
| $G(\varphi)$ | $\equiv$ | $G(jmp \to \varphi)$ |
| $F(\varphi)$ | $\equiv$ | $F(\varphi)$ |
| $\alpha\, U\, \beta$ | $\equiv$ | $(jmp \to \alpha)\, U\, \beta$ |
| $\alpha\, R\, \beta$ | $\equiv$ | $\alpha\, V^2\, (jmp \to \beta)$ |
| $Y(\varphi)$ | $\equiv$ | $Y[\neg jmp\, S\, (jmp \,\&\, \varphi)]$ |
| $H(\varphi)$ | $\equiv$ | $H(jmp \to \varphi)$ |
| $O(\varphi)$ | $\equiv$ | $O(\varphi)$ |
| $(\alpha\, S\, \beta)$ | $\equiv$ | $(jmp \to \alpha)\, S\, \beta)$ |
| $(\alpha\, T\, \beta)$ | $\equiv$ | $\alpha\, T\, (jmp \to \beta)$ |
| $AX(\varphi)$ | $\equiv$ | $AX\ A[\neg jmp\, U\, (jmp \,\&\, \varphi)]$ |
| $AG(\varphi)$ | $\equiv$ | $AG(jmp \to \varphi)$ |
| $AF(\varphi)$ | $\equiv$ | $AF(\varphi)$ |
| $A[\alpha\, U\, \beta]$ | $\equiv$ | $A[(jmp \to \alpha)\, U\, \beta]$ |
| $EX(\varphi)$ | $\equiv$ | $EX\ E[\neg jmp\, U\, (jmp \,\&\, \varphi)]$ |
| $EG(\varphi)$ | $\equiv$ | $EG(jmp \to \varphi)$ |
| $EF(\varphi)$ | $\equiv$ | $EF(\varphi)$ |
| $E[\alpha\, U\, \beta]$ | $\equiv$ | $E[(jmp \to \alpha)\, U\, \beta]$ |

**Table 3.1:** Translated temporal operators from TADL to NuSMV. Notice that the only operators unaffected by the refinement are $F(\varphi)$, $O(\varphi)$, $AF(\varphi)$, $EF(\varphi)$.

Such operation requires splitting the original transition $x \to y$ into $x \to x_{guard} \to y$ and also adding the back-transition $x_{guard} \to x$. The new state $x_{guard}$ is used in NuSMV to deterministically check a negated version of the original guarding condition. In order to avoid infinite loops, fairness is enforced using a dedicated FAIRNESS condition featured by NuSMV.

**Temporal formulae:** A formula defined in TADL describes a temporal constraint on top of use-case steps. Obviously, the Rule 1 causes an increase in the granularity of states. An event that would normally have occurred in UCM as a continuous sequence, becomes discontinued in UCBA, occurring only in $jmp$ states that are surrounded by the other states where the event does not occur. That is, a single use-case step on which an event occurs is decomposed to multiple states containing one $jmp$ state and the event occurs only in that state. This can be demonstrated using the Figure 3.6. The variables $tadl_a$ and $tadl_b$, corresponding to annotations **#(a)** and **#(b)**, are set to $true$ only in the $jmp$ states (in $x^{\text{jump}}$ and $y^{\text{jump}}$ corresponding to steps $x$ and $y$).

To preserve the semantics of TADL formulae in NuSMV, we need to translate the temporal operators to a slightly longer form as defined in Table 3.1. In our example from Figure 3.6 the TADL formula $(a \,\&\, X(b))$ is transformed to NuSMV as CTL formula $(a \,\&\, X[\neg jmp\, U\, (jmp \,\&\, b)])$. The variable $jmp$ becomes $true$ in every state $x_1^{\text{jump}}, \ldots, x_n^{\text{jump}}$ corresponding to all use-case steps $x_1, \ldots, x_n$ and also in all looping states $s_1, \ldots, s_m$ such that $\forall s_i \in \{s_1, \ldots, s_m\} : \exists(s_i \to s_i)$.

```
MODULE main
    VAR state : {s_1, ..., s_n} -- all states of UCBA
    ASSIGN init(state) := init_0; -- initial state of UCBA
        next(state) := case
            state=x : {y_1, ..., y_n}; -- transitions x → y_1, ..., x → y_n
            state=y_i & !(g) : x; ... -- guarded transition x -g-> y_i
        esac;

    -- We need to highlight states relevant for checking temporal formulae.
    DEFINE jmp := s in { x_1^jmp, ... x_n^jmp, s_1, ..., s_m };

    FAIRNESS ! guardloop -- avoids infinite loops when testing guards
    DEFINE guardloop := state in {x_1, ..., x_m} -- states in guards

    -- variable v from UCBA (for simplicity, this includes variables for mark
    -- annotations, actions on transitions and temporal annotations
    VAR v : boolean;
    ASSIGN init(v) := FALSE;
        next(v) := case
            state = s^v : TRUE/FALSE; ... -- assigns value in state s^v
            TRUE : v; -- preserves the current value of v
        esac;

    -- LTL/CTL formula f ∈ F_A which uses variables t_1, ..., t_j
    LTLSPEC f(t_1, ..., t_j) ... CTLSPEC f(t_1, ..., t_j)
```

**Figure 3.7:** A simplified template for NuSMV code used in the transformation from UCBA.

## 3.4 Expressiveness of FOAM

To reflect the common guidelines in creating use-cases, FOAM features a number of restrictions on the control flow annotations – rules regarding the placement of **#(goto)**, **#(abort)** and **#(guard)** (e.g. guards allowed only at the beginning of variations and extensions). In this section we show that these restrictions do not actually impact the overall theoretical expressive power of the formalism. We show this by proving that a sufficiently general Kripke structure [16] and a related temporal logic formula, which form a typical input used in model-checking theory, can be transformed into a use-case and an annotation group while keeping the semantics. In particular, we show this for Kripke structures that have one initial state and one state in which all computation eventually ends in an infinite cycle. The first assumption does not cause any loss of generality, as we can always add a single initial state. The second assumption restricts us to describing functionality that eventually ends, which is one of the main characteristics of scenarios that are being described by use-cases.[3]

The claim showing the expressive power is formalized by the theorem below:

**Theorem 1.** *Let $K$ be a Kripke structure without unreachable states such that it has only one initial state $i$ and one state $f$, in which all computations eventually end in an infinite cycle. Let $F$ be an LTL or CTL formula. Then there exists a UCM $\mathbb{M}$ and a set*

[2]The "Release" temporal operator in NuSMV is denoted as "V"

[3]Allowing for valid use cases with infinite cyclic functionality would be also possible (along with transformation from the Kripke structure), but it would make no sense to speak about a set of use-cases and use-case precedences; also the UCBA would have to be constructed differently, thus we treat use-cases to be valid only when they have finite execution.

*of related annotation groups $G$ such that $F$ is satisfied in $K$ if and only if $\mathbb{M}$ is correct with respect to $G$.*

*Proof.* To prove the theorem, we construct UCM with one primary use-case $u$ and with no precedence constraints. Further we construct $G$ with just one annotation group $g$. We introduce a synthetic step $s_{start}$ as the first step of the main scenario of $u$. We define the remaining steps of $u$ as the edges in $K$ (note that we treat $K$ as an oriented graph). We add a particular path $p_m$ in $K$, which starts in $i$ and ends in $f$ (such path has to exist due to our assumptions), as the remaining steps of the main scenario.

Now we iterate the following steps until all vertices and edges in $K$ have been processed (we deem vertices and edges in $p_m$ and the edge forming the infinite cycle $f \rightarrow f$ as already processed): We select the path $p = v_1 \rightarrow \ldots \rightarrow v_n$ in $K$ such that (i) it starts in some of the processed vertices, (ii) when not considering the last vertex of $p$, the vertices of $p$ are disjunctive, (iii) when not considering the first and last vertex of $p$, the vertices have not yet been processed, (iv) the path cannot be made longer without violating (i)–(iii).

We define the path $p$ as a variation and the last edge of the path as a step annotated with goto. The variation is attached to the already processed edge (i.e. step) which originates in $v_1$ and which is not the first step in its scenario.

The annotation group $g$ is constructed as follows. Formula $F$ is used as the temporal logic formula in $g$. An annotation is introduced to the group for each distinct atomic proposition in the formula. A temporal annotation is attached to a step in the use-case $u$, on condition that a corresponding atomic proposition has been associated with a vertex in $K$ such that the vertex was the target of the step. □

| **Project Description** | **# of UCs** |
|---|---|
| Web and standalone application for managing members of organization | 17 |
| Web-based Customer Relationship Management (CRM) system | 37 |
| UK Collaboration for a Digital Repository (UKCDR) | 39 |
| Web-based e-government Content Management System (CMS) | 77 |
| Web-based Document Management System (DMS) | 41 |
| Web-based invoices repository for remote accounting | 10 |
| Protein Information Management System (PIMS) | 90 |
| Integration of two sub-system s in ERP scale system | 16 |
| Banking system | 21 |
| Single functional module for the web- based e-commerce solution | 9 |
| Web-based workflow system with Content Management System (CMS) | 75 |
| Polaris - Mission Data System (MDS) process demonstration | 16 |
| Vesmark Smartware$^{TM}$ - Financial decission system | 26 |
| Photo Mofo - Digital images management | 18 |
| iConf - Java based conference application | 16 |
| One Laptop Per Child - Web-based Content Management | 16 |

**Table 3.2:** Analysed projects before creating the referential use-case specification. Based on the work of Alchimowicz et al. [5].

## 3.5 Evaluation of scalability

In this section, we discuss scalability of FOAM with respect to industrial-size specifications. Obtaining industrial specifications is difficult due to intellectual property issues. Thus we used freely available reference specifications to get a better idea about the size and complexity of the specification usually encountered. In particular, we have relied on the benchmark conducted in [5] and [6]. The authors derived a referential specification based on the common characteristics of such specifications.

The benchmark covers 16 industrial specifications with together 524 use-cases (see Table 3.2). The purpose of the analysis was to create a referential specification reflecting typical patterns found in software projects. According to the authors, such specification can be used for presenting, testing, and verifying methods and tools for use-case analysis. The published referential specification is available on-line[4].

Relevant properties are listed in Table 3.3 from which we focused on the following 4 properties: the average number of steps in the main scenario is $4.8$, the average number of branches in a use-case is $1.6$, the average size of a branch is $2.5$ and the number of steps of validation nature is $4\%$.

| Analysed Property | Result |
|---|---|
| Number of analysed use-cases | 524 |
| The average number of steps in main scenario | 4.82, stdev=2.41 |
| Use cases with extensions | 72.1% |
| Number of extensions in use-case | 1.57, stdev=1.88 |
| Number of steps in extension | 2.46, stdev=1.61 |
| Steps with validation actions | 3.4% |
| Use cases with pre-conditions | 37.4% |
| Number of steps with reference to use-cases | 6.4% |

**Table 3.3:** Selected properties from the quantitative UCDB analysis. Based on the work of Alchimowicz et al. [5].

The main factors influencing the complexity of a use-case specification can be characterized by the following 5 parameters:

- $u$ the number of use-cases in **r**UCM,
- $m$ the number of steps within the main scenario
- $bc$ the number of branches
- $bl$ the length of a branch,
- $a$ the number of temporal annotations to be verified (e.g. create-use pairs) in the **r**UCM,

We conducted 4 experiments, each highlighting the dependency between the verification time $t$ and two of the parameters. Each dependency is rendered as a 3D surface plot accompanied with a 2D projection. In other words, our experiments show 3D projections of the 6-dimensional space:

$$(t \times u \times m \times bc \times bl \times a)$$

[4] http://ucdb.cs.put.poznan.pl/benchmark/2.f.n/srs/

As our experiments indicate, the time required for the verification depends exponentially on the size of the specification (with respect to every parameter). Fortunately, real specifications, such as [29] depicted in Figure 3.8, tend to be divisible into fairly small **r**UCM chunks. The complexity of the precedence relation within these chunks is usually very low. Typically, the use-cases are linearly ordered. Therefore, in our experiments, we decided to order our use-cases linearly with just a single cluster of 3 parallel use-cases within the **r**UCM (3! = 6 possible orderings in every experiment).

For the end user, it is crucial to obtain the verification results quickly, especially when FOAM is expected to be used iteratively in the development process. Therefore, we have chosen a one-minute deadline to which the verification should yield a result – either the specification is consistent or there is a violation with a counter-example. A NuSMV process running more than one minute was terminated and such measurement was not included into the result diagram.

For a given tuple of parameters $(u, m, bc, bl, a)$ in an experiment, we generate **r**UCM with the following properties:

1. All generated use-cases are identical, i.e. the same number of states, transitions and branches. The number of steps within a use-case is therefore: $bc \times bl + m$

2. All branches are attached to the step #2 in the main scenario.

3. Each branching scenario continues its execution in the main scenario instead of aborting.

4. Each temporal annotation added to the **r**UCM is either **#(create)** annotation in the step #1 of use-case #1 or **#(use)** annotation in the last step of a randomly picked use-case. This way, the model-checker has to verify both consistent and inconsistent specifications which should mitigate the bias towards one of these alternatives.

5. We presume a precedence relation which yields $min(3, u)! \leq 6$ possible orderings of use-cases within the **r**UCM.

We used the following hardware setup for our test: **CPU:** Core2 Duo CPU P9600 2.53GHz; **RAM:** 4GiB RAM; **OS:** 64bit Linux Mint 14 Nadia with kernel 3.5.0-17-generic; **NuSMV version:** NuSMV-zchaff-2.5.4-x86_64.

The following commands were used to run a single NuSMV instance:

```
read_model -i generated-nusmv-code.smv
flatten_hierarchy
encode_variables
build_model
print_usage
check_ctlspec
quit
```

The generated NuSMV code is similar to the following example:

```
MODULE main -- NuSMV code generated automatically from an annotated specification
    FAIRNESS !guardloop; -- prevent infinite loops in the following states
    DEFINE guardloop := s in {init_0, succ_0, init_1, init_2, init_3};

    VAR exec_1 : boolean;
    ASSIGN
        init(exec_1) := FALSE;
        next(exec_1) := case
            s=succ_1 : TRUE;
            TRUE : exec_1;
        esac;

    VAR exec_2 : boolean; ...

    VAR s : {init_0, init_1, init_2, init_3, u1_s0, u1_s1 ... u3_s2_b0_s3, succ_1, succ_2, succ_3, succ_0};
    ASSIGN
        init(s) := init_0;
        next(s) := case
            s=init_0 : {succ_0, init_1, init_2, init_3}; -- initial state
            s=succ_0 & !( exec_1 & exec_2 & exec_3 ) : init_0;
            s=succ_0 : succ_0; -- final state
            -- UC1
            s = init_1 & !(!exec_1) : init_0;
            s = init_1 : u1_s0;
            s = u1_s0 : u1_s1; ... s = u1_s18 : succ_1;
            -- UC2
            s = init_2 & !(!exec_2) : init_0; ...
            -- UC3
            s = init_2 & !(!exec_2) : init_0; ...
        esac;

    VAR v_create0 : boolean; -- temporal annotations create-use
    ASSIGN
        init(v_create0) := FALSE;
        next(v_create0) := case
            s = u1_s0 : TRUE;
            s = u1_s1 : FALSE;
            TRUE: v_create0;
        esac;

    VAR v_use0 : boolean;
    ASSIGN
        init(v_use0) := FALSE;
        next(v_use0) := case
            s = u2_s3 : TRUE; -- u2 picked randomly
            s = u2_s4 : FALSE;
            TRUE: v_use0;
        esac;

    CTLSPEC AG( v_create0 -> EF(v_use0) )  -- Branch with use required after create
    CTLSPEC AG( v_create0 -> AX(AG(!v_create0)) ) -- Only one create
    CTLSPEC A[!v_use0 U v_create0 | !EF(v_use0)] -- First create then use
```

### 3.5.1 FOAM scalability experiment 1

This experiment shows how the verification time depends on the number of use-cases in **r**UCM (parameter $u$) and the number of states within the main scenario (parameter $m$). We set the parameters as follows: $bc = 2$, $bl = 3$ (according to [6]) and $a = 5$. The surface plot depicted in Figure 3.9 shows an exponential growth in both axes. NuSMV is hitting the limits at **r**UCMs with more than 15 use-cases. The size of the main scenario within use-cases is, however, not an issue. FOAM can deal with scenarios of more than hundred steps. Only after 10 use-cases, increase of verification time is noticeable.

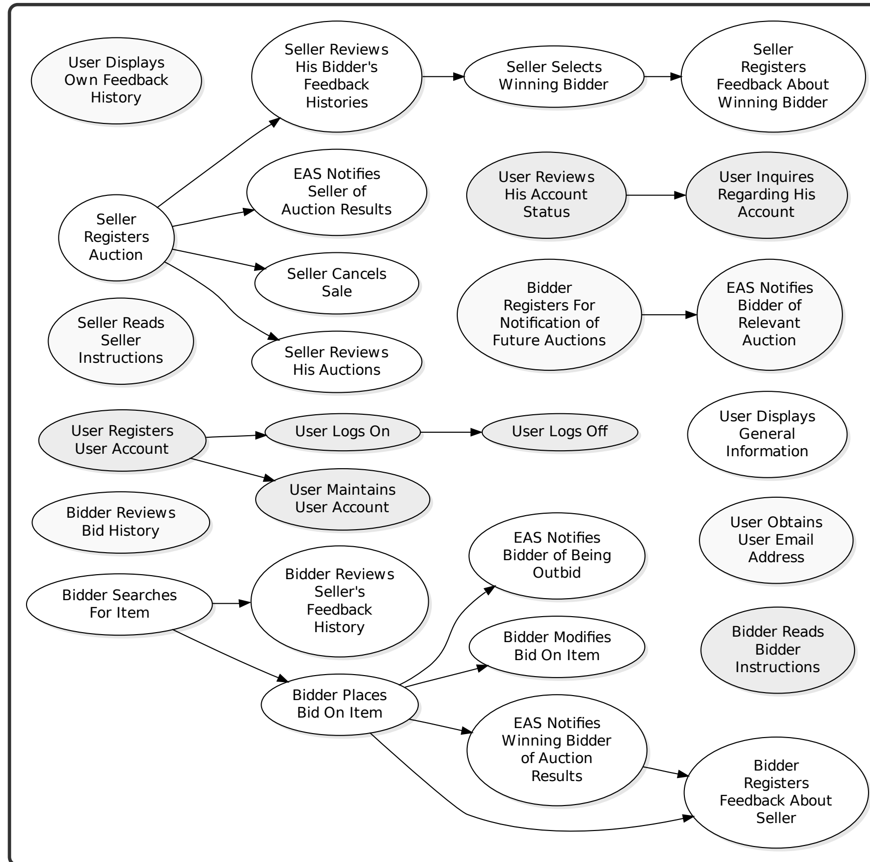



**Figure 3.8:** Precedence relation of 28 use-cases in our FOAM scalability experiment taken from the GPM Specification [29].

### 3.5.2 FOAM Scalability Experiment 2

In this experiment (Figure 3.10), we measured how FOAM deals with temporal annotations within the specification. We set the parameters as follows: $m = 5$, $bc = 2$ and $bl = 3$ (according to [6]). The surface plot again shows an exponential growth in both axes $u$ and $a$. It should be noted that the amount of temporal annotations is not a limiting factor because FOAM verifies even 50 create-use pairs in a reasonable time (under 10 seconds). We don't expect to find more temporal annotations in real-life specifications.

### 3.5.3 FOAM Scalability Experiment 3

The goal of this experiment was to estimate how FOAM deals with use-cases containing multiple branches. The surface-plot in Figure 3.11 shows that even large **r**UCMs (10+ use-cases) may contain more than 30 branches.

### 3.5.4 FOAM Scalability Experiment 4

For a firm number of use-cases ($u = 8$ in this experiment), we measured the verification time depending on the number of steps. Therefore, we combined parameters $m$ and

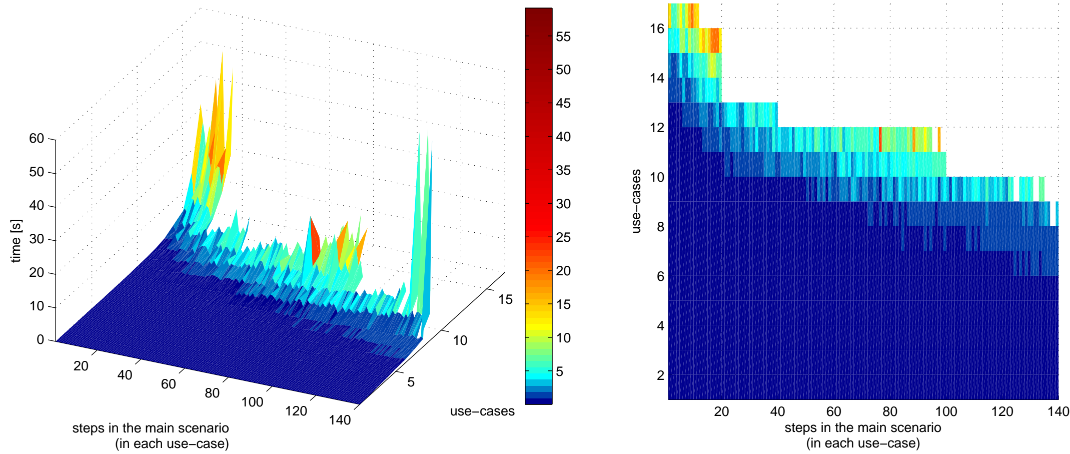



**Figure 3.9:** X-Y Axes: $u$, $m$; Fixed parameters: $bc = 2$, $bl = 3$, $a = 5$

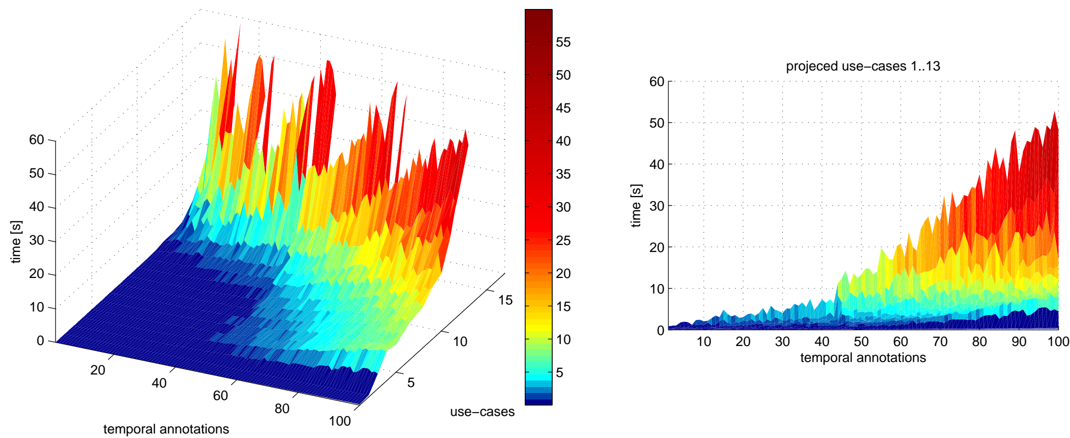



**Figure 3.10:** X-Y Axes: $u$, $a$; Fixed parameters: $m = 5$, $bc = 2$, $bl = 3$

$bc$ into a single surface plot (Figure 3.12). It is interesting how steep the curvature becomes when reaching approximately 160 steps per use-case.

### 3.5.5 Summary of the Experimental Results

To summarize all 4 experiments, an exponential growth in verification time on all axes in the diagrams can be seen. However, the results show that:

1. Up to 15 use-cases can be verified in a single **r**UCM if not more than 3 parallel use-cases are present in the precedence relation.
2. The number of temporal annotations is not a limiting factor (50+ create-use pairs).
3. FOAM can handle lot of branches (30+) and large scenarios (100+ steps).

## 3.6 Evaluation of learning curve

In the previous section, we focused on exploring the limits of the FOAM method such as the maximum number of use-cases to be handled, the amount of annotations that

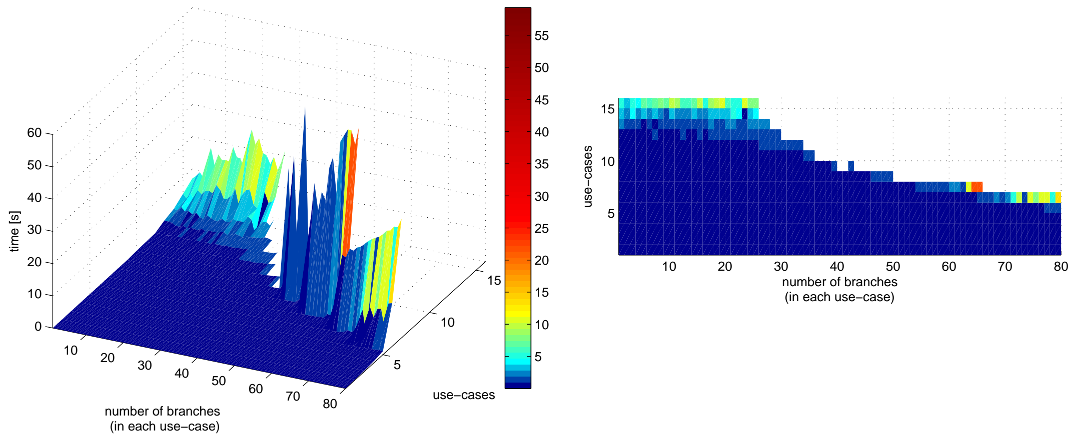



**Figure 3.11:** X-Y Axes: $u$, $bc$; Fixed parameters: $m = 5$, $bl = 3$, $a = 5$

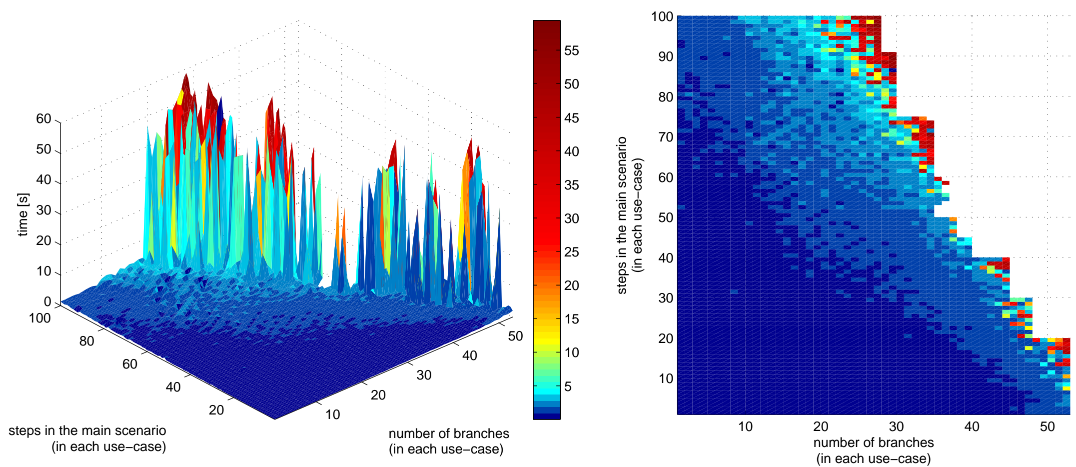



**Figure 3.12:** X-Y Axes: $m$, $bc$; Fixed parameters: $u = 8$, $bl = 3$, $a = 5$

can be verified etc. On the other hand, this chapter focuses on the ease of use in terms of learning curve.

We asked three unbiased testers to annotate a set of use-cases. We measured the time required to learn the basic concepts of FOAM and the time required for annotating a typical use-case. Before performing the experiment, one of the FOAM authors selected a set of suitable use-cases and created an annotated *referential specification* which was then compared with the testers' answers.

### 3.6.1 Selection of use-cases for the test

Just like in Section 3.5 on page 34, we chose the specification [6] (Admission System version 2.0F quantitative) because it represents 16 industrial specifications distilled into 34 use-cases. The goal was to demonstrate all important aspects of the FOAM method using a minimal set of use-cases. Therefore, our selection criteria were as follows:

- The set of use-cases should be closed on precedence and inclusion (transitively).
- At least one use-case should contain the include relation.

- There should be extensions or variations in most of the use-cases.
- Some branches should contain aborts and jumps (goto).
- Some scenarios should demonstrate the use of the open-close and create-use annotations.

For the sake of simplicity, we selected the following nine use-cases:

```
UC1: Login to the system
MOD1_UC1: Register in the system
MOD1_UC2: Provide personal and education information
MOD1_UC3: Choose a major
MOD1_UC4: Assign an application fee to a major
MOD1_UC5: Check application status
MOD2_UC1: Create a new admission
MOD2_UC8: Add a new user
MOD2_UC12: Import admissions fees
```

Each of them can be found in Appendix A.1. Comparing to the original specification [6], we slightly updated the use-cases based on the verification results from our FOAM tool. Further details about the inconsistencies identified are discussed in Section 3.7.

### 3.6.2 Method applied by independent testers

Three testers (*TesterA*, *TesterB*, *TesterC*) without any prior knowledge of the method were asked to apply FOAM on the set of use-cases selected in Section 3.6.1 but without any annotations and precedence relation specified. Additionally, they received another already annotated use-case[5] as an example and an explanation of the flow and temporal annotations (Appendix A).

Then, each of them was exposed to an oral explanation session about the method. The explanation was held without the presence of the other testers so that they cannot influence each other by questions, etc. Finally, the testers applied the method and we measured the time spent on each of the following stages: (1) reading the use-cases, (2) ordering the use-cases according to the precedence relation, (3) adding flow annotations, (4) adding temporal annotations, and (5) reporting on identified inconsistencies.

### 3.6.3 Feedback from testers

Table 3.4 summarizes the measured time spent by each tester on different stages of the use-case analysis.

#### Adding flow annotations

As can be seen from the table, there are minor differences between the testers. When we compared the annotations from all the testers, we noticed significant similarities in their choice and placement of annotations. As expected, the precedence relation and flow annotations were the same for all of them and also the same as in the reference specification.

[5] The use-case MOD2_UC6 is not part of the evaluated specification. It just served as an example for the testers.

There were slight differences in the use of the **#(include)** annotation (it is actually ambiguous whether the extension "user decided to contact admin" in UC1 is an inclusion of MOD2_UC2 or an abort).

### Adding temporal annotations

More interesting are the results of temporal annotations. The **#(create)** - **#(use)** pair was the most challenging one while almost all **#(open)** - **#(close)** annotations matched the referential solution. For example, comparing to the referential solution, none of the testers identified the **#(create:chosenMajor)** annotation. It seems that the notion of "transactions" was more natural to them than a slightly vague notion of "data dependency". The testers obviously avoided capturing data dependencies across multiple use-cases. Since a transaction does not usually cross boundaries of a single use-case, it was easier for them to correctly identify the **#(open)** and **#(close)** annotations.

We think that the accuracy of the testers' answers could have been improved had we provided more examples demonstrating the use of temporal annotations on more than one use-case. In our case study we showed them only MOD2_UC6 (Appendix A) without demonstrating data dependencies across multiple use-cases.

### Summary

In general, all the testers gave a positive feedback. They claimed that the process of annotating use-cases forced them to think about the dependencies from a different perspective comparing to just simple reading. After annotating the text, they could easily spot important places in the text with respect to the temporal dependencies among use-case steps.

According to the testers' responses, the time invested in learning FOAM can be considered negligible. Table 3.4 shows that annotating a single use-case takes approximately 6 minutes. This number is based on our sample-set of 36 use-cases (9 use-cases analyzed by 4 people). Unfortunately, we cannot compute the standard deviation, because we did not measure the required time for each use-case individually.

| | Author | TesterA | TesterB | TesterC | AVG(A,B,C) |
|---|---|---|---|---|---|
| **Reading** | n/a | 10 min | 30 min | 7 min | 16 min |
| **Precedence** | 12 min | 6 min | 5 min | 13 min | 8 min |
| **Flow annot.** | 7+3 min | 11 min | 8 min | 17 min | 12 min |
| **Temp. annot.** | 25 min | 23 min | 24 min | 23 min | 23 min |
| **Total Analysis** | 47 min | 50 min | 67 min | 60 min | **59 min** |
| **Analysis per UC** | 5 min | 5 min | 7 min | 6 min | 6 min |
| **Learning** | n/a | 13 min | 15 min | 18 min | 15 min |
| **Total** | 47 min | 63 min | 82 min | 78 min | 74 min |

**Table 3.4:** Results of the FOAM case study.

## 3.7 Evaluation of the FOAM tool

After collecting the responses from testers, we compiled a new version of the specification. Using our verification tool, we were able to detect inconsistencies that slipped through our manual review. (The referential specification in the Appendix A.1 already contains the updated version together with markings showing the fixes applied after the verification.)

**Inconsistencies detected by visual inspection:**

1. Extensions $1a$ and $1b$ from MOD1_UC2 had to be attached to a different step compared to the original. Also the sequencing of actions in MOD1_UC2 had to be redefined properly by adding variations $4a2a$ and $4b2a$. (Figure A.3).

2. MOD1_UC4 (Figure A.5) – Instead of variation $5a$, the original specification defined an extension $5a$. It is obvious that the "money transfer" scenario is a variation of the "credit card payment" scenario because either the former or the latter is executed, never both.

**Inconsistencies detected by the tool:**

1. MOD1_UC1 (Figure A.2) – The tool detected the missing **#(close:registration)** which is necessary due to the infinite loop introduced by the **#(goto:4)** annotations in **Extensions 5a, 5b, 5c**. This type of errors are not very serious. However, this way, FOAM framework guides us to explicitly state the boundaries of an "open-close" transaction which increases the clarity of the specification.

2. MOD1_UC3 (Figure A.4) – We identified that this use-case is not a primary use-case because the token "chosenMajor" was created in step 3 and never used. This finding however relies on the particular annotations attached to the use-case. We may also think of a different scenario, when this use-case might be primary. However, due to the current data dependencies, MOD1_UC3 is not a primary use-case. This nicely shows a type of inconsistency which can be easily spotted by verification but may get undetected when inspecting visually.

3. MOD1_UC4 (Figure A.5) – A missing **Extension 5a3a** was identified. The problem was that the money transfer may fail due to the included MOD2_UC12 (**Extension 2a**). If it fails, the **#(create:registeredMoney)** will not be visited and therefore the subsequent **#(use:registeredMoney)** will cause a verification error.

   It can be assumed that such a pattern is common in large specifications if the structure of use-cases changes over time (e.g., the **Extension 2a** in MOD2_UC12 might have been added later in the project; by a different analyst). When the dependencies are captured formally, such issues can be spotted easily.

4. MOD2_UC1 (Figure A.7) – This use-case also contained an interesting type of error. The main goal of this use-case is to perform a transaction concerning the "admissionForm". However, there were two **Extensions 3a, 4a** introducing

an infinite loop. When writing a use-case specification, we normally assume that the execution eventually reaches either success state or some abort state. Sometimes, however, we would like to limit the number of executions of a given loop. (e.g. a password may be entered maximum N times.) In FOAM, this can be achieved with a guard that can disable the execution of an already visited branch.

In MOD2_UC1, the first problem spotted by the verification tool was the missing annotation **#(close:admissionForm)** in case the execution loops forever. However tempting it might seem, we cannot just add this annotation to steps $3a1$ and $4a1$. In that case, **#(close:admissionForm)** would be visited repeatedly, which is forbidden by the semantics of "open-close". The best option here was to break the infinite loop by introducing a guarded variation for each of the looping extensions.

5. MOD2_UC12 (Figure A.9) – Here, we also detected that this use-case is not primary, similarly to the use-case MOD1_UC3 (due to data dependency). This case is, however, more obvious because MOD2_UC12 does not act as a precedence of any other use-case, whereas MOD1_UC3 precedes MOD1_UC4 and MOD1_UC5.

6. MOD2_UC8 (Figure A.8) – Again, the tool identified that an "open-close" transaction needs to be properly closed in a looping extension $3a$.

## 3.8 Implemented FOAM tool

The FOAM tool is implemented in Java as a collection of Eclipse plugins. The implementation is based on the EMF framework through which we designed all our meta-models. All transformations between models are implemented in Xtend[6] language. Xtend is a statically-typed programming language which translates to comprehensible Java source code and due to its powerful features, it is suitable for model-to-model and model-to-text transformations.

A high-level overview of the tool's pipeline is depicted in Figure 3.13. Input of the transformation are (i) textual use-cases containing annotations represented as one file per use-case, (ii) definition of temporal annotations defined in TADL syntax, one file per annotation group. All use-cases are aggregated in a single UCM model, which is stored as a file in XMI format. At this point, all the annotations within UCM are treated as "unknown" annotations, without any specific semantics assigned.

The resolution of annotations starts by resolving all flow annotations. Then, using TADL definitions, temporal annotations are resolved. Optionally, we support custom resolvers to be implemented in future. This allows users to assign semantics to custom annotations which can be leveraged by other external tools. Most importantly, our framework ensures that the traceability links will be preserved throughout transformation phases. For example, we can trace back the custom annotations from the generated UCBA automaton or even from the generated counter-example.

[6] http://www.eclipse.org/xtend/documentation.html

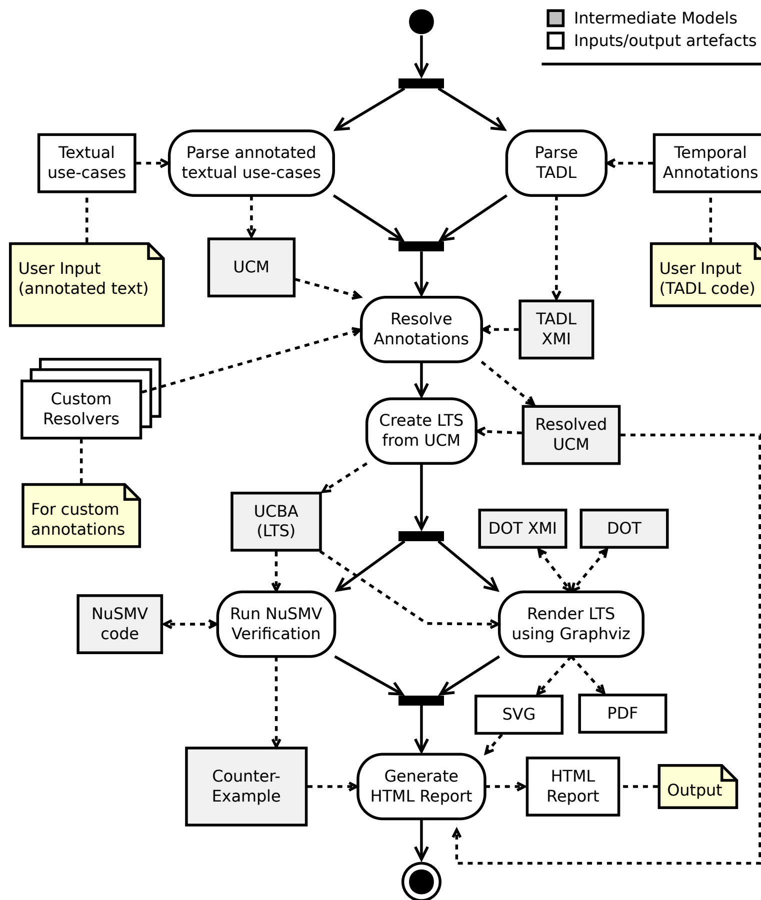



**Figure 3.13:** Transformation pipeline of the FOAM tool.

Next, the "Resolved UCM" is converted to UCBA. We also preserve traceability links between the states of the newly created automaton and the original use-cases in UCM. The automaton is converted into NuSMV code and passed to the NuSMV model checker. When a counter-example is produced by NuSMV, it is converted to an HTML report depicted in Figure 3.16 that shows the sequence of use-case steps causing the validation errors. A visual representation of the UCBA automaton is rendered using Graphviz. For each use-case, we obtain a single HTML web page (Figure 3.15 that shows the steps in textual form together with a flow graph. The generated report also contains overview of all the use-cases – the precedence relations and inclusions depicted in Figure 3.14. Each group of related temporal annotations is explained on a separate web page of the generated report.

## 3.9 Summary of Chapter 3

In this chapter we have presented a method for verifying correct sequencing of actions in use-cases. The main advantage of the method is its ease of use, in particular it works with use-cases in their natural language form and requires only a few basic annotations to be inserted in the use-cases. Thus, it can be easily integrated with existing development processes. By allowing for user-defined annotations, it can be further customized. For instance to verify domain-specific properties or to inject custom meta-data.

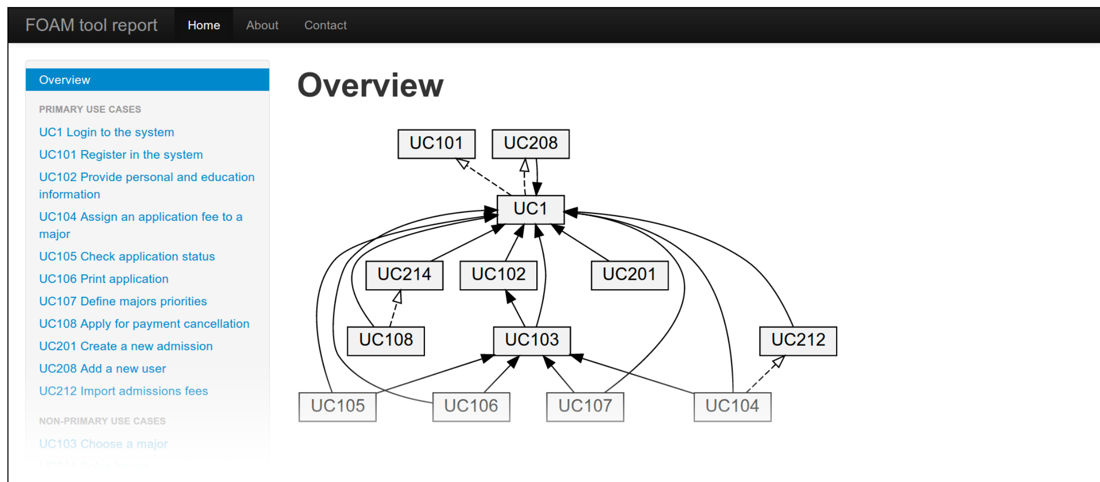



**Figure 3.14:** This is a screenshot from the HTML report generated by FOAM tool. The page visualizes the precedence relation (→) and include relation (- - →) among all use-cases in the specification. On-line example is available at http://foam-tool.appspot.com.

The idea of encapsulating complex temporal properties behind a simpler interface was successfully applied by many researchers. For example, authors of [19] and [26] proposed *property specification patterns* to capture the essential structure of some aspect of a system's behavior.

**Implementation:** We have developed a command-line-based tool that performs verification of use-case models annotated with FOAM annotations. The architecture of our tool is modular and extensible. Each transformation phase is clearly separated with well-defined meta-models describing its inputs and outputs. At the time of writing this text, the tool can be downloaded from http://vlx.matfyz.cz/Projects/FoamTool.

FOAM tool report Home About Contact

Overview

PRIMARY USE CASES

UC1 Login to the system
UC101 Register in the system
UC102 Provide personal and education information
UC104 Assign an application fee to a major
UC105 Check application status
UC106 Print application
UC107 Define majors priorities
UC108 Apply for payment cancellation
UC201 Create a new admission
UC208 Add a new user
UC212 Import admissions fees

NON-PRIMARY USE CASES

UC103 Choose a major
UC214 Solve issues

TADL DEFINITIONS

create-use
open-close

ERRORS

create-use registeredMoney

UC108: Apply for payment cancellation

preceeded: UC1 Login to the system

1. Candidate chooses one of the selected and payed majors.
2. Candidate chooses the apply for cancellation option.
3. System presents the application form.
4. Candidate provides justification for the payment cancellation.
5. System stores the application.
6. Selecting committee analyses application as an issue. #create:issue, #include:UC214
7. Selecting committee confirms payment cancellation. #assert:issueMarkedPositively
8. System sends information to the Candidate that the cancellation has been accepted and how money can be refunded.

**Variation** : 7a. The issue was marked negatively.
7a1. System sends information to the Candidate that the cancellation has been refused. #guard:mark_issueMarkedNegatively
7a2. Use case aborted. #abort

1 2 3 4 5 6 7 8
UC214 Solve issues
Variation: 7a.
The issue was marked negatively.
7a1 7a2

**Figure 3.15:** This is a screenshot from the HTML report generated by FOAM tool. The page shows a single use-case with all its steps (the main scenario and a variation). There are several annotations visible – flow, temporal and a custom "assert" annotation. The sequencing of actions in the use-case is visualized in a flow diagram. On-line example is available at http://foam-tool.appspot.com.

FOAM tool report Home About Contact

Overview

PRIMARY USE CASES

UC1 Login to the system
UC101 Register in the system
UC102 Provide personal and education information
UC104 Assign an application fee to a major
UC105 Check application status
UC106 Print application
UC107 Define majors priorities
UC108 Apply for payment cancellation
UC201 Create a new admission
UC208 Add a new user
UC212 Import admissions fees

NON-PRIMARY USE CASES

UC103 Choose a major
UC214 Solve issues

TADL DEFINITIONS

create-use
open-close

ERRORS

create-use registeredMoney

Error: create-use registeredMoney

at least one branch with use required after create:

```
CTL AG (create -> EF use)
```

Trace:

| | | |
|---|---|---|
| UC1 | 1 | User opens main page. |
| UC1 | 2 | System presents main page with a login form. |
| UC1 | 3 | User fills the login form with the authentication data. |
| UC1 | 4 | System verifies the given data. |
| UC1 | 5 | System welcomes Candidate. |
| UC212 | 1 | Administrator chooses an option to import payments from the bank system. |
| UC212 | 2 | System imports payment entries from the bank. |
| UC212 | 3 | System displays a list containing information about all imported admission fees. #create:registeredMoney |

**Figure 3.16:** This is a screenshot from the HTML report generated by FOAM tool. The page shows the generated counter-example. We can see the temporal formula causing the error – *"at least one branch with use required after create"*. The trace shows a sequence of use-case steps that lead to the violation of the temporal constraint. On-line example is available at http://foam-tool.appspot.com.

# Chapter 4

# Domain model elicitation

The second part of this thesis is focused on statistical classification and prediction of software engineering artefacts. In particular, we introduce an automated approach to elicitation of a domain model directly from natural language. We have summarized this idea in the papers [12, 90, 92, 96, 95]. The method described in this chapter acts as a proof-of-concept of a broader idea. Here, we show, how we have combined artefacts such as textual documents, domain models in a model which supports statistical inference. We formulate the problem as a supervised machine-learning classification task.

**Our approach combines:**

- linguistic features gathered from text by existing Natural Language Processing (NLP) tools and
- features related to software engineering, such as relations between domain model entities.

**The contribution consists of 2 important parts:**

1. Method for elicitation of the domain model from text.
2. A tool that automates all phases – elicitation, training, evaluation. Most importantly, the tool automates the evaluation of classifiers' performance which is necessary for designing new features and also for adapting existing features to a new specification domain.

We show evaluation results from our experiment on a Library System specification.

## 4.1 Domain modeling

[96]
[95]

At the very beginning of software development, there should be a document containing a detailed textual description of requirements. The software analyst writes it in cooperation with other stakeholders of the project, i.e. with the customer, the end users, and the domain experts.

To solve the semantic gap between stakeholders, software engineers adopted domain modeling as a mandatory part of their work. The domain model serves as a common vocabulary in the communication among technical and non-technical stakeholders throughout all project phases. This helps them come to an agreement on the meaning of important concepts. In [85] (p. 23), the domain model is defined as "a live, collaborative artefact which is refined and updated throughout the project, so that it always reflects the current understanding of the problem space".

From the modeling point of view, the domain model consists of classes, associations (aggregation, inheritance) and other elements commonly found in UML class-diagrams. The main difference is that a domain model focuses on entities in the problem-space and should always be independent of any particular implementation technology. On the other hand, UML class-diagrams are tight to the solution space and may therefore entail elements such as caching- or UI-related classes.

### 4.1.1 Iterative development and refinement

A domain model is usually not constructed en bloc, yet it undergoes refinement starting from the first prototype elicited from text. By reading the specification documents, the analyst tries to identify important concepts that may be included to the domain model. The analyst has to read the text multiple times while each time focusing on a certain aspect of the domain model, e.g. naming of entities, aggregation or dependencies.

Naturally, due to the vagueness of a textual specification, it is not uncommon that two analysts may derive significantly different models from the same input texts. However, this is not an issue since the prototype domain model is further refined as the project development progresses. Even the original specifications may be changed when inconsistencies are identified during the elicitation process.

It would be desirable that an initial prototype can be derived automatically to some extent. Apart from the creative nature of the refinement phase, which involves humans, the initial prototype can be derived automatically to some extent. The motivation for the automation is to save the initial effort of the analyst ranging from a couple of hours to a couple of days. Instead of starting from a plain text, the analyst can already start from a model which points to specific locations in the source text.

### 4.1.2 Grammatical Inspection

Grammatical inspection [1] is an approach commonly used by practitioners and extraction tools to get a quick start. The text is scanned for nouns, adjectives, verbs or other linguistic units. For example, the book [85] suggests that 80% of domain classes can be discovered in the initial domain modeling session. [1] The analyst should:

1. Create a list of candidates for domain entities by scanning the text for nouns and noun phrases as a representatives of classes/objects. (The simplest approach would be to consider a simple equivalence *noun=class*)

[1] It is advised that the analyst does not spend too much time in this phase and that the rest of the objects are identified during the robustness analysis, i.e. when analyzing the textual use-cases.

2. Identify named entities such as places, addresses, names of organizations, years, etc. This is important since these entities should usually be excluded from the domain model.

3. Remove obviously duplicate terms, such as the "User Account" might be a duplicate of "Customer Acount", or "Book Review" might be a duplicate of "Review Comment".

4. Remove generic terms (e.g. "Internet") and UI-related terms. (e.g. "Password")

Due to the complexity of the natural language, such approach requires human intervention.

## 4.2 Natural language processing techniques

In this section, we briefly summarize several concepts from computational linguistics that are related to our statistical approach explained below. Natural language processing plays an important role in tasks such as requirements elicitation, extracting formal models from text (static or dynamic), enforcing constraints imposed on the specification language etc.

### 4.2.1 Linguistic pipeline and common analysis structure

Existing NLP frameworks, such as Stanford CoreNLP [2], Apache OpenNLP [3] or Apache UIMA [4], are conveniently based around a central data structure (in CoreNLP it is called "Annotation", in UIMA it is the Common Analysis Structure or CAS). Usually, a linguistic pipeline consists of multiple components focused on a specific linguistic task (annotators) that enrich the common data structure. Most of the tasks are conditionally dependent, forming a lattice. Some pipelines, such as the CoreNLP is able to automatically resolve dependencies using topological ordering. For example, the POS-tagger component requires that the text is already converted into tokens by the Tokenizer component.

### 4.2.2 Tokenization

One of the first steps in linguistic analysis is tokenization. The source text is transformed into a stream of tokens that represent individual words, numbers, punctuation marks or other special tokens. For example, the sentence *"A librarian is able to create, edit and delete a medium (manage medium)."* will be transformed into the following tokens:

```
[A] [librarian] [is] [able] [to] [create] [,] [edit]
[and] [delete] [a] [medium] [(] [manage] [medium] [)] [.]
```

[2] http://nlp.stanford.edu/software/corenlp.shtml
[3] http://opennlp.apache.org
[4] http://uima.apache.org

### 4.2.3 Part-of-speech tagging

Another important linguistic task is part-of-speech tagging which assigns a single POS-tag for each token. Standard POS-tags are defined by the Penn Treebank Project [70], as depicted in Table 4.1. In our implementation, we use the POS-tagger shipped as a part of the Stanford CoreNLP package. As a POS-tagging model, we use a relatively fast model *english-left3words-distsim.tagger* with the reported accuracy of 96.97%.

Even though the state-of-the-art taggers claim over 97% accuracy in POS-tagging, the trick is that the accuracy is computed for individual tokens, including punctuation marks. If we consider the POS-tagging accuracy of a whole sentence, as reported by Manning in [67], we immediately drop at modest 55-57%, mostly due to differences between training and real texts in terms of topics and writing style.

| Tag | POS-tag description |
|---|---|
| CC | Coordinating conjunction |
| CD | Cardinal number |
| DT | Determiner |
| EX | Existential there |
| FW | Foreign word |
| IN | Preposition or subordinate conjunction |
| JJ | Adjective |
| JJR | Adjective, comparative |
| JJS | Adjective, superlative |
| LS | List item marker |
| MD | Modal |
| NN | Noun, singular or mass |
| NNS | Noun, plural |
| NNP | Proper noun, singular |
| NNPS | Proper noun, plural |
| PDT | Predeterminer |
| POS | Possessive ending |
| PRP | Personal pronoun |

| Tag | POS-tag description |
|---|---|
| PRPS | Possessive pronoun |
| RB | Adverb |
| RBR | Adverb, comparative |
| RBS | Adverb, superlative |
| RP | Particle |
| SYM | Symbol |
| TO | to |
| UH | Interjection |
| VB | Verb, base form |
| VBD | Verb, past tense |
| VBG | Verb, gerund or present participle |
| VBN | Verb, past participle |
| VBP | Verb, nonrd person singular present |
| VBZ | Verb, rd person singular present |
| WDT | Whdeterminer |
| WP | Whpronoun |
| WPS | Possessive whpronoun |
| WRB | Whadverb |

**Table 4.1:** POS-tags as defined by the Penn Treebank Project [70]. Also used by the Stanford POS-tagger.

### 4.2.4 Lemmatization

The goal of lemmatization is to transform inflectional and derived form of a word into its basic form. For example, all the forms "presents", "presenting", "presented" would be lemmatized into a single word "present". This operation is useful if we want to search in a collection of words. Also, we use it in our approach when deriving names of entities based on their original form in the text. The lemmatization task requires that the words are already POS-tagged.

### 4.2.5 Sentence detection

Another important linguistic task is to identify sentences within a continuous block of text. It is because many linguistic components expect the input text to be divided into

individual sentences. In fact, the task can be reduced to binary classification which decides whether a punctuation character terminates a sentence or not. For example, in the text *"The frequency 2.4MHz is used for transmission."*, the sentence should not be terminated in the middle of "2.4MHz" but after the "transmission" word. Another frequent mistake would be to treat a title of a section as a part of the first sentence of the following paragraph.

In our framework, we needed a sentence splitter which would operate on HTML document rather than on plain text files. In HTML, the XML tags introduce additional evidence for the classifier. For example, the terminating `</p>` tag indicates a high probability of sentence termination if it is placed before a punctuation character.

Similarly to the sentence detection component[5] available in the Apache OpenNLP framework, we have built a sentence splitter which uses MaxEnt classification model and we plugged it to the Stanford CoreNLP linguistic pipeline.

### 4.2.6 Named entity recognition

[96]
[95]

In computational linguistics, the term Named Entity Recognition (NER) represents a variety of tasks related to extracting relevant entities and their relations from unstructured or semi-structured text. (known as Relation Extraction)

Historically, the task was focused on the identification of real-world entities such as places, organizations or person names, hence the term "Named". Our method, discussed in this paper, also belongs to the NER/RE family. In particular, we use the supervised machine learning approach. Rather than identifying places or names, we identify entities and their relations forming a potential domain model. The NER task must also executed before the coreference resolution task (see below).

### 4.2.7 Hand-written rules and patterns

Early approaches to NER and Relation Extraction (RE) were using hand-written extractors (first proposed by Hearst in [35]). These include (i) regular expressions, (ii) special query languages, such as PMLTQ [104], TPL [109], LPath [57], Tregex [61], NetGraph [77], XPath, etc. (iii) or simply a general-purpose programming language such as Perl or Java.

The main drawback of these approaches is their inability to cope with unexpected variations. That is why the state-of-the-art NER techniques are based on some form of statistical approach. Our method also uses a statistical classification technique to learn the patterns from training data. Instead of enumerating all the possible patterns, which at some point becomes unmanageable, we just feed the classifier with more training data.

### 4.2.8 Parsing : constituency

The goal of a constituency parser is to obtain phrase structure of a sentence as depicted in Figure 4.3 on page 53 and Figure 2.1 on page 10. Non-terminals (inner nodes) are the phrase types (such as Noun Phrase, Verb Phrase). Terminals (leaves) are individual

[5] http://opennlp.apache.org/documentation/manual/opennlp.html#tools.sentdetect.detection

words. The parse tree contains unlabeled edges. The generated tree is built on top of the original sentence preserving word order.

Instead of the constituency parse trees, we use dependency representation in our approach.

### 4.2.9 Parsing : dependency

Dependency parsing is a method of deriving the syntactic structure of a sentence as a dependency graph (depicted in Figure 4.1 and 4.4) which, apart from the phrase structure, does not contain any non-terminal nodes (i.e. no noun-phrase or verb-phase nodes). Nodes of the dependency graph represent individual tokens while the edges represent grammatical dependencies between the tokens. Figure 4.2 shows all dependency relations as defined by the Stanford typed dependency representation. Dependency structure of a sentence can be deterministically derived from its constituency structure.

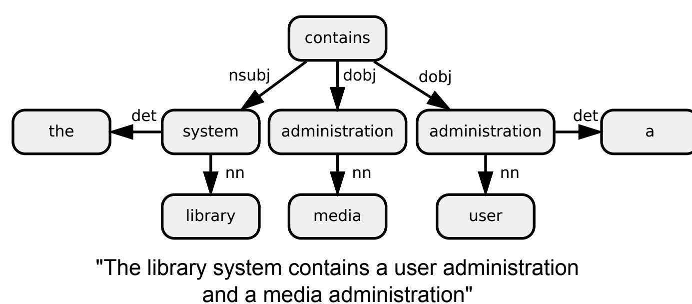



**Figure 4.1:** Stanford typed dependencies representation of a sentence.

### 4.2.10 Coreference resolution

Coreference resolution, also known as anaphora resolution, is the task of identifying which expressions in the text refer to the same entity. The Stanford CoreNLP framework contains a sieve-based deterministic anaphora resolution system[6] described in [60, 59], with the reported average accuracy of $F_1 \simeq 58\%$, to this date the best achieved performance. Therefore, we decided to include the "dcoref" annotator to our framework as a part of the linguistic pipeline. The identified coreferences are extracted into out specification model ready to be used for further prediction of software engineering artefacts.

### 4.2.11 Sentence analysis example

The results in the following example were obtained using the on-line version of the Stanford parser[7] and CoreNLP[8]. We use the same linguistic models in our framework.

[6]Deterministic Coreference resolution system is described in http://nlp.stanford.edu/software/dcoref.shtml

[7]Stanford parser web demo is available here: http://nlp.stanford.edu:8080/parser/

[8]CoreNLP web demo is available here: http://nlp.stanford.edu:8080/corenlp/

- **root** – root
- **dep** – dependent
  - **aux** – auxiliary
    - **auxpass** – passive auxiliary
    - **cop** – copula
  - **arg** – argument
    - **agent** – agent
    - **comp** – complement
    - **acomp** – adjectival complement
    - **attr** – attributive
    - **ccomp** – clausal complement with internal subject
    - **xcomp** – clausal complement with external subject
    - **complm** – complementizer
    - **obj** – object
      - **dobj** – direct object
      - **iobj** – indirect object
      - **pobj** – object of preposition
    - **mark** – marker (word introducing an advcl)
    - **rel** – relative (word introducing a rcmod)
  - **subj** – subject
    - **nsubj** – nominal subject
      - **nsubjpass** – passive nominal subject
    - **csubj** – clausal subject
      - **csubjpass** – passive clausal subject
- **cc** – coordination
- **conj** – conjunct
- **expl** – expletive (expletive "there")

- **mod** – modifier
  - **abbrev** – abbreviation modifier
  - **amod** – adjectival modifier
  - **appos** – appositional modifier
  - **advcl** – adverbial clause modifier
  - **purpcl** – purpose clause modifier
  - **det** – determiner
  - **predet** – predeterminer
  - **preconj** – preconjunct
  - **infmod** – infinitival modifier
  - **mwe** – multi–word expression modifier
  - **partmod** – participial modifier
  - **advmod** – adverbial modifier
    - **neg** – negation modifier
  - **rcmod** – relative clause modifier
  - **quantmod** – quantifier modifier
  - **nn** – noun compound modifier
  - **npadvmod** – noun phrase adverbial modifier
    - **tmod** – temporal modifier
  - **num** – numeric modifier
  - **number** – element of compound number
  - **prep** – prepositional modifier
  - **poss** – possession modifier
  - **possessive** – possessive modifier ('s)
  - **prt** – phrasal verb particle
- **parataxis** – parataxis
- **punct** – punctuation
- **ref** – referent
- **sdep** – semantic dependent
  - **xsubj** – controlling subject

**Figure 4.2:** Hierarchy of Stanford Typed Dependencies. Further information can be found in [72],[71].

The analyzed input sentence is *"A user has only access to his own user account by using his user number."* Figure 4.4 shows (i) Tokenization of the sentence including the POS-tags. (ii) Resolved coreferences where the phrase *"A user"* is the representative mention and the two *"his"* words are the other mentions. (iii) Basic and collapsed dependencies, where the collapsed version transformed the *"to"* preposition into *"prep_to"* dependency. The constituency parse tree is depicted in Figure 4.3.

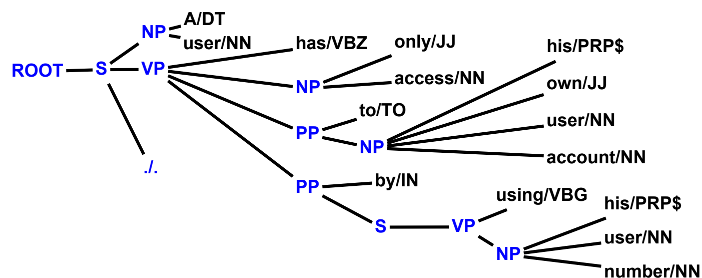
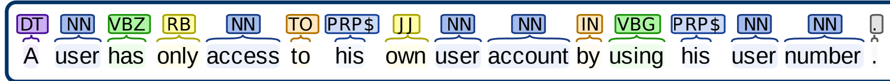
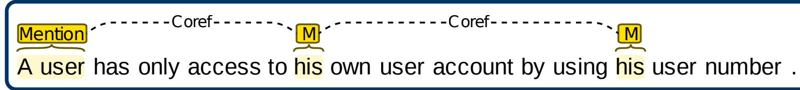



**Figure 4.3:** Generated consitituency parse tree for the sentence: *"A user has only access to his own user account by using his user number."*

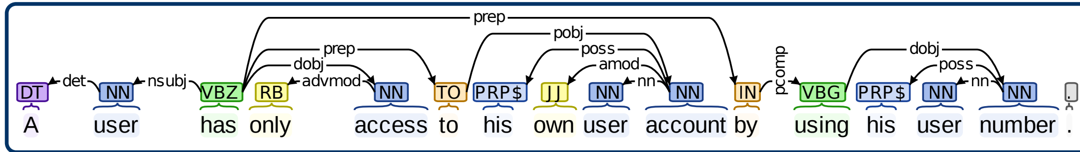
**Part-Of-Speech**




**Coreference**




**Basic Dependencies**




**Collapsed Dependencies**

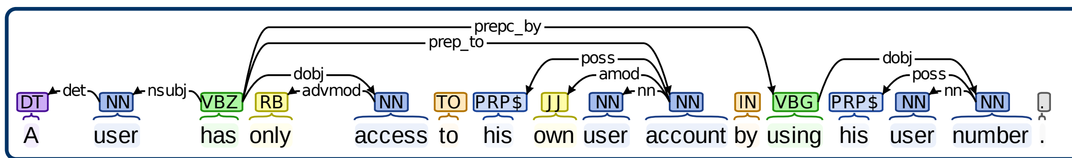



**Figure 4.4:** Sentence parsed using the Stanford CoreNLP framework.

## 4.3 Statistical classification related to our method

[96]

Before we present our method, we outline important concepts related to statistical classification and used in our method.

The task of statistical classification, in general, is to estimate the probability of category $c \in C$ occurring in the context of observed data $d \in D$. This can be written as a function $\phi(c, d)$ with a bounded real value, i.e. $\phi : C \times D \mapsto \mathbb{R}$

### 4.3.1 Features

In practice, it is useful to restrict the function $\phi$ to a particular form – a boolean matching function where $f_1, \ldots, f_n$ are features extracted from the data point $d$ and $c_j$ is some category.

$$\phi(c, d) \equiv [f_1(d) \wedge f_2(d) \wedge \ldots \wedge f_n(d) \wedge c = c_j] \tag{4.1}$$

### 4.3.2 Feature extractors and context generators

We use the term *feature extractor* for an algorithm that computes the value of a single $f_i(x)$ from an input $x$.

Each feature extractor expects a particular type of the input $x$. For example, in case of the feature $f_{flet}$ (which extracts the first letter of a given word), the input $x$ must be a word. The purpose of a *context generator* is to prepare a stream of objects that are passed to feature extractors.

### 4.3.3 Statistical classifier

A statistical classifier (binary, multi-class or probabilistic) is able to learn what data (context) leads to what category (outcome). The classifier performs the classification using a *trained classification model* which models the function $\phi$. In our method, we use a classifier which chooses the outcome value of the highest probability estimated using the trained model.

### 4.3.4 Training samples

When we train a classifier, we need to encode the Formula (4.1) above as training samples for the classifier.

In machine learning, samples are also called feature-vectors that encode relevant information about a given data-point $x$. In our case, this would be a tuple $\vec{y}$ composed of $n+1$ values ($n$ features encoding the context and 1 feature for encoding the outcome) representing a single data-point $x$:

$$\vec{y} = (f_1(x), \ldots, f_n(x), c(x))$$

Then, for $k$ data-points $x_1, \ldots, x_k$, the training samples form a matrix:

$$\begin{bmatrix} \vec{y_1} \\ \vdots \\ \vec{y_k} \end{bmatrix} = \begin{bmatrix} f_1(x_1) & \ldots & f_n(x_1) & c(x_1) \\ \vdots & \ddots & \vdots & \vdots \\ f_1(x_k) & \ldots & f_n(x_k) & c(x_k) \end{bmatrix}$$

Each column in the matrix represents a single feature, whereas rows represent individual samples. By convention, the last column is regarded as the "outcome" while the rest is the "context". A typical input required by the OpenNLP MaxEnt API conforms to the following format:

```
pos=NN  indep=agent   true
pos=VBZ indep=auxpass false
...
```

Here, the "outcome" is a boolean value depending on the "context" represented by $f_{pos}$ and $f_{indep}$ features.

### 4.3.5 Training samples in our method

To experiment with the classification performance, we need experimental samples. Therefore, we manually prepared annotated texts and domain models[9]. Our annotations represent links between a sequence of words and some element from the domain model as depicted in Figure 4.5. In this example, the words "media administration" are linked to the *MediaAdministration* domain entity. Annotations are encoded in the training data as HTML hyperlinks:

```
The <a href="MediaAdministration">media administration</a>contains an entry for each <a href="Medium">medium
    </a> in the <a href="Library">library</a>.
```

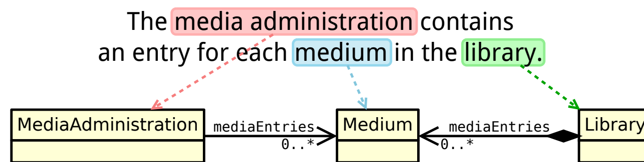



**Figure 4.5:** Links between manually annotated text and domain model

When parsing the annotated text, each sentence is enriched with automatically generated data from the Stanford parser applied on each sentence found in the input text (Figure 4.1). All the information is stored together in a single graph we call the specification model. Our context generators can then traverse the graph and generate training samples using feature extractors (see Figure 4.6 showing its meta-model)

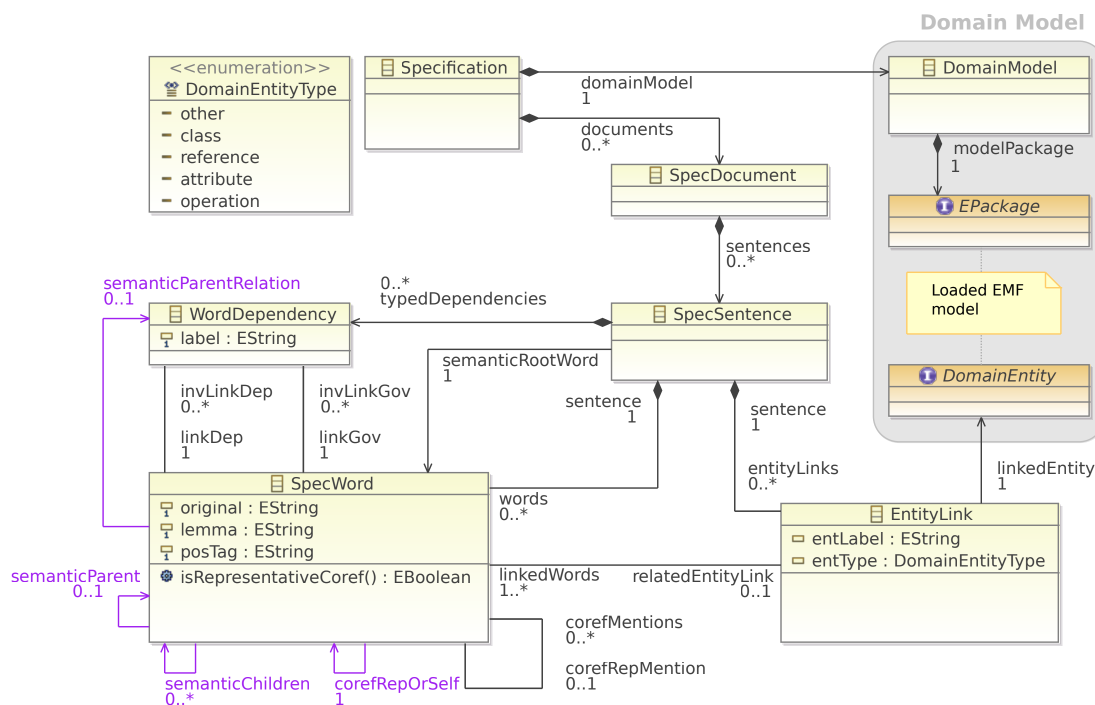



**Figure 4.6:** Specification meta-model (input data for feature extraction).

## 4.3.6 Maximum entropy models for classification

Inspired by the state-of-the art NLP tools, such as the Stanford POS-tagger [106] or other approaches, such as [79], [37], [14], we utilized MaxEnt models [69] for the classification. However, the ideas presented in our contribution should also work with other classifiers such as Bayes classifiers [68], Support Vector Machines (SVMs) [68], Perceptrons [76], Conditional Random Fields (CRFs) [56], each having their specific strengths and weaknesses.

[9] We use EMF ecore meta-model for encoding domain models.

We opted for the MaxEnt approach because it is well suited for classification tasks comprising vast amount of dependent binary features. This has been demonstrated by a number of MaxEnt implementations in field of NLP, e.g. [14]. To understand the term Maximum entropy modeling, we quote the book [69], page 589: *"Maximum entropy modeling is a framework for integrating information from many heterogeneous information sources for classification. The data for a classification problem is described as a (potentially large) number of features. These features can be quite complex and allow the experimenter to make use of prior knowledge about what types of informations are expected to be important for classification. Each feature corresponds to a constraint on the model. We then compute the maximum entropy model, the model with the maximum entropy of all the models that satisfy the constraints. ... If we chose a model with less entropy, we would add 'information' constraints to the model that are not justified by the empirical evidence available to us. Choosing the maximum entropy model is motivated by the desire to preserve as much uncertainty as possible."*

In our implementation, we use the Apache OpenNLP framework[10] which allows us to easily integrate MaxEnt classification to Java applications.

### 4.3.7 Maximum entropy Markov models

As we show later in the text, we use multiple dependent classification tasks. Unknown values to be classified are connected in a Markov chain rather than being conditionally independent. This approach is called Maximum Entropy Markov Model (MEMM). Since MaxEnt models support potentially large number of features, with MEMMs we can easily make use of *lexicalization*. It means that the features are constructed from an unrestricted set of all words rather that from a restricted set of classes. Comparing to the unlexicalized approach, we gain better prediction performance at the expense of time- and space efficiency.

## 4.4 From text to domain model in 4 phases

Figure 4.7 provides an overview of our method divided into 4 phases: (1) *Preprocessing*, (2) *Feature Selection*, (3) *Training*, (4) *Domain Model Elicitation*.

The *Elicitation phase* is a common usage of the method, however, it depends on classification models that are trained in the *Training phase*. To train a classifier, we need to find features suitable for a particular classification task which is done in the *Feature Selection phase*.

Each phase involves a different type of user. While the *Elicitation Phase* is intended for the analyst, who does not have a deep knowledge of the specification domain, the *Training Phase* involves domain experts, who can prepare training data. *Feature Selection phase* requires a scientist to design, implement and measure the performance of features.

[10] http://opennlp.apache.org/documentation/manual/opennlp.html

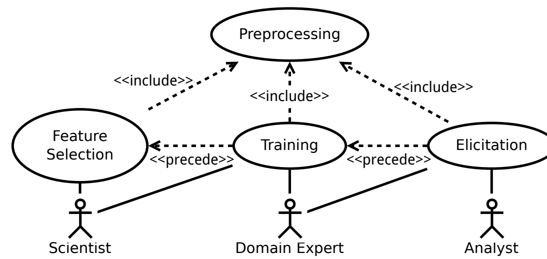



**Figure 4.7:** Usage Overview - main phases of our method.

### 4.4.1 Preprocessing phase

A specification model is the central data structure in our method (the model's meta-model is depicted in Figure 4.6). It captures relations between sentences, words, linguistic features and domain model elements. We can either build the specification model from scratch or use an existing model as a source of training samples.

Figure 4.8 shows our preprocessing pipeline in detail. The process starts by creating an empty container for the specification model. Then, an *input document* is processed by a series of annotators from the Stanford CoreNLP framework[11] on top of a common data structure, here depicted as the *Annotated Structure*.

**Annotators reused from CoreNLP are the following:**

- Tokenizer,
- POS-tagger,
- Lemmatizer,
- a set of NER classifiers,
- Lexicalized parser that produces dependency graphs for sentences (mentioned in Section 4.2.9),
- Deterministic Coreference Resolution System (mention in Section 4.2.10)

**Additionally, we added:**

- Tidy HTML Cleaner (for filtering faulty HTML),
- XML preprocessor for extracting hyperlinks and evidence for splitting sentences,
- Sentence Splitter, implemented as a MaxEnt classifier, for identifying tokens that may terminate a sentence (mentioned in Section 4.2.5).

After running the linguistic pipeline, we transform useful linguistic information into *Specification Model*. Optionally, if the pipeline is executed as a part of the training process, we load an existing domain model and resolve links from the text.

[11] http://www-nlp.stanford.edu/software/corenlp.shtml

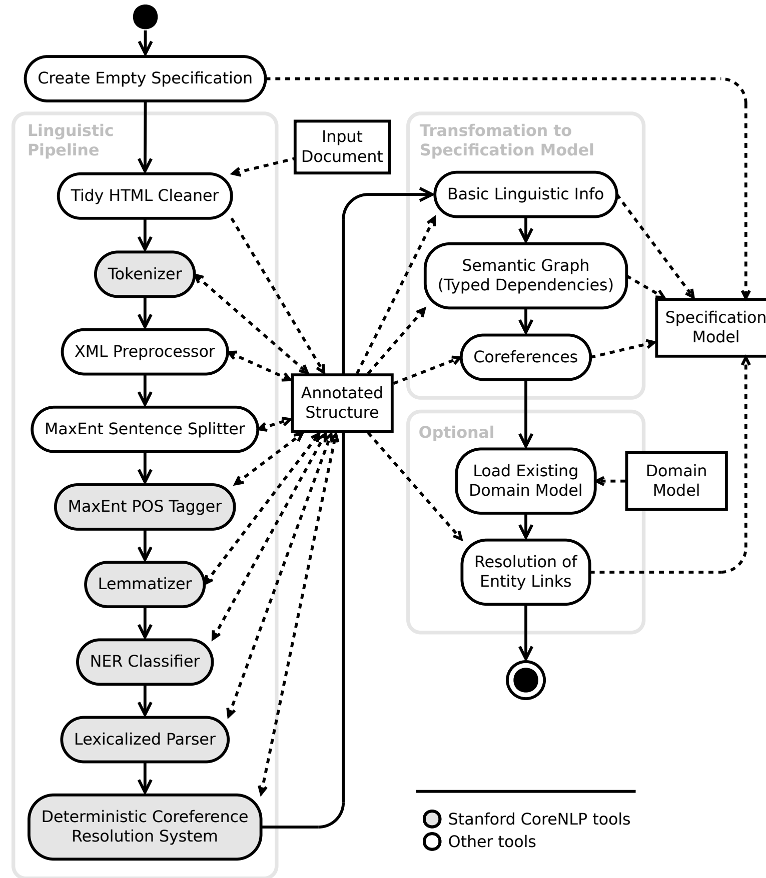



**Figure 4.8:** Preprocessing phase : Linguistic pipeline and transformation to the specification model.

### 4.4.2 Feature selection phase

The goal of the Feature Selection phase is to identify the best feature set $S_{best}$ among alternative feature-sets $S_1, \ldots, S_c$ for each classification task in terms of their prediction-performance (the number $c$ is usually lower than the number of combinations of all features). An example of a feature set could be $S = \{f_{lemma}, f_{flet}\}$ which contains two features $f_{lemma}$ (the linguistic lemma-form of a given word) and $f_{flet}$ (first letter of a given word).

Feature Selection involves (i) designing features, (ii) implementing their feature extractors and (iii) measuring the performance of combinations of these features. Therefore, as depicted in Figure 4.7, we expect that only scientists are participating in this phase. It is a hill-climbing process guided by our intuition when we measured the prediction-performance of candidate feature-sets:

$$S_{best} = \arg\max_{x} \mathbf{Perf}(S_x) \tag{4.2}$$

The function $\mathbf{Perf}$ is a suitable statistical measure (selection discussed in Section 4.5.3 on page 63).

The activity diagram depicted in Figure 4.9 shows steps involved in the feature selection phase. First, the input document containing experimental training data is

preprocessed (as already mentioned in Section 4.4.1). Based on a configuration file, we generate random combinations of feature sets.

We do not need to exhaustively evaluate all combinations. We just want to find a reasonably good combination by exploring a representative sample. Suppose, we have $n$ features. The number of all possible combinations is:

$$\sum_{k=1}^{n} \binom{n}{k}$$

To reduce the amount of measurements, we choose a parameter $m \in \mathbb{N}$ and pick a normally distributed random subset of size:

$$c = \sum_{k=3}^{n} min\left(m, \binom{n}{k}\right)$$

For example, suppose we have $n = 20$ features, we might want to set $m = 30$, which reduces the amount of combinations from $c = 1048575$ down to $c = 551$.

Each feature set is measured as follows. Training samples are generated from the specification model, the MaxEnt model is trained and evaluated using the k-fold cross validation (see Section 4.5) which yields a number of statistical measures defined in Section 4.5.3. The results of the "Feature Selection" phase are: (i) feature extractors and (ii) ranking of feature-sets according to the aforementioned statistical measures.

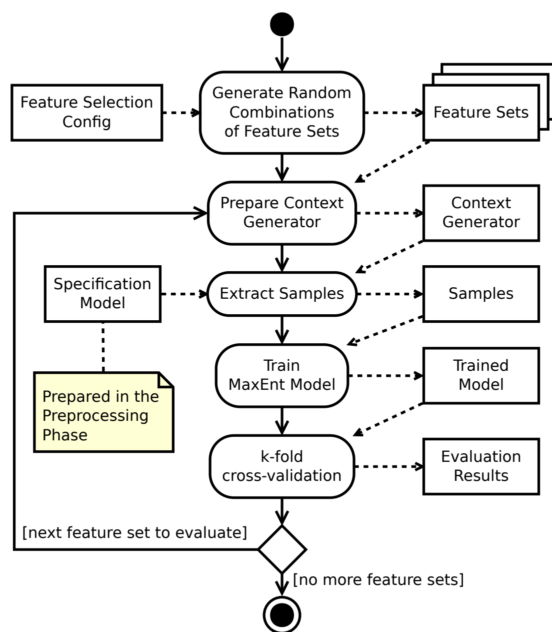



**Figure 4.9:** Feature-selection phase

### 4.4.3 Training phase

Knowing the $S_{best}$ from the feature selection phase, the classifier can be automatically retrained with new input samples, for instance training data from already finished

projects. For the purpose of our experiment, we prepared manually annotated experimental data. Figure 4.10 shows the activity diagram for the training phase. As an input, the training phase uses an existing specification model filled with training data, and a configuration file which defines feature sets $S^1_{best}$, ..., $S^n_{best}$ for training the classifier. As a result we get $n$ trained MaxEnt models.

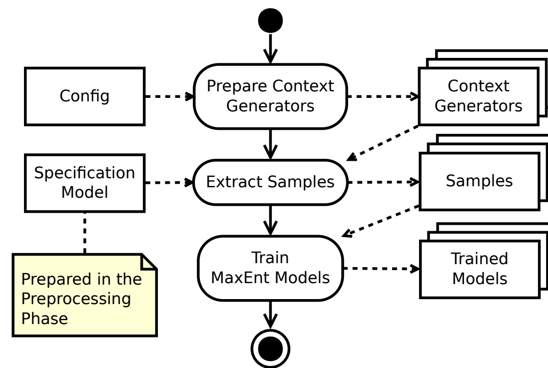



**Figure 4.10:** Training phase

### 4.4.4 Domain model elicitation phase

Figure 4.11 shows all the steps involved in the elicitation process. Several steps are classification tasks that use the trained classification models. The order of steps is important because each classification task uses the information generated in the previous step, i.e. the steps are conditionally dependent and form a Markov chain.

#### Step: Identifying words forming a domain entity

The first step is a classification task that aims at identifying words that *may* represent domain entities. For every positively identified word we add an instance of a new domain entity to the specification model. Some multi-word entities will be represented as multiple entities. For example, words "User Account" will be represented as "User" and "Account" entities at this point. Since this phase acts as a first element of the Markov chain, we have an empty domain model. Therefore we can only leverage "linguistic" features.

#### Step: Identifying multi-word entities

Here, we merge selected candidate entities into a single entity using a *sequence classification model*. We process words from left to right within each sentence and predict the features of the current word depending on the previously classified words. After the classification, a continuous sequence of labeled words is merged into a single entity.

#### Step: Deriving names for entity links

In this step, we construct the name of an entity by taking lemma forms of the words. This straightforward approach is planned to be improved in future versions of our tool.

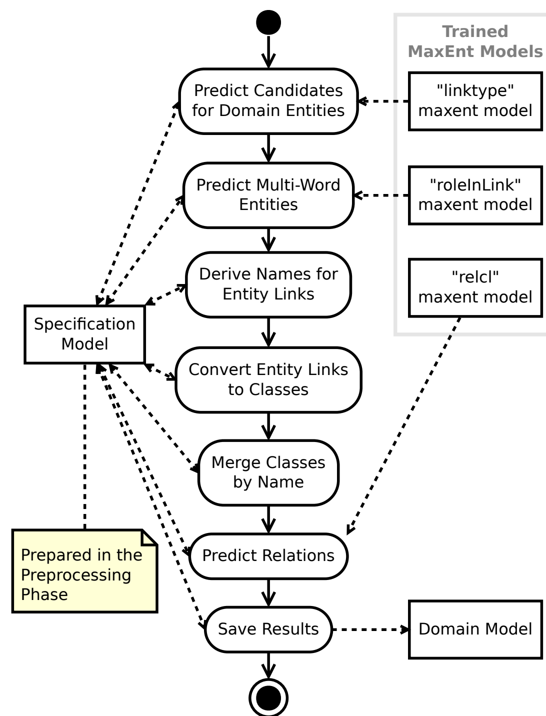



**Figure 4.11:** Elicitation phase

#### Step: Creating classes in the domain model

We transform one-to-one entity links to classes in the domain model.

#### Step: Merging duplicate classes in the domain model

Currently, we perform merging of classes based solely on the literal equivalence of names. This will also be improved in future version.

#### Step: Predicting relations

At this point, we have a number of classes in the domain model linked to the sentences in the specification document. For each sentence $s$, we take all the linked classes and generate all $n(n-1)$ combinations of ordered pairs of these classes. For a given pair $(a, b)$, we ask the classifier to predict whether there is a relation in the domain model from $a$ to $b$ in the context of the sentence $s$. (See the paragraph "**relations**" context generator in Section 4.5.5)

After the *Elicitation phase*, the specification model contains the predicted domain model which can be further refined by the analyst. Finally, when the project is finished, the refined version of the specification model can be used as a source of training data to improve the classification performance in a next project.

## 4.5 Evaluation

It is not possible to use a classifier without a proper evaluation of its performance. A classifier can perform badly even though its design may look reasonable. Moreover, since we have a number of alternative combinations of features, we want to rank them in a systematic way. The evaluation should be automated and well understood.

We have implemented an automated tool for evaluating our method and used the Library System specification (Appendix C) for the evaluation.

### 4.5.1 Training vs. testing data

In general, a supervised machine learning approach requires a representative corpus containing labeled relations between entities (usually labeled by hand) divided into (i) *training set*, (ii) *development test set* and (iii) *test set*. To achieve good performance, the classifier should be trained on a *training set* which sufficiently captures the variability of data. During the development process, when the features are designed, the *development test set* is used to assess the performance of selected features by computing statistical measures such as $F_1$ explained below. To avoid overfitting, the classifier's performance is evaluated against the unseen *test set*.

### 4.5.2 Cross-validation

When the training data is scarce (as in our experiment), "hold out" methods are used for evaluation. The most popular is the k-fold cross validation. The data is divided into $k$ subsets (folds) of equal size. Then, we can perform $k$ measurements where one fold is considered as the *test set* and the other folds are considered as the *training set*. It means that in every iteration, one of the folds is unseen to the classifier. Although, this approach is slightly biased, because we saw all the samples when designing the features, we expect that in practice the classifiers will be reevaluated on domain-specific training data and a new feature set can be selected to suit the new environment.

### 4.5.3 Evaluation metrics in the experiment

The standard approach to evaluate the performance of a classifier is to construct a confusion matrix (or contingency table) by comparing the answers from the classifier and manually labeled answers:

| | **correct** | **wrong** |
|---|---|---|
| **selected** | TP | FP |
| **not selected** | FN | TN |

**true positives (TP)** are elements correctly selected by the classifier;

**false positives (FP)** are elements incorrectly selected by the classifier;

**true negatives (TN)** are elements correctly not selected by the classifier;

**false negatives (FN)** are elements incorrectly not selected by the classifier.

From these 4 numbers, we can compute a variety of useful statistical measures as suggested by e.g. [110]:

$$Precision = \frac{TP}{TP + FP} \tag{4.3}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.4}$$

$$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{4.5}$$

$$Specificity = \frac{TN}{TN + FP} \tag{4.6}$$

$$FallOut = 1 - Specificity \tag{4.7}$$

**MCC** (Matthew's correlation coefficient) =

$$\frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{4.8}$$

The number $Precision$ represents the probability that the predicted elements are relevant. $Recall$ represents the probability that all relevant elements were predicted. For our purposes, $Recall$ is more important, because we can manually delete redundant elements from the predicted domain model. However, in machine learning, the $F_1$ measure is used because it represents both $Precision$ and $Recall$ in a balanced way. It is also possible to compute $MCC$ (Matthew's correlation coefficient) which is a balanced measure usable in situations when sizes of classified categories differ significantly. Lastly, we also compute $FallOut$ (and thus also $Specificity$) to be able to represent our results using the ROC diagram explained below.

Due to the k-fold cross validation, we obtain $k$ values for each of these measures. Thus, we compute the mean, median, standard deviation and confidence interval ($\alpha = 95\%$, assuming a normal distribution of the measured results). Confidence intervals obtained this way are shown in the diagrams in Figure 4.12.

### 4.5.4 Data used in the experiment

For the experiment, we chose a software specification describing a Library System. The input document was an HTML file (see the annotated document in Appendix C).

As a first step, one person was reading the document several times while drawing the diagram on a sheet of paper. Detected ambiguities were resolved by the analyst according to his understanding of the domain.

When the prototype model was ready, we used the standard Ecore Diagram editor available in Eclipse to design the model formally in EMF. The final model entails (1) classes, (2) relations, including inheritance, containment and cardinality, (3) attributes, including types and cardinality, (4) methods.

A simplified version of the domain model is depicted in Figure C.1. Not all visible elements were used for our experiment, we focused only on classes and relations. After polishing the domain model, we annotated the text with links to the domain model. by adding hyperlinks.:

```
<a href="#EntityName">sequence of words</a>
```

Finally, we designed two context generators and a number of features.

### 4.5.5 Classification in the experiment

In this text, we show 3 cases of classification used in our tool, where we predicted the "outcome" feature given a set of "context" features.

#### List of context generators

**words** : A stream of all words in all sentences in all documents within the specification. This context generator is used for classification of `linktype` and `roleInLink` outcomes and is compatible with features:

```
linktype       wprefix:n   wminlen:n   sprel
roleInLink:n   wsuffix:n   whascl:n    sppos
pos:n          lemma:n     whasdigit
```

**relations** : A stream of triples $(s, a, b)$, each representing a relation from class $a$ to class $b$ within the scope of a single sentence $s$. In other words, for a given sentence, we take all words that are linked to some class in the domain model. Then, we take the set of all linked classes $C$, and generate all combinations of pairs, which yields in total $\binom{|C|}{2}$ relations. This context generator is used for classification of `relcl` and is compatible with features:

```
relcl             relivf:lemma   reldepOnRoot:src
relpassroot       relivf:pos     reldepOnRoot:dest
relsemrootlemma
```

[96]

#### All features related to the "words" context generator

**`linktype`** This outcome feature extracts one of the values {*class, operation, attribute, reference, other, none*} depending on the type of the domain entity the given word is linked to.

**`roleInLink:n`** is also a multi-class outcome feature whose purpose is to identify multi-word entities, i.e. whether a given word is the starting word of an entity (`head`), a word in the middle of the entity name (`cont`), the last word (`last`), or not linked to any entity (`none`). The parameter $n$ defines the word-offset in the sentence. This feature is intended to be used in a sequence classifier, which makes decision based on the previously classified outcomes, e.g. we can use the feature `roleInLink:-1` when processing words from left to right.

**`pos:n`** Part-of-speech tag of a word. The optional parameter $n$ defines the word-offset within the sentence. For example, `pos:-1` represents POS-tag of the previous word.

**`lemma:n`** Lemma-form of a word with $n$ being the word-offset within the sentence.

**`wminlen:n`** This feature decides whether the lemma-form of the given word has at least $n$ characters.

**`whascl:n`** Decides whether the word contains a capital letter, parameter $n$ is the word-offset within the sentence.

**whasdigit** Decides whether the word contains a digit.

**wprefix:n** Prefix of the word ($n$ is the prefix length)

**wsuffix:n** Suffix of the word ($n$ is the suffix length)

**sprel** Returns the label of the semantic relation between the given word and its parent in the dependency graph generated by the Stanford Parser.

**sppos** Returns the POS-tag of the parent word in the dependency graph.

**Features related to the "relations" context generator**

**relcl** This is the outcome feature which decides whether classes $A$ and $B$ are related regardless of the sentence $S$.

**relsemrootlemma** Extracts the lemma-form of the semantic root word from the sentence, i.e. the root of the Stanford typed dependencies graph. This word is usually a verb, e.g. "contain", "manage", ...

**relpassroot** Decides whether the sentence uses a passive voice. At the moment, we just check if there is a "nsubjpass" relation in the dependency graph from the semantic root word.

**reldepOnRoot:x** Assuming the input triple (S, A, B) as a relation, this feature decides whether the semantic root word of the sentence $S$ has a direct dependant that is linked to the class $x \in \{A, B\}$.

**relivf:x** Assuming the input triple (S, A, B) as a relation, we find an intersection of paths $P_A \bigcap P_B$ in the dependency graph, where $P_A$, $P_B$ are paths from words (linked to classes $A$, $B$) to the semantic root word of the sentence $S$. Then we take the first verb from the intersection and return its lemma form or POS-tag depending on the parameter $x \in \{lemma, pos\}$.

### 4.5.6 Results of the experiment

[96]

Finally, we measured the performance of our classifiers. The results are depicted in Figure 4.12. We use two types of diagrams to graphically represent the classification performance. The first type is a line plot showing $Precision$, $Recall$ and $F_1$ measure. The second is a scatter plot representing the ROC-curve (relative operating characteristic) that illustrates the true positive rate ($Recall$) vs. the false positive rate ($FallOut$).

In the ROC diagram, an ideal classifier would be placed in the upper left corner, while the worst classifier would be placed on the $y = x$ diagonal.

In both diagrams, we also show the confidence intervals to better illustrate the distribution of multiple measurements.

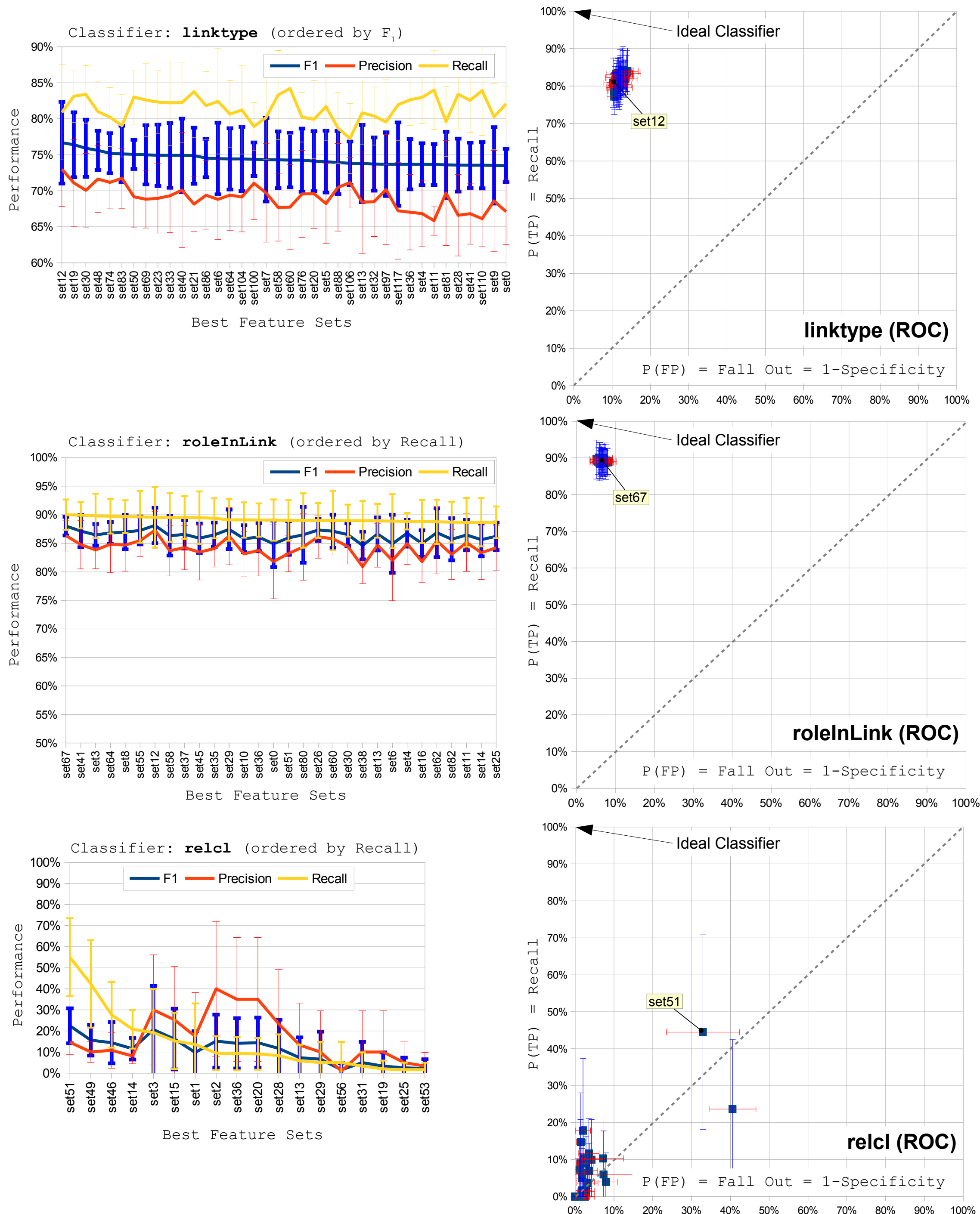



**Figure 4.12:** Evaluation results for classification of "linktype", "roleInLink" and "relcl" outcome features.

**"linktype" classifier:** For the classification of `linktype`, we chose 19 context features (denoted as "set12") with the performance $Recall = 80.9\%$, $Precision = 73\%$, $F_1 = 76.7\%$. The chosen context features are:

```
lemma      wminlen:2  wprefix:1  wsuffix:2  pos
sprel      wminlen:3  wprefix:2  wsuffix:3  pos:1
sppos      wminlen:4  wprefix:3  wsuffix:4
whascl     wminlen:5  wprefix:4
whasdigit  wminlen:6
```

**"roleInLink" classifier:** For the classification of `roleInLink`, we chose 20 context features (denoted as "set67") with the performance $Recall = 90\%$, $Precision = 86.3\%$, $F_1 = 88\%$. Although, there was a result with higher $F_1$, we chose "set67" due to its higher $Recall$. The chosen context features are:

```
lemma      wprefix:1  wminlen:2  roleInLink:-1  pos
sprel      wprefix:2  wminlen:4  roleInLink:-2  pos:-1
whasdigit  wprefix:3  wminlen:6
wsuffix:4  wprefix:4
```

**"relcl" classifier:** For the classification of `relcl`, we chose 2 context features (denoted as "set51") with the performance $Recall = 55\%$, $Precision = 14.6\%$, $F_1 = 22.4\%$. The chosen context features are:

```
reldepOnRoot:src
relivf:pos
```

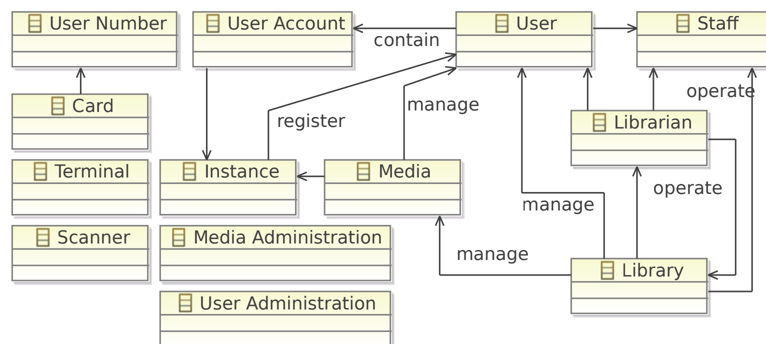



**Figure 4.13:** Domain model automatically predicted in our experiment without redundant entities and relations deleted manually.

## 4.6 Summary of Chapter 4

To summarize the results, the performance of `linktype` and `roleInLink` classification models is impressive. It is because we implemented a wide range of useful features from which the evaluator could pick the best combination. In case of the `relcl`, we only implemented 5 features so far which did not give much choice to the evaluator.

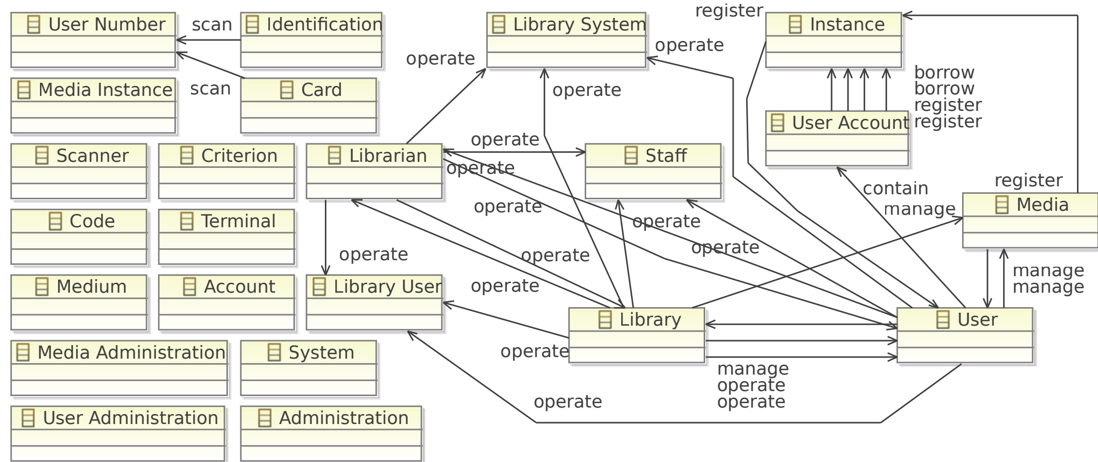



**Figure 4.14:** Domain model automatically predicted in our experiment as generated by the tool, without manual editing.

Moreover, only 2 features were actually useful for predicting `relcl`. Even with these constraints, we achieved a relatively high $Recall = 55\%$ performance. We expect that the accuracy will be dramatically increased by implementing more features and using larger training set.

After applying the selected classification model to our example specification, our tool finally generated the prototype domain model depicted in Figure 4.14. Due to the low precision of the `relcl` we had to clean the generated model manually (Figure 4.13). However, we only deleted redundant elements from the model without adding any new elements. The similarity to the manually prepared model (Figure C.1) is obvious.

**The elicitation process can be further improved as follows:**

1. By designing more features for the `relcl` classification task.

2. When deriving names for entity links, a more sophisticated normalization can be employed.

3. We can also improve the identification of duplicate entities, e.g. by defining an appropriate hashing function.

4. The pipeline can be extended with the following steps:

   - identification of the aggregation relation,
   - identification of attributes and their types,
   - identification of the inheritance relation.

**Implementation:** The implemented framework, titled *Prediction Framework*, is a headless (command-line-based) Eclipse-based product composed of multiple OSGi bundles. Some of the bundles contain linguistic and statistical models. The application demonstrates all 4 phases mentioned above: (1) preprocessing, (2) feature selection,

(3) training, and (4) elicitation. At the time of writing this text, a demo application running on Linux is available for download at:

`http://vlx.matfyz.cz/Projects/PredictionTool`

When started, without any arguments, the application shows available tools:

```
$ ./eclipse
Available tools are:
- FeatureSelectionPhase
- CreateEmptySpec
- ExportDomainModel
- TrainingPhase
- ResolveLinks
- MaxentTrainer
- ShowExtractors
- ElicitationPhase
- CleanAnnotatedDoc
- ExtractSamples
- LoadDomainModel
- LoadDocument
```

The domain model elicitation can be executed using the following commands (preprocessing + elicitation):

```
$ ./eclipse CreateEmptySpec  spec.xmi
$ ./eclipse LoadDocument     spec.xmi document.html
$ ./eclipse ElicitationPhase spec.xmi
```

Feature selection and training also requires loading of the existing domain model and resolution of links:

```
$ ./eclipse CreateEmptySpec spec.xmi
$ ./eclipse LoadDocument    spec.xmi document.html
$ ./eclipse LoadDomainModel spec.xmi dmodel.ecore
$ ./eclipse ResolveLinks    spec.xmi

$ ./eclipse TrainingPhase spec.xmi train.properties
```

. . . or optionally for feature selection:

```
$ ./eclipse FeatureSelectionPhase eval.properties
```

# Chapter 5

# Related work

This chapter summarizes existing work in areas of requirements engineering that are related to the verification of use-cases and NLP. Some of the referenced papers below are listed in multiple sections depending on the topics they cover.

## 5.1 Systematic reviews

To get a high-level view on some area of research, it is always worth looking at systematic reviews. Here, we would like to mention two papers that conducted such reviews in the area of requirements engineering.

In 2009, the authors of [20] found 5198 publications about requirement engineering spanning the years from 1963 through 2008. In 2010, the authors of [113] conducted a systematic review of transformation approaches between user requirements (in textual form) and analysis models (mostly UML-based). The authors searched for publications in electronic databases (i) IEEE Xplore, (ii) ACM Digital Library, (iii) Compendex, (iv) Inspec, and (v) SpringerLink.

Next, they searched in peer-reviewed journals (i) IEEE Transactions on Software Engineering, (ii) Automated Software Engineering, (iii) Requirements Engineering Journal, (iv) Journal of Natural Language Engineering, (v) ACM Transactions on Software Engineering and Methodology, (vi) Journal of Systems and Software, Software and Systems Modeling, (vii) Information and Software Technology, and Data and Knowledge Engineering and conference proceedings (i) ACM/IEEE International Conference on Software Engineering, (ii) IEEE International Conference on Software Maintenance, (iii) IEEE/ACM International Conference on Automated Software Engineering, (iv) IEEE International Conference on Model Driven Engineering Languages and Systems and the former UML workshops, (v) IEEE International Requirements Engineering Conference, and finally, in several software engineering textbooks.

After applying the inclusion/exclusion criteria they identified 16 different approaches from 20 studies out of which (i) 12 were focused on deriving structural elements, (ii) 9 focused on deriving behavioral elements, (iii) 5 focused on generating complete analysis models. These approaches were then compared using multiple criteria, such as: (i) Level of automation. (ii) The way, how requirement were represented. (iii) Difficulty of documentation for users. (iv) Tool support. (v) Intermediate models used during the transformation. (vi) Analysis models that can be generated (structural /

behavioral) (vii) Efficiency of the transformation (how many steps) (viii) Traceability management support.

The papers [10], [24], [42], [62], [103] evaluated in the aforementioned systematic reviews are further discussed in the rest of this chapter and compared to our FOAM method and prediction framework.

## 5.2 NLP in requirements engineering

One of the most important areas in requirements engineering is the application of Natural Language Processing (NLP) techniques. The literature discusses various NLP approaches ranging from classification of documents to semantic analysis of their content. Here is a list of approaches closely related to the topics of this thesis:

- Classification of documents and detection of parts containing requirements (e.g. [107] discussed in detail in Section 5.6 on page 75).
- Elicitation of requirements (e.g. [115] discussed in detail in Section 5.7),
- Semantic analysis of requirements documents (e.g. [107, 48, 10, 24] discussed in the following sections).
- Establishing of traceability links between formally specified requirements and the text (e.g. [42] discussed in detail in Section 5.4 on the facing page).
- Detection of ambiguities in requirements documents and filtering of duplicates (discussed in detail in [31] below).

The work of Gleich et al. [31] focuses on automated ambiguity detection in requirements documents. By performing only the lexical and syntactic analysis, they reported to have detected four times as many ambiguities than an average human analyst while keeping the number of false positives low. Their tool can also provide an explanation for each detected ambiguity. From the NLP point of view, the tool uses only the POS-tagger. Then, a set of hand-crafted regular expressions operate on the POS-tagged sentence to detect potential ambiguities. No concept extraction is used, the approach focuses just on ambiguity detection. It would be interesting to use this tool as a preprocessing step in our linguistic analysis of specification documents. Our pipeline already extracts a much richer model of the natural language, not just POS-tags. Moreover, it would then be possible to improve the performance of [31] by employing statistical classification alongside the regular expressions approach.

## 5.3 Controlled natural languages

Controlled language, also called restricted languages, stand in the middle between the unrestricted natural language and a formal models. They use a subset of the natural language that can be described by a grammar similarly to programming languages. Not only they avoid ambiguity of natural language, they are also easier to process, requiring simpler parsers or other linguistic tools. The tools usually provide a context-aware assistant that simplifies manual writing of text conforming to the defined grammar.

**Controlled languages can be divided into three groups by their purpose:**

- CLs intended to present technical documents in an unambiguous way to human readers, e.g. [2],
- CLs making multilingual machine translation of technical documents more effective and efficient, e.g. [45],
- CLs helping authors write specifications from which machines can acquire knowledge and build models, e.g. [84, 21, 51, 108].

Grammar restrictions and the language expressiveness should be well balanced. In [114], the authors described a set of restriction rules and a modified use-case template with the aim of reducing ambiguity and facilitating manual derivation of initial analysis models. The proposed restriction rules have been evaluated in a case-study, where the specifications conforming to the rules were compared to the same specifications without applying the rules.

The paper [108] describes ProjectIT-RSL, a controlled language based on a set of common linguistic patterns found in requirements documents, in particular, describing interactive systems. By applying the patterns, it is possible to extract concepts and their relations from the text which are then stored in a model as actors, entities, properties (attributes) or operations (activities). The method is focused on keeping the specification consistent with a derived model in order to support model-driven development. The method is implemented within a CASE tool ProjectIT workbench that supports various software engineering activities, such as project management, analysis and design, code generation. The paper describes an interactive text editor, similar to a word processor, that warns the writer about possible violations of rules for writing requirements. The tool is implemented as a set of Eclipse plugins.

## 5.4 Use-case templates

To simplify the analysis of use-cases and to avoid ambiguities, various templates have been proposed in the past to structure textual use-cases. The comparison of several approached has been nicely presented in the paper [114] from which we have taken the Table 5.1 and added two additional columns to highlight how FOAM tool (the last column [94]) and the older Procasor tool (column [25]) relate to these approaches.

The relation between the approaches in Table 5.1 and FOAM can be summarized as follows:

- FOAM does not rely on use-case descriptions when capturing the behavior of a use case. It is an optional field which is preserved during the transformation and may be a subject for further linguistic analysis, e.g. in our prediction framework.
- Instead of preconditions and postconditions, FOAM utilizes just the explicitly defined precedence relation.
- In Procasor, the notion of actors was used during the linguistic analysis to extract parameters of an action from a sentence. FOAM, on the other hand is focused on sequencing of actions without considering actors. The necessary semantics of

actions is encoded by the use-case template and annotations attached to use-case steps.

- Custom annotations can be defined in FOAM and attached to use-case steps. They are also preserved during all stages of the transformation. This way, it is possible to capture cross-references to high-level requirements, which is useful when external tools are involved in the verification process.

| **Field** | a | b | c | d | e | f | g | h | i | j | k |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Use case name | * | * | * | * | * | * | * | * | * | * | * |
| Description | * | * | * | * | * | * | * | * | * | | |
| Precondition | * | * | * | * | * | * | * | | * | | |
| Postcondition | * | * | * | * | * | * | * | | * | | |
| Basic flow | * | * | * | * | * | * | * | * | * | * | * |
| Alt. flow | * | * | * | * | * | * | * | * | * | * | * |
| Primary actor | * | | | | | | * | * | * | * | |
| Secondary actor | | | | | | | * | * | * | * | |
| Scope | * | | | | | | | | | * | |
| Level of abstraction | * | | | | | | | | | | |
| Stakeholders and interests | * | | | | | | | | | | |
| Special requirements | | | * | | | | | * | | | |
| Ext. points | | | * | | | | | | * | | |
| Exc. path[1] | | | | * | | | | | | | * |
| Crossref. to high-level req. | | | | | | | | * | | | * |
| Extension | * | | | | | | | | | * | * |
| Variation | * | | | | | | | | | * | * |
| Precedence | | | | | | | | | | | * |

**Table 5.1:** Summary of use-case templates based on the work of Yue et al. [114]. Each column in this table represents a single paper: a = [17], b = [43], c = [54], d = [55], e = [58], f = [62], g = [103], h = [42], i = [114], j = [25], k = [94].

## 5.5 Extended use-case models

The UML standard defines "include", "extend" and "generalize" relations between use-cases. However, we can also find other relations in literature that makes use-case specification clearer. For example, Doug Rosenberg proposed in [85] to use "precedes" and "invoke" relations in requirements specifications. The precedence was also adopted in the paper [83] and in the example specification [29]. In the papers [11] and [3], authors discuss the control-flow of common use-cases, variant use-cases, component use-cases, specialized use-cases, ordered use-cases and their relations such as the *uses*-relation, the *extends*-relation and the *precedes*-relation. In their terminology, our FOAM method can be characterized as using *variant* and *ordered* use-cases because we focus on verification of use-cases that are related using the "include" and "precede" relations and allow extension and variation of use-case scenarios.

[1]Exception paths are distinguished from alternative flows since they are taken when errors occur.

## 5.6 Modeling static structures from requirements

An automated extraction of object and data models from a broad range of textual requirements is outlined in [65]. The extraction is based on a set of predefined rules (patterns encoded as context-free grammar) that are used against the constituency parse trees (as opposed to dependency parsing in our linguistic pipeline). This approach also requires a manual specification of domain entities prior to the actual language processing. Another difference, comparing to our domain model elicitation process, is that [65] does not attempt to resolve coreferences. The outcome of [65] is a list of facts about entities, attributes and operations, such as:

```
(OBJECT (:TYPE FUNCTION) (:VALUE "entry"))
(OBJECT (:TYPE ENTITY) (:VALUE "patient"))
(OBJECT (:TYPE ATTRIBUTE) (:VALUE "age"))
```

A similar approach can also be found in the TESSI tool [53]. TESSI utilizes handwritten grammatical templates for matching patterns inside the constituency parse trees of a sentence. The pattens were designed to identify potential classes, attributes and relations. The TESSI tool can semi-automatically build a UML use-case model from text. It is also possible to verify consistency and completeness of the model with the help of a domain ontology prepared manually by the user, as described in [52]. The verification employs description logic reasoners (Pellet, RacerPro and Jess) that verify the constraints imposed by the given ontology. Moreover, the TESSI tool can generate textual description from use-cases, sequence diagrams and state machines for the purpose of reviewing the requirements analysis models.

Apart from our statistical approach, TESSI utilizes hand-written rules for identifying domain entities. These manually fine-tuned rules work well for a particular style of writing or a particular domain. However, they require redefinition when applied to a new domain. In contrast, our approach requires just retraining of the classification models using data from the new domain. Surely, it can be argued that the selection of features would require reevaluation as well. Thus, we designed our prediction framework to simplify the reevaluation in an automated way.

Another example of an approach that extracts class diagram out of textual requirements written in natural language is [36]. The linguistic pipeline is implemented using the GATE framework[2]. Using a set of predefined rules, the tool can extract candidate elements of the class diagram. In this process, a domain ontology, prepared beforehand manually by the user, is used. Again, in contrast to our statistical approach, the aforementioned method uses a hand-crafted set of rules and heuristics to extract the domain model. Even though the paper claims a Performance and Recall of more than 80%, there is no explanation in the paper of what these figures actually mean. On the other hand, in this thesis, we have discussed in detail the design and performance of each classification model employed in the pipeline.

In [107], the Requirements Analysis Tool (RAT) is introduced that supports wide range of syntactic and semantic analyses of requirements documents. Using a controlled syntax and a use-defined glossary, the tool allows the user to write documents in natural language that are easier to read and understand. RAT performs a syntactic analysis of sentences and potential problematic phrases are highlighted to the user. As a set of "best practices", the tool considers the following points: (1) Writing complete

[2] http://gate.ac.uk

sentences that have proper spelling and grammar. (2) Using of active voice instead of passive voice. (3) Using terms consistently and as defined in the glossary, avoiding multiple phrases that refer to the same concept. (4) Writing sentences in a consistent fashion starting with the agent/actor, followed by an action verb, followed by an observable result. (5) Clearly specifying the trigger condition that causes the system to perform a certain behavior. (6) Avoiding the use of ambiguous phrases. An experimental version of RAT also performs semantic analysis using domain ontologies and structured content extracted from requirements documents during syntactic analysis. The semantic analysis, based on semantic Web technologies, detects conflicts, gaps, and interdependencies between different sections (corresponding to different subsystems and modules within an overall software system) in a requirements document. The ontology (model) created this way contains relations between entities of the domain model, requirements and relations among them. The model shares some similarities with the specification model in our prediction framework. However, instead of capturing the relationships among whole requirements, we focus on relationships among linguistic artefacts and entities in the domain model. It is because, we aim at extracting statistical information for our classifiers from these artefacts.

In [46], a semi-automated approach was proposed that learns which grammatical constructs represents which model elements. Within an interactive tool, the user marks word sequences and assigns the element type and its container. The tool relies on the document structure represented as a tree of sections/subsections/sentences (discourse model), whereas our method uses relations between entities across the whole specification.

Other approaches that, at some point in their pipeline, generate static structures similar to our method, are [34, 86, 10, 108, 41, 115, 22, 4, 23]; most of them are discussed in the following sections.

## 5.7 Modeling dynamic structures from requirements

In [48], a method for deriving message sequence charts (MSCs from International Telecommunication Union similar to UML sequence diagrams) from textual scenarios is described. The process has been extended by the same author in [47] with the aim of avoiding sentences that should not be translated into MSC messages. Further, in [49] the author also tackled the issue of sentences in passive voice. He showed on a case study (Instrument Cluster [3]), how to treat compound sentences and passive voice. These studies aim at extracting the formal model of the behavior encoded in textual form, but not the actual verification of its consistency. On the other hand, we decided to capture the behavior using FOAM annotations and rather focus on the verification aspect.

Authors of [87] elicit user requirements from natural language and create a high-level contextual view on system actors and services – Context Use-Case Model. Their method uses the Recursive Object Model (ROM) [115] which basically captures the linguistic structure of the text. Then, a metric-based partitioning algorithm is used to identify actors and services using information from the domain-model. The method generates two artefacts – (1) a graphical diagram showing actors and services and (2)

[3] http://www.empress-itea.org/deliverables/D5.1_Appendix_B_v1.0_Public_Version.pdf

textual use-case descriptions. Unlike [87], where every sentence is transformed to a use-case, we divide the text into use-cases and further into use-case steps. Such a granularity is necessary when identifying components and modeling the dependencies among them.

In [10], an environment for analysis of natural language requirements is presented. The result of all the transformations is a set of models for the requirements document, and for the requirements writing process. Also the transformation to ER diagrams, UML class diagrams and Abstract State Machines specifying the behavior of modeled subsystems (agents) is provided. Unlike our generator, however, CIRCE does not generate an executable applications, just code fragments.

Authors of [24] proposed an approach (called Metamorphosis) for transforming textual use-cases to interaction models (sequence diagrams). They represent specifications as 3 models describing their syntactic, semantic and conceptual aspects. The transformation is driven by predefined patterns applied on the models. Each sentence yields an "interaction fragment". When these fragments are combined together, the complete interaction diagram is constructed. The paper, however, does not mention any specific NLP tools employed in the sentence analysis.

In [101], a notation for representing use-case specification is presented. This model is more complicated than what we use in FOAM. Similarly to FOAM, the notation describes use-cases, their scenarios with steps and branching, inclusion and sequencing constraints. However, it also models lifelines, messages, pre/post-conditions, activities, actors and other elements that are commonly used in requirements specifications. The main focus of [101] is to capture various use-case specification in a single model while FOAM focuses on verification of the control flow in use-cases.

## 5.8 Formal semantics of requirements specification

In [98], a formal semantics based on Labeled Transition Systems (LTSs) is proposed for use-cases containing *extensions* and *include steps*. The authors utilize LTSs to automatically detect *livelocks*. They also propose a method for verifying *refinement* of use-case models, namely checking their equivalence and deterministic reduction. A tool *Use-Case Model Analyzer* is mentioned which assists the developer in the automated verification. All of the checks are focused on global properties of use-case models. The same authors also wrote a number of papers about mapping use-cases into different formalisms – POSETs [100], finite state machines [97] and LTS+POSETs [99]. As opposed to FOAM, the properties to be verified are pre-defined (FOAM allows for user-defined properties) and branching scenarios are not considered.

In [102], textual use-cases are formalized via reactive Petri nets, taking into account the *include* and *extend* Unified Modeling Language (UML) relationships and sequencing constraints using pre/post-conditions. The method assumes that use-case steps comply with a restricted English grammar. The approach does not allow expressing other relationships and constraints. This is similar to our FOAM method because we use an explicitly modeled precedence relation as a sequencing constraint. Instead of verifying petri-nets, we verify LTSs using a model-checker (currently a symbolic model-checker).

Related to FOAM are also the methods that map use-cases into the UML activity

or sequence diagrams [8, 111, 114, 112, 9]. These works focus on the *generalization*, *include*, and *extend* relationships in UML. However, to verify temporal constraints within these diagrams, they would need an additional transformation to a model understood by a model-checker, which in FOAM is already provided.

There are also many approaches aiming at formalizing UML models in general. For instance in [27] the authors propose an automated method for translating UML sequence diagrams into Petri nets for evaluating reliability of software architectures. Their method uses annotations in the form of stereotypes based on the UML profile for QoS and Fault Tolerance [80]. In contrast to FOAM, these approaches rely on a model in UML that already provides a semi-formalized input in the form of annotations.

In [51], a toolkit SPIDER for analysis of UML models is presented. The authors designed a method for verifying temporal properties in UML activity diagrams using the SPIN model-checker. The temporal properties are specified in natural language which is mapped by the tool to a predefined set of patterns (as proposed by Dwyer et al. [26]) representing LTL properties to be verified. The tools aims at novice users who are not experts in temporal logic. If they follow the presented controlled language, the tool can automatically derive and verify the properties. In FOAM, we use formally defined temporal properties. We assume that experts in temporal logic would encapsulate semantics of temporal constraints into a simpler FOAM annotations.

Authors of [84] demonstrate a MDA-compliant approach to development of autonomous ground control system for NASA directly from textual requirements, called Requirements-to-Design-to-Code (R2D2C) approach. Their method relies on a special English-like syntax of the input text defined using the ANTLR grammar. User requirements are first transformed to CSP (Hoare's language of Communicating Sequential Processes) and then Java code is generated. The R2D2C is more suitable for modeling the behavior of autonomous agents while FOAM targets directly the specification and verification of textual use-cases.

## 5.9 Consistency of computational models

Model-checking of the UML use-cases using SPIN is proposed in [88]. This method assumes that pre/post-conditions of use-cases are expressed in first-order predicate logic. A graph representing the possible sequences of use-cases is constructed from the pre/post-conditions similarly to our *precede* relation. Even though the method supports branching in scenarios, it is restricted to extensions only. Also the *include* relation is not supported. Moreover in contrast to FOAM, the method assumes that use-cases are already provided in a formal notation (predicates). Also the LTL formulae to be verified by SPIN are constructed from the pre/post-conditions only.

Several languages and formalisms for behavior modeling of software systems have been proposed. They range from very generic ones (e.g., process algebras [38, 75]), to those specific to components (e.g., Darwin [66], Interface automata [7], or Threaded Behavior Protocols [50]).

## 5.10 Use of ontologies

Many authors, for instance [107, 52, 36] already mentioned in previous sections, proposed to use ontologies in areas of requirements engineering to capture relations among various artefacts and to represent domain knowledge. (A review of similar approaches can be found in [13]). An ontology can then be used for word sense disambiguation, detection of entities and relations.

The approach is not limited to requirements engineering. For example, the authors of [39] applied semantics-based information extraction for detecting economic events. Instead of an ontology, we use just a fixed meta-model for representing all linguistic information and domain model elements.

Authors of [22] employed dependency parsing and graph query patterns to extract information about traffic accidents from newspapers to a predefined ontology. For the linguistic analysis, the authors used tectogrammatical (deep syntax) dependency trees similar to the Stanford typed dependencies we use in our tool. In this aspect, their approach is very similar to our linguistic pipeline. When mapping the linguistic representation to the ontology, they employed a fixed set of rules expressed in the Netgraph [77] syntax. When the ontology is filled, it is possible to run various queries such as how many people were injured. They also trained the tool with examples of relevant and irrelevant sentences from the perspective of the traffic accidents ontology.

# Chapter 6

# Conclusions

Recent studies, such as [33], confirmed a well known principle that clear requirements and specifications are the critical success factors in software engineering. If the first phases of a project are not handled correctly, the project will probably fail or go over budget. From all the relevant artefacts, textual use-cases and domain models are the most important ones. They belong together; describing static structure and also the dynamic aspect of the developed system in a platform independent way.

This thesis aimed at designing an approach that would play well with specification documents written in natural language, to allow analysts to capture dependencies among use-case steps and also precedence among use-cases. Second, we wanted to support statistical inference of software engineering artefacts, such as the domain model. Looking back at the goals from Section 1.3, this thesis can be summarized as follows:

**G1** Formalize behavior of textual use-cases,

**G2** Design a method for Verification of use-cases,

**G3** Combine linguistic and software engineering artefacts.

**Fulfilling the goals G1 and G2:** We introduced the FOAM method/tool that attempts to help analysts write use-case specifications. The main advantage of the method is its ease of use. In particular, it works with use-cases in their natural language form and requires only a few basic annotations to be inserted into the text. The *flow annotations* formally capture the behavior of use-cases, while the *temporal annotations* capture dependencies among use-case steps. We showed that the quality of a use-case specification annotated this way can be improved to support the iterative development process. At the same time, the way we designed the system of temporal annotations gives a sufficient variability for different application domains. Users can define their own temporal constraints relevant just in their own problem space.

Using the presented tool, we can quickly generate a HTML report describing the whole use-case model. The report also explains the detected errors when temporal constraints are violated. The architecture of our tool is modular and extensible. Each transformation phase is clearly separated with well-defined meta-models describing its inputs and outputs.

Before implementing the tool, we formalized all the involved transformations using platform-independent inference rules. In FOAM, we transform the text of *annotated use-cases* into a *use-case model* which is further transformed into an *automaton* with a precisely defined semantics. This structure is then converted to the input language of the NuSMV model checker which finally verifies the specification.

**Fulfilling the goal G3:** In the second part of the thesis, we also presented our framework for statistical classification of software engineering artefacts combined with linguistic features. We demonstrated the approach on a tool that can predict a prototype domain model from text. An integral part of our prediction framework is the automated evaluation of prediction performance of the design classification models. We evaluated 3 models employed in the domain model elicitation process and showed how this approach can be extended in future. We demonstrated two classification models with a very high prediction performance and one suboptimal model. The particular measured performance figures are not that important as the principle itself. We showed that by designing a wide range of features, our feature selection component is able to identify the best performing classification model.

## 6.1 Future work

Currently the FOAM annotations have to be added manually during the preparation of a specification. We plan to extend our prediction framework to semi-automatically predict flow and temporal annotations.

The usual verification speed we encountered so far has always been just few seconds. Most of the time is spent in JVM initialization, model-to-model transformations and serialization/deserialization of models. The actual NuSMV verification time has been negligible so far. However, we plan to implement other model-checking backends, such as SPIN [40] (namely its Java reimplementation – Spinja [44]). Then we would be able to compare the speed of those model-checkers. The NuSMV model-checker we are currently using cannot utilize multiple CPU cores, whereas SPIN-based implementation might benefit from SPIN's multi-core model checking algorithm.

Sentences of use-case steps in FOAM can be written in unrestricted natural language as long as they are organized in scenarios of a defined structure. The current implementation does not focus on handling freely formatted input text which is something we would like to tackle in future. The input should be a HTML document containing annotated use-cases together with other text. We already work with this kind of specification in our prediction framework.

At the moment, we have two separate tools – first is the FOAM tool for verification of use-cases, second is the prediction framework containing the component for predicting domain models from text. We want to integrate both tools into a single framework with an Eclipse-based GUI. At the same time, we still want to keep the pure command-line interface available. Such integration should be straightforward, since both tools are already modularized into OSGi bundles.

# List of Tables


3.1 Translated temporal operators from TADL to NuSMV. . . . . . . . . 31
3.2 Analysed projects before creating the referential use-case specification 33
3.3 Selected properties from the quantitative UCDB analysis . . . . . . . 34
3.4 Results of the FOAM case study. . . . . . . . . . . . . . . . . . . . . 41

4.1 POS-tags as defined by the Penn Treebank Project . . . . . . . . . . . 50

5.1 Summary of use-case templates . . . . . . . . . . . . . . . . . . . . . 74


# List of Figures


1.1 Techniques used for requirements elicitation . . . . . . . . . . . . . . 2
1.2 Requirements formality of modeling notation . . . . . . . . . . . . . 2
1.3 Example of use-cases sharing an artefact relevant for the sequencing of actions. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.4 An inconsistency introducing variation added to the specification. . . 3
1.5 An inconsistency resolving extension added to the specification. . . . 4

2.1 Example parse tree as used by the Procasor tool . . . . . . . . . . . . 10
2.2 Example input and output format used by the Procasor tool . . . . . . 11
2.3 Example domain model used as an input to the JEE generator . . . . . 12
2.4 System usage overview of the J2EE generator. . . . . . . . . . . . . . 12
2.5 Architecture of the uc2comp generator. . . . . . . . . . . . . . . . . . 13

3.1 Overview of the verification method . . . . . . . . . . . . . . . . . . 16
3.2 Example of a consistent specification using annotated use-cases. . . . 17
3.3 Examples of custom annotations (templates) defined in TADL. . . . . 20
3.4 Verification method in detail . . . . . . . . . . . . . . . . . . . . . . 21
3.5 UCBA constructed from use-cases $u_1, \ldots, u_n$ . . . . . . . . . . . . . 25
3.6 Transformation of TADL Formulas to NuSVM LTL/CTL specification. 30
3.7 A simplified template for NuSMV code used in the transformation from UCBA. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
3.8 Example: precedence relation in GPM specification. . . . . . . . . . . 37

3.9 FOAM Scalability Experiment 1 . . . . . . . . . . . . . . . . . . . . 38
3.10 FOAM Scalability Experiment 2 . . . . . . . . . . . . . . . . . . . . 38
3.11 FOAM Scalability Experiment 3 . . . . . . . . . . . . . . . . . . . . 39
3.12 FOAM Scalability Experiment 4 . . . . . . . . . . . . . . . . . . . . 39
3.13 Transformation pipeline of the FOAM tool. . . . . . . . . . . . . . . 44
3.14 Screenshot from FOAM HTML report – overview. . . . . . . . . . . 45
3.15 Screenshot from FOAM HTML report – single use-case. . . . . . . . 46
3.16 Screenshots from an HTML report generated by the FOAM tool. . . . 46

4.1 Stanford typed dependencies representation of a sentence. . . . . . . . 52
4.2 Hierarchy of Stanford Typed Dependencies . . . . . . . . . . . . . . 53
4.3 Generated Constituency Parse Tree . . . . . . . . . . . . . . . . . . 53
4.4 Sentence parsed using the Stanford CoreNLP framework . . . . . . . 54
4.5 Links between manually annotated text and domain model . . . . . . 56
4.6 Specification meta-model (input data for feature extraction). . . . . . 56
4.7 Usage Overview - main phases of our method. . . . . . . . . . . . . . 58
4.8 Preprocessing Phase . . . . . . . . . . . . . . . . . . . . . . . . . . 59
4.9 Feature-selection phase . . . . . . . . . . . . . . . . . . . . . . . . 60
4.10 Training phase . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
4.11 Elicitation phase . . . . . . . . . . . . . . . . . . . . . . . . . . . 62
4.12 Evaluation results in the Experiment . . . . . . . . . . . . . . . . . 67
4.13 Predicted domain model in the experiment . . . . . . . . . . . . . . 68
4.14 Predicted domain model (before editing) . . . . . . . . . . . . . . . 69

A.1 UC1: Login to the system . . . . . . . . . . . . . . . . . . . . . . . 96
A.2 MOD1_UC1: Register in the system . . . . . . . . . . . . . . . . . 97
A.3 MOD1_UC2: Provide personal and education information . . . . . . 97
A.4 MOD1_UC3: Choose a major . . . . . . . . . . . . . . . . . . . . . 98
A.5 MOD1_UC4: Assign an application fee to a major . . . . . . . . . . 98
A.6 MOD1_UC5: Check application status . . . . . . . . . . . . . . . . 99
A.7 MOD2_UC1: Create a new admission . . . . . . . . . . . . . . . . . 99
A.8 MOD2_UC8: Add a new user . . . . . . . . . . . . . . . . . . . . . 100
A.9 MOD2_UC12: Import admissions fees . . . . . . . . . . . . . . . . . 100

B.1 Results for the "relcl" classification . . . . . . . . . . . . . . . . . . 101
B.2 Results for the "linktype" classification . . . . . . . . . . . . . . . . 102
B.3 Results for the "roleInLink" classification . . . . . . . . . . . . . . . 103

C.1 Training domain model of the Library System . . . . . . . . . . . . 105

# Bibliography


[1] R. J. Abbott. “Program design by informal English descriptions”. In: *Commun. ACM* 26.11 (1983), pp. 882–894. ISSN: 0001-0782. DOI: 10.1145/182.358441.

[2] AECMA (The European Association of Aerospace Industries). *AECMA Simplified English*. Issue 1, Revision 1. A Guide for the Preparation of Aircraft Maintenance Documentation in the International Aerospace Maintenance Language. Jan. 1998.

[3] M. Aksit and K. v. d. Berg. “Use Cases in Object-Oriented Software Development”. In: *Language* (1999).

[4] L. A. E. Al-safadi. “Natural Language Processing for Conceptual Modeling”. In: *International Journal of Digital Content Technology and its Applications* 3.3 (2009), pp. 47–59. ISSN: 19759339. DOI: 10.4156/jdcta.vol3.issue3.6.

[5] B. Alchimowicz, J. Jurkiewicz, J. Nawrocki, and M. Ochodek. “Towards Use-Cases Benchmark”. In: *Proc. of CEE-SET’08*. Vol. 4980. LNCS. Springer, 2008, pp. 20–33.

[6] B. Alchimowicz, J. Jurkiewicz, M. Ochodek, and J. Nawrocki. “Building Benchmarks for Use Cases”. In: *Computing and Informatics* 29.1 (2010), pp. 27–44.

[7] L. de Alfaro and T. A. Henzinger. “Interface Automata”. In: *SIGSOFT Softw. Eng. Notes* 26.5 (2001), pp. 109–120. ISSN: 0163-5948. DOI: 10.1145/503209.503226.

[8] J. M. Almendros-Jimenez and L. Iribarne. “Describing Use-Case Relationships with Sequence Diagrams”. In: *TCJ* 50.1 (2006), pp. 116–128. ISSN: 0010-4620. DOI: 10.1093/comjnl/bxl053.

[9] J. M. Almendros-Jimenez and L. Iribarne. “Describing Use Cases with Activity Charts”. In: *Proc. of MIS’04*. Springer, 2004, pp. 141–159. DOI: 10.1007/11518358_12.

[10] V. Ambriola and V. Gervasi. “On the Systematic Analysis of Natural Language Requirements with CIRCE”. In: *Automated Software Engineering* (2006).

[11] K. v. d. Berg. “Control-Flow Semantics of Use Cases in UML”. In: *Science* (1999), pp. 1–18.

[12] T. Bures, P. Hnetynka, P. Kroha, and V. Simko. *Requirement Specifications Using Natural Languages*. Tech. rep. 2012/5. D3S, Charles University in Prague, 2012. URL: http://d3s.mff.cuni.cz/publications/download/D3S-TR-2012-05.pdf.

[13] V. Castaneda, L. Ballejos, M. L. Caliusco, and M. R. Galli. "The Use of Ontologies in Requirements Engineering". In: *Global Journal of Research Eng.* 10.6 (2010), pp. 2–8.

[14] E. Charniak. "A maximum-entropy-inspired parser". In: *Proc. of NAACL'00*. 2000, pp. 132–139.

[15] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. "NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking". In: *Proc. of CAV 2002, Copenhagen, Denmark*. Vol. 2404. LNCS. Springer, 2002, pp. 359–364. ISBN: 3-540-43997-8.

[16] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. Cambridge, MA: MIT Press, 1999. ISBN: 0262032708.

[17] A. Cockburn. *Writing Effective Use Cases*. Boston, MA, USA: Addison-Wesley, 2001, p. 270. ISBN: 0201702258, 9780201702255.

[18] M. J. Collins. "A new statistical parser based on bigram lexical dependencies". In: *Proc. of ACL'96*. Morristown, New York, USA: Association for Computational Linguistics, 1996, pp. 184–191. DOI: 10.3115/981863.981888.

[19] J. Corbett, M. Dwyer, J. Hatcliff, and Robby. "A Language Framework for Expressing Checkable Properties of Dynamic Software". In: *SPIN Model Checking and Software Verification*. Vol. 1885. LNCS. Springer, 2000, pp. 205–223. ISBN: 978-3-540-41030-0. DOI: 10.1007/10722468_13.

[20] A. Davis and A. Hickey. "A Quantitative Assessment of Requirements Engineering Publications – 1963-2008". In: *Requirements Engineering: Foundation for Software Quality*. Vol. 5512. LNCS. Springer, 2009, pp. 175–189. ISBN: 978-3-642-02049-0. DOI: 10.1007/978-3-642-02050-6_15.

[21] J. L. De Coi, N. E. Fuchs, K. Kaljurand, and T. Kuhn. "Controlled English for Reasoning on the Semantic Web". In: *Semantic Techniques for the Web — The REWERSE Perspective*. Ed. by F. Bry and J. Maluszynski. Vol. 5500. LNCS. Springer, 2009, pp. 276–308. ISBN: 978-3-642-04580-6.

[22] J. Dedek and P. Vojtas. "Computing Aggregations from Linguistic Web Resources: A Case Study in Czech Republic Sector/Traffic Accidents". In: *2008 The Second International Conference on Advanced Engineering Computing and Applications in Sciences*. IEEE, 2008, pp. 7–12. ISBN: 9780769533698. DOI: 10.1109/ADVCOMP.2008.17.

[23] D. K. Deeptimahanti and R. Sanyal. "Semi-automatic generation of UML models from natural language requirements". In: *Proc. of ISEC'11*. ACM Press, 2011, pp. 165–174. ISBN: 9781450305594. DOI: 10.1145/1953355.1953378.

[24] I. Diaz, O. Pastor, and A. Matteo. "Modeling Interactions using Role-Driven Patterns". In: *Proceedings of the 13th IEEE International Conference on Requirements Engineering*. RE '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 209–220. ISBN: 0-7695-2425-7. DOI: 10.1109/RE.2005.42.

[25] J. Drazan and V. Mencl. "Improved Processing of Textual Use Cases: Deriving Behavior Specifications". In: *Proc. of SOFSEM '07*. Harrachov, Czech Republic: Springer, 2007, pp. 856–868. ISBN: 978-3-540-69506-6.

[26] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. “Property specification patterns for finite-state verification”. In: *Proc. of FMSP’98*. New York, USA: ACM Press, 1998, pp. 7–15. ISBN: 0897919548. DOI: 10.1145/298595.298598.

[27] S. Emadi. “Mapping annotated sequence diagram to a Petri net notation for reliability evaluation”. In: *Proc. of ICETC’10*. Vol. 3. 2010, pp. 57–61. ISBN: 9781424463701. DOI: 10.1109/ICETC.2010.5529597.

[28] J. Enge, A. Glowacz, M. Grega, M. Leszczuk, Z. Papir, P. Romaniak, and V. Simko. “OASIS Archive – Open Archiving System with Internet Sharing”. In: *Future Multimedia Networking*. Vol. 5630. LNCS. Springer, 2009, pp. 254–259. ISBN: 978-3-642-02471-9. DOI: 10.1007/978-3-642-02472-6_28.

[29] D. Firesmith. *GPM SRS*. 2003. URL: http://www.it.uu.se/edu/course/homepage/pvt/SRS.pdf.

[30] J. Francu and P. Hnetynka. “Automated generation of implementation from textual system requirements”. In: *Software Engineering Techniques* (2011), pp. 34–47. DOI: 10.1007/978-3-642-22386-0_3.

[31] B. Gleich, O. Creighton, and L. Kof. “Ambiguity Detection: Towards a Tool Explaining Ambiguity Sources”. In: *Requirements Engineering: Foundation for Software Quality*. Ed. by R. Wieringa and A. Persson. Vol. 6182. LNCS. Springer, 2010, pp. 218–232. ISBN: 978-3-642-14191-1. DOI: 10.1007/978-3-642-14192-8_20.

[32] A. Glowacz, M. Grega, M. Leszczuk, Z. Papir, P. Romaniak, P. Fornalski, M. Lutwin, J. Enge, T. Lurk, and V. Simko. “Open internet gateways to archives of media art”. In: *Multimedia Tools and Applications* (Mar. 2011), pp. 1–24. ISSN: 1380-7501. DOI: 10.1007/s11042-011-0784-3.

[33] M. Hairul, N. Nasir, and S. Sahibuddin. “Critical success factors for software projects : A comparative study”. In: 6.10 (2011), pp. 2174–2186. DOI: 10.5897/SRE10.1171.

[34] H. M. Harmain and R. Gaizauskas. “Cm-builder: A natural language-based case tool for object-oriented analysis”. In: *Automated Software Engineering* (2003), pp. 157–181. DOI: 10.1023/A:1022916028950.

[35] M. A. Hearst. “Automatic acquisition of hyponyms from large text corpora”. In: *Proc. of COLING’92*. 1992. DOI: 10.3115/992133.992154.

[36] H. Herchi and W. B. Abdessalem. “From user requirements to UML class diagram”. In: *CoRR* abs/1211.0713 (2012).

[37] H. Heyan and Z. Xiaofei. “Part-of-speech tagger based on maximum entropy model”. In: *Proc. of ICCSIT’09*. 2009. DOI: 10.1109/ICCSIT.2009.5234787.

[38] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall Int. (UK) Ltd., 1985. ISBN: 0-13-153289-8.

[39] A. Hogenboom, F. Hogenboom, F. Frasincar, K. Schouten, and O. Meer. “Semantics-based information extraction for detecting economic events”. English. In: *Multimedia Tools and Applications* 64.1 (2013), pp. 27–52. ISSN: 1380-7501. DOI: 10.1007/s11042-012-1122-0.

[40] G. J. Holzmann. “The Model Checker SPIN”. In: *IEEE TSE* 23 (5 May 1997), pp. 279–295. ISSN: 0098-5589. DOI: 10.1109/32.588521.

[41] M. G. Ilieva and O. Ormandjieva. “Models Derived from Automatically Analyzed Textual User Requirements”. In: *Fourth International Conference on Software Engineering Research, Management and Applications (SERA’06)*. IEEE, 2006, pp. 13–21. ISBN: 0-7695-2656-X. DOI: 10.1109/SERA.2006.51. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1691356.

[42] E. Insfran, O. Pastor, and R. Wieringa. “Requirements engineering-based conceptual modelling”. In: *Requirements Engineering* (2002), pp. 61–72. URL: http://link.springer.com/article/10.1007/s007660200005.

[43] I. Jacobson. *Object-Oriented Software Engineering; A Use Case Driven Approach*. New York: Addison-Wesley, 1992. ISBN: 978-0201544350.

[44] M. de Jonge and T. C. Ruys. “The SpinJa model checker”. In: *Proceedings of SPIN 2010, Enschede, The Netherlands*. Vol. 6349. LNCS. Springer, 2010, pp. 124–128. ISBN: 978-3-642-16163-6. DOI: 10.1007/978-3-642-16164-3_9.

[45] C. Kamprath, E. Adolphson, T. Mitamura, and E. Nyberg. *Controlled Language for Multilingual Document Production: Experience with Caterpillar Technical English*. 1998.

[46] L. Kof. “From Requirements Documents to System Models: A Tool for Interactive Semi-Automatic Translation”. In: *Proc. of RE’10*. 2010, pp. 391–392. DOI: 10.1109/RE.2010.53.

[47] L. Kof. “From Textual Scenarios to Message Sequence Charts: Inclusion of Condition Generation and Actor Extraction”. In: Ieee, 2008, pp. 331–332. ISBN: 978-0-7695-3309-4. DOI: 10.1109/RE.2008.12.

[48] L. Kof. “Scenarios: Identifying Missing Objects and Actions by Means of Computational Linguistics”. In: *15th IEEE International Requirements Engineering Conference (RE 2007)* (2007), pp. 121–130. DOI: 10.1109/RE.2007.38.

[49] L. Kof. “Treatment of Passive Voice and Conjunctions in Use Case Documents”. In: *Natural Language Processing and Information Systems*. Ed. by Z. Kedad, N. Lammari, E. Métais, F. Meziane, and Y. Rezgui. Vol. 4592. LNCS. Springer Berlin Heidelberg, 2007, pp. 181–192. ISBN: 978-3-540-73350-8. DOI: 10.1007/978-3-540-73351-5_16.

[50] J. Kofron, T. Poch, and O. Sery. “TBP: Code-Oriented Component Behavior Specification”. In: *Proc. of SEW’08*. Washington, DC, USA: IEEE CS, 2008, pp. 75–83. ISBN: 978-0-7695-3617-0. DOI: 10.1109/SEW.2008.14.

[51] S. Konrad and B. H. C. Cheng. “Automated Analysis of Natural Language Properties for UML Models”. In: *Satellite Events at the MoDELS 2005 Conference*. Vol. 3844. LNCS. Springer, 2006, pp. 48–57. ISBN: 978-3-540-31780-7. DOI: 10.1007/11663430_6.

[52] P. Kroha, R. Janetzko, and J. E. Labra. "Ontologies in Checking for Inconsistency of Requirements Specification". In: *2009 Third International Conference on Advances in Semantic Processing*. IEEE, Oct. 2009, pp. 32–37. ISBN: 978-1-4244-5044-2. DOI: 10.1109/SEMAPRO.2009.11.

[53] P. Kroha and M. Rink. "Text Generation for Requirements Validation". In: *Enterprise Information Systems*. Vol. 24. LNBIP. Springer, 2009, pp. 467–478. ISBN: 978-3-642-01346-1. DOI: 10.1007/978-3-642-01347-8_39.

[54] P. Kruchten. *The Rational Unified Process: An Introduction*. The Addison-Wesley Object Technology Series. Addison Wesley Professional, 2004. ISBN: 201707101.

[55] D. Kulak and E. Guiney. *Use cases: requirements in context*. ADDISON WESLEY Publishing Company Incorporated, 2004. ISBN: 9780321154989. URL: http://books.google.cz/books?id=qOjxBo7gTLwC.

[56] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". In: *Proc. of ICML'01*. 2001, pp. 282–289. ISBN: 1-55860-778-1.

[57] C. Lai and S. Bird. "LPath+ : A First-Order Complete Language for Linguistic Tree Query". In: *Proc. of PACLIC'05*. Academia Sinica, 2005, pp. 1–12.

[58] C. Larman. *Applying UML and patterns: An introduction to object-oriented analysis and design and iterative development*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004. ISBN: 0131489062.

[59] H. Lee, A. Chang, Y. Peirsman, N. Chambers, M. Surdeanu, and D. Jurafsky. "Deterministic coreference resolution based on entity-centric, precision-ranked rules". In: *Computational Linguistics* (2012), pp. 1–54. ISSN: 0891-2017. DOI: 10.1162/COLI_a_00152.

[60] H. Lee, Y. Peirsman, A. Chang, N. Chambers, M. Surdeanu, and D. Jurafsky. "Stanford's multi-pass sieve coreference resolution system at the CoNLL-2011 shared task". In: *Proc. of CoNLL'11*. Stroudsburg, PA, USA: Association for Computational Linguistics, Sept. 2011, pp. 28–34. ISBN: 9781937284084.

[61] R. Levy and G. Andrew. "Tregex and Tsurgeon: tools for querying and manipulating tree data structures". In: *Proc. of LREC'06*. 2006.

[62] D. Liu. "Automating transition from use-cases to class model". MA thesis. UNIVERSITY OF CALGARY, 2003. URL: http://homepage.usask.ca/~dol142/Files/ThesisDLiu.pdf.

[63] A. Ludtke, B. Gottfried, O. Herzog, G. Ioannidis, M. Leszczuk, and V. Simko. "Accessing Libraries of Media Art through Metadata". In: *2009 20th International Workshop on Database and Expert Systems Application*. IEEE, 2009, pp. 269–273. ISBN: 978-0-7695-3763-4. DOI: 10.1109/DEXA.2009.93.

[64] M. Luisa, F. Mariangela, and N. I. Pierluigi. "Market research for requirements analysis using linguistic tools". In: *Requirements Engineering* 9.1 (Feb. 2004), pp. 40–56. ISSN: 0947-3602. DOI: 10.1007/s00766-003-0179-8.

[65] S. G. MacDonell, K. Min, and A. M. Connor. “Autonomous Requirements Specification Processing Using Natural Language Processing”. In: *Proceedings of the ISCA 14th International Conference on Intelligent and Adaptive Systems and Software Engineering (IASSE-05)*. Toronto, Canada: ISCA, 2005, pp. 266–270.

[66] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. “Specifying Distributed Software Architectures”. In: *Proc. of ESEC’95*. 1995. URL: http://pubs.doc.ic.ac.uk/SpecifyDistributedArchitectures/.

[67] C. Manning. “Part-of-Speech Tagging from 97Linguistics?” In: *Computational Linguistics and Intelligent Text Processing*. Vol. 6608. LNCS. Springer, 2011, pp. 171–189. ISBN: 978-3-642-19399-6. DOI: 10.1007/978-3-642-19400-9_14.

[68] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008. ISBN: 0521865719, 9780521865715.

[69] C. D. Manning and H. Schutze. *Foundations of statistical natural language processing*. MIT Press, 1999. ISBN: 0-262-13360-1.

[70] M. Marcus and M. Marcinkiewicz. “Building a large annotated corpus of English: The Penn Treebank”. In: *Computational linguistics* (1993). URL: http://dl.acm.org/citation.cfm?id=972475.

[71] M.-c. D. Marneffe and C. D. Manning. “Stanford typed dependencies manual”. In: 09 (2011), pp. 1–24. URL: http://nlp.stanford.edu/software/dependencies_manual.pdf.

[72] M.-c. D. Marneffe and C. D. Manning. “The Stanford typed dependencies representation”. In: *Proc. of the Workshop on Cross-Framework and Cross-Domain Parser Evaluation*. 08. Stroudsburg: Association for Computational Linguistics, 2008, pp. 1–8. URL: http://dl.acm.org/citation.cfm?id=1608858.1608859.

[73] V. Mencl. “Deriving Behavior Specifications from Textual Use Cases”. In: *Proc. of WITSE’04*. Linz, Austria, 2004.

[74] V. Mencl, J. Francu, J. Ondrusek, M. Fiedler, and A. Plsek. *Procasor Environment: Interactive Environment for Requirement Specification*. 2005. URL: http://dsrg.mff.cuni.cz/~mencl/procasor-env/.

[75] R. Milner. *Communication and Concurrency*. Hertfordshire, UK, UK: Prentice Hall International (UK) Ltd., 1995. ISBN: 0-13-115007-3.

[76] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1988.

[77] J. Mirovsky. “Netgraph - Making Searching in Treebanks Easy”. In: *Proc. of IJCNLP’08*. 2008, pp. 945–950.

[78] C. Neill and P. Laplante. “Requirements engineering: The state of the practice”. In: *IEEE Software* 20.6 (Nov. 2003), pp. 40–45. ISSN: 0740-7459. DOI: 10.1109/MS.2003.1241365.

[79] K. Nigam, J. Lafferty, and A McCallum. "Using maximum entropy for text classification". In: *IJCAI-99 Workshop on Machine Learning for Information Filtering*. Stockholm, Sweden, 1999, pp. 61–67. URL: http://www.cc.gatech.edu/~isbell/reading/papers/maxenttext.pdf.

[80] OMG. *UML for Modeling QoS and Fault Tolerance Characteristics and Mechanisms*. OMG document formal/2008-04-05. 2008.

[81] F. Plasil and S. Visnovsky. "Behavior Protocols for Software Components". In: *IEEE TSE* 28.11 (2002), pp. 1056–1076. ISSN: 0098-5589.

[82] F. Plasil and V. Mencl. "Getting 'Whole Picture' Behavior In A Use Case Model". In: *Journ. of Integrated Design and Process Sci.* 7.4 (2003), pp. 63–79. ISSN: 1092-0617.

[83] J. A. Pow-Sang, A. Nakasone, R. Imbert, and A. M. Moreno. "An Approach to Determine Software Requirement Construction Sequences Based on Use Cases". In: *Proc. of ASEA'08*. Washington, DC, USA: IEEE, 2008, pp. 17–22. ISBN: 978-0-7695-3432-9. DOI: 10.1109/ASEA.2008.33.

[84] J. L. Rash, M. G. Hinchey, C. A. Rouff, D. Gracanin, and J. Erickson. "A requirements-based programming approach to developing a NASA autonomous ground control system". In: *Artificial Intelligence Review* 25.4 (2007), pp. 285–297. ISSN: 0269-2821. DOI: 10.1007/s10462-007-9029-2.

[85] D. Rosenberg and M. Stephens. *Use Case Driven Object Modeling with UML: Theory and Practice*. Springer, 2007, p. 471. ISBN: 9781590597743.

[86] N. Samarasinghe and S. Some. "Generating a domain model from a use case model". In: 2005, pp. 5–10. URL: http://www.csi.uottawa.ca/~ssome/UCEdWeb/publis/domainGen.pdf.

[87] M. Seresh S. and O. Ormandjieva. "Automated Assistance for Use Cases Elicitation from User Requirements Text". In: *Proc. of WER'08*. 16. Barcelona, Spain, 2008, pp. 128–139.

[88] Y. Shinkawa. "Model Checking for UML Use Cases". In: *SERA*. Vol. 150. Studies in Computational Intelligence. Springer, 2008. ISBN: 978-3-540-70774-5. DOI: 10.1007/978-3-540-70561-1_17.

[89] V. Simko, J. Vinarek, O. Fiala, J. Krajicek, R. Tomori, P. Hnetynka, T. Bures, and D. Hauzar. *Requirements Processing Tool*. Project hosted at Google Code, http://code.google.com/a/eclipselabs.org/p/reprotool/. 2011.

[90] V. Simko. *Patterns In Specification Documents*. Tech. rep. 2011/6. D3S, Charles University in Prague, 2011. URL: http://d3s.mff.cuni.cz/publications/download/D3S-TR-2011-06.pdf.

[91] V. Simko, D. Hauzar, T. Bures, P. Hnetynka, and F. Plasil. "Verifying Temporal Properties of Use-Cases in Natural Language". In: *Postproc. of FACS'2011*. LNCS. Springer, 2011. DOI: 10.1007/978-3-642-35743-5_21.

[92] V. Simko, P. Hnetynka, and T. Bures. "From Textual Use-Cases to Component-Based Applications". In: *Proc. of SNPD'10*. Vol. 295. Studies in Computational Intelligence. Springer, 2010, pp. 23–37. ISBN: 978-3-642-13264-3. DOI: 10.1007/978-3-642-13265-0.

[93] V. Simko, P. Hnetynka, T. Bures, and F. Plasil. “FOAM: A Lightweight Method for Verification of Use-Cases”. In: *Software Engineering and Advanced Applications (SEAA), 38th EUROMICRO Conference on*. 2012, pp. 228–232. DOI: 10.1109/SEAA.2012.15.

[94] V. Simko, P. Hnetynka, T. Bures, and F. Plasil. *Formal Verification of Annotated Use-Cases (FOAM Method)*. Tech. rep. 2012/2. D3S, Charles University in Prague, 2012. URL: http://d3s.mff.cuni.cz/publications/download/D3S-TR-2012-02.pdf.

[95] V. Simko, P. Kroha, and P. Hnetynka. *Domain Model Generation With the Help of Supervised Machine Learning*. Tech. rep. 2012/6. D3S, Charles University in Prague, 2012. URL: http://d3s.mff.cuni.cz/publications/download/D3S-TR-2012-06.pdf.

[96] V. Simko, P. Kroha, and P. Hnetynka. *Implemented Domain Model Generation*. Tech. rep. 2013/3. D3S, Charles University in Prague, Apr. 2013. URL: http://d3s.mff.cuni.cz/publications/download/D3S-TR-2013-03.pdf.

[97] D. Sinnig, P. Chalin, and F. Khendek. “Consistency between Task Models and Use Cases”. In: *EIS’08*. Vol. 4940. LNCS. Springer, 2008, pp. 71–88. ISBN: 978-3-540-92697-9.

[98] D. Sinnig, P. Chalin, and F. Khendek. “LTS semantics for use case models”. In: *Proc. of SAC’09, Honolulu, Hawaii*. ACM, 2009, pp. 365–370. ISBN: 978-1-60558-166-8. DOI: 10.1145/1529282.1529362.

[99] D. Sinnig, F. Khendek, and P. Chalin. “Partial order semantics for use case and task models”. In: *FAC* 23.3 (2010), pp. 307–332. ISSN: 0934-5043. DOI: 10.1007/s00165-010-0158-z.

[100] D Sinning, P Chalin, and F Khendek. “Towards a Common Semantic Foundation for Use Cases and Task Models”. In: *ENTCS* 183 (2007), pp. 73–88. ISSN: 15710661. DOI: 10.1016/j.entcs.2007.01.062.

[101] M. Smialek, J. Bojarski, W. Nowakowski, A. Ambroziewicz, and T. Straszak. “Complementary Use Case Scenario Representations Based on Domain Vocabularies”. In: *Model Driven Engineering Languages and Systems*. Ed. by G. Engels, B. Opdyke, D. Schmidt, and F. Weil. Vol. 4735. LNCS. Springer, 2007, pp. 544–558. ISBN: 978-3-540-75208-0. DOI: 10.1007/978-3-540-75209-7_37.

[102] S. S. Somé. “Formalization of Textual Use Cases Based on Petri Nets”. In: *International Journal of Software Engineering and Knowledge Engineering* 20.05 (2010), p. 695. ISSN: 0218-1940. DOI: 10.1142/S0218194010004931.

[103] S. S. Somé. “Supporting use case based requirements engineering”. In: *Information and Software Technology* 48.1 (2006), pp. 43–58. ISSN: 09505849. DOI: 10.1016/j.infsof.2005.02.006.

[104] J. Stepanek and P. Pajas. “Querying diverse treebanks in a uniform way”. In: *Proc. of LREC’10*. 2010.

[105] T. Symul, S. M. Assad, and P. K. Lam. “Real time demonstration of high bitrate quantum random number generation with coherent laser light”. In: *Applied Physics Letters* 98.23, 231103 (2011), p. 231103. DOI: 10.1063/1.3597793.

[106] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer. “Feature-rich part-of-speech tagging with a cyclic dependency network”. In: *Proc. of NAACL’03*. 2003. DOI: 10.3115/1073445.1073478.

[107] K. Verma and A. Kass. “Requirements analysis tool: A tool for automatically analyzing software requirements documents”. In: *The Semantic Web-ISWC 2008* (2008), pp. 751–763. DOI: 10.1007/978-3-540-88564-1_48.

[108] C. Videira, D. Ferreira, and A. Rodrigues da Silva. “A Linguistic Patterns Approach for Requirements Specification”. In: *Proc. of EUROMICRO’06*. Washington, DC, USA: IEEE, 2006, pp. 302–309. ISBN: 0-7695-2594-6. DOI: 10.1109/EUROMICRO.2006.8.

[109] K. Wenzel. *A language for searching tree-like structures*. Project hosted at Tigris.org, http://tpl.tigris.org. 2007.

[110] Y. Yang. “An Evaluation of Statistical Approaches to Text Categorization”. In: *Information Retrieval* 1.1-2 (1999), pp. 69–90. ISSN: 1386-4564. DOI: 10.1023/A:1009982220290.

[111] T. Yue and L. Briand. “An automated approach to transform use cases into activity diagrams”. In: *Modelling Foundations and Applications* (2010), pp. 337–353.

[112] T. Yue, S. Ali, and L. Briand. “Automated Transition from Use Cases to UML State Machines to Support State-Based Testing”. In: *MFA*. Vol. 6698. LNCS. Springer, 2011, pp. 115–131. ISBN: 978-3-642-21469-1.

[113] T. Yue and L. Briand. “A systematic review of transformation approaches between user requirements and analysis models”. In: *Requirements Engineering* (2011), pp. 1–41.

[114] T. Yue, L. C. Briand, and Y. Labiche. “Facilitating the Transition from Use Case Models to Analysis Models:Approach and Experiments”. In: *Transactions On Software Engineering And Methodology* (2011).

[115] Y. Zeng. “Recursive object model (ROM)-Modelling of linguistic information in engineering design”. In: *Comput. Ind.* 59.6 (2008), pp. 612–625. ISSN: 0166-3615. DOI: 10.1016/j.compind.2008.03.002.

# Appendix A

# FOAM case study

**MOD2_UC6: Transfer Candidates' data**
Summary: When the admission is finished, data of qualified candidates should be transferred to the Students management system.

Preconditions:
– Administrator is logged in to the system. → **Preceding**: "UC???..."

Trigger
– Admission end date has passed and Selection committee has qualified Candidates. → **Preceding**: "UC???..."

Main Scenario:
1. Administrator chooses all qualified Candidates who provided paper version of their application form.
2. Administrator chooses the transfer option. **#(create:transferOption)**
3. System transfers Candidates data to the Students management system. **#(open:transfer)**

**Extension**: System is unable to transfer some of the accounts.
– System informs that some accounts have not been transferred.
– Administrator selects detailed information about the error.
– System presents details concerning errors during transfer process.
– Use case finishes. **#(close:transfer)**, **#(abort)**

**Variation**: Is a recoverable error.
– System tries the transfer operation again in step 3. **#(goto:3)**

4. System authorizes in the Students management system. **#(include:UC???...)**
5. System sends Candidates data. **#(use:transferOption)**
6. Students management system verifies the data.
7. Students management system sends an acknowledge message to the System.
8. System displays information about a successful transfer. **#(close:transfer)**

---

"Flow Annotations"
**#(abort)**
– This annotation expresses abort of the scenario. It can only be added to the last step of a variation or an extension.
**#(goto:s)**
– This annotation represents a jump within the use–case.
– The parameter $s$ indicates the target use–case step of the jump.
**#(include:u)**
– This annotation specifies inclusion (inlining) of another use–case $u$.

---

"Temporal Annotations"
**#(open:x)** ... **#(close:x)** – something like a transaction:
– there open–close has to be properly paired
– when $x$ is opened, it has to be closed
– $x$ cannot be closed before being opened first
**#(create:x)** ... **#(use:x)** – useful for marking data dependency:
– $x$ cannot be used before being created
– $x$ can be used multiple times
– if $x$ is created, there has to be at least one branch where $x$ is used (some branches can well be without it)

## A.1 FOAM case study : Answers

**UC1: Login to the system**

Summary: In order to use system one has to authenticate.

Preconditions:
– User is not logged in.

Main Scenario:
1. User opens main page.
2. System presents main page with a login form.
3. User fills the login form with the authentication data.
4. System verifies the given data.
5. System welcomes Candidate.

**Extension**: 4a. Not all obligatory data was given.
4a1. System points which data is missing.
4a2. Go back to step 3. **#(goto:3)**

**Extension**: 4b. No account with the certain login exists in the system.
4b1. System informs the User that there is no account with the given user name in the system.
4b2. System suggests the User to register in the system as a Candidate or contact Administrator to create new account.
4b3. Go back to step 3. **#(goto:3)**

**Extension**: 4b2a. User decided to register as a Candidate.
4b2a1. Include MOD1_UC1. **#(include:MOD1_UC1)**

**Extension**: 4b2b. User decided to contact admin.
4b2b1. Include MOD2_UC8. **#(include:MOD2_UC8)**

**Figure A.1:** UC1: Login to the system

**MOD1_UC1: Register in the system**

Summary: Candidate has to register in the system in order to apply for studies.

Preconditions:
– Candidate is not logged in

Main Scenario:
1. Candidate opens system main–page.
2. Candidate chooses registration option.
3. System presents a registration data form and asks to enter the registration data.
4. Candidate fills the registration data form and submits the registration data form. **#(open:registration)**
5. System verifies if data is correct.
6. System informs that account has been created. **#(close:registration)**

**Extension**: 5a. Some obligatory fields were not filled.
5a1. Systems highlights the missing fields. **#(close:registration)** · · · · · · · · · **FIXED**
5a2. Back to step 4. **#(goto:4)**

**Extension**: 5b. Account with the given user name already exist.
5b1. System informs that the user name is in use. **#(close:registration)** · · · · · · · · · **FIXED**
5b2. Back to step 4. **#(goto:4)**

**Extension**: 5c. Given passwords don't match.
5c1. System informs Candidate that passwords don't match. **#(close:registration)** · · · · · · · · · **FIXED**
5c2. Back to step 4. **#(goto:4)**

**Figure A.2:** MOD1_UC1: Register in the system

**MOD1_UC2: Provide personal and education information**

Summary: Candidate has to provide personal information as well as some facts concerning his/her previous education.

Preconditions:
– Candidate is logged in to the system.
**Preceding**: "UC1: Login to the System"

Triggers:
– Either Candidate has logged to the system for the first time or has chosen to enter his/her application data.

Main Scenario:
1. Candidate provides personal information.
2. Candidate chooses to provide information concerning former education.
3. System presents the education data form.
4. Candidate fills the education data form and confirms.
5. System stores the data.
6. System displays a confirmation message.

**Extension**: 4a Some obligatory data was not provided.
4a1 System informs that required some data was not provided and highlights the missing fields.
4a2 Go back to step 2. **#(goto:2)**

**Variation**: 4a2a Candidate logged to the system for the first time
4a2a1 Go back to step 1. **#(goto:1)**

**Extension**: 4b. Some data was provided in wrong format.
4b1. System informs that some data was not provided correctly and highlights the fields that were consider as wrongly formatted.
4b2. Go back to step 2. **#(goto:2)**

**Variation**: 4b2a. Candidate logged to the system for the first time
4b2a1. Go back to step 1. **#(goto:1)**

**Figure A.3:** MOD1_UC2: Provide personal and education information

**MOD1_UC3: Choose a major**

Primary: FALSE · · · · · · · · · **FIXED**

Summary: Candidate would like to choose one or more majors he/she would like to apply for.

Preconditions:
– Candidate is logged in to the system
**Preceding**: "UC1: Login to the System"
– Candidate provided personal and education information
**Preceding**: "MOD1_UC2: Provide personal and education information"

Main Scenario:
1. Candidate chooses the adding–new–major option.
2. System presents a list of majors for which admission is available.
3. Candidate chooses a major. **#(create:chosenMajor)**
4. System presents a list of majors chosen by Candidate.

**Extension**: 3a. Candidate would like to apply for more majors.
3a1. Candidate chooses many majors.
3a2. Continue with step 4. **#(goto:4)**

**Figure A.4:** MOD1_UC3: Choose a major

**MOD1_UC4: Assign an application fee to a major**

Summary: Candidate has to pay an application fee for each major he/she chooses.

Preconditions:
– Candidate is logged in to the system
**Preceding**: "UC1. Login to the System"
– Candidate has chosen at least one major
**Preceding**: "MOD1_UC3: Choose a major"

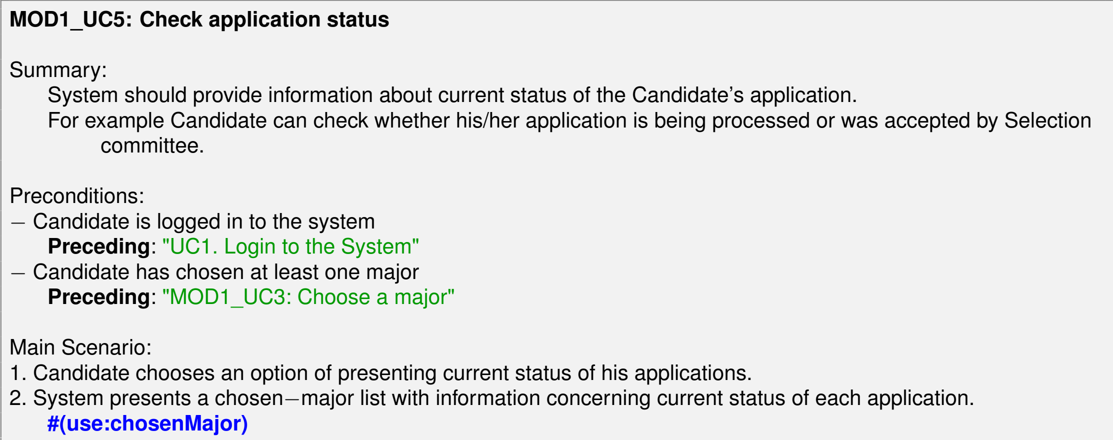
Main Scenario:
1. Candidate proceeds to the chosen–majors view.
2. System presents list containing chosen majors.
3. Candidate chooses a major that he/she wants to pay for. **#(use:chosenMajor)**
4. System presents a payment form and asks about the method of payment. **#(open:payment)**
5. Candidate chooses to use a credit card.
6. Candidate provides credit card data and confirms payment. **#(close:payment)**
7. System presents updated list of the chosen majors.

**Variation**: 5a. Candidate chooses to pay by money transfer.
5a1. System presents Candidate's individual account number.
5a2. Candidate performs money transfer ( outside the system ).
5a3. Money is registered by the System (MOD2 UC12) **#(include:MOD2_UC12)**
5a4. After money is registered candidate assigns the payment to a major. **#(use:registeredMoney)**
5a5. Use cases finishes. **#(abort)**, **#(close:payment)** · · · · · · · · · **FIXED**

**Extension**: 5a3a. Error occurred while registering. **#(guard:!create:registeredMoney)** · · · · · · · · · **FIXED**
5a3a1. Transaction terminated. **#(abort)**, **#(close:payment)** · · · · · · · · · **FIXED**

**Variation**: 5a4a. If he/she don't do that
5a4a1. the payment will be assigned automatically according to priorities.

**Figure A.5:** MOD1_UC4: Assign an application fee to a major



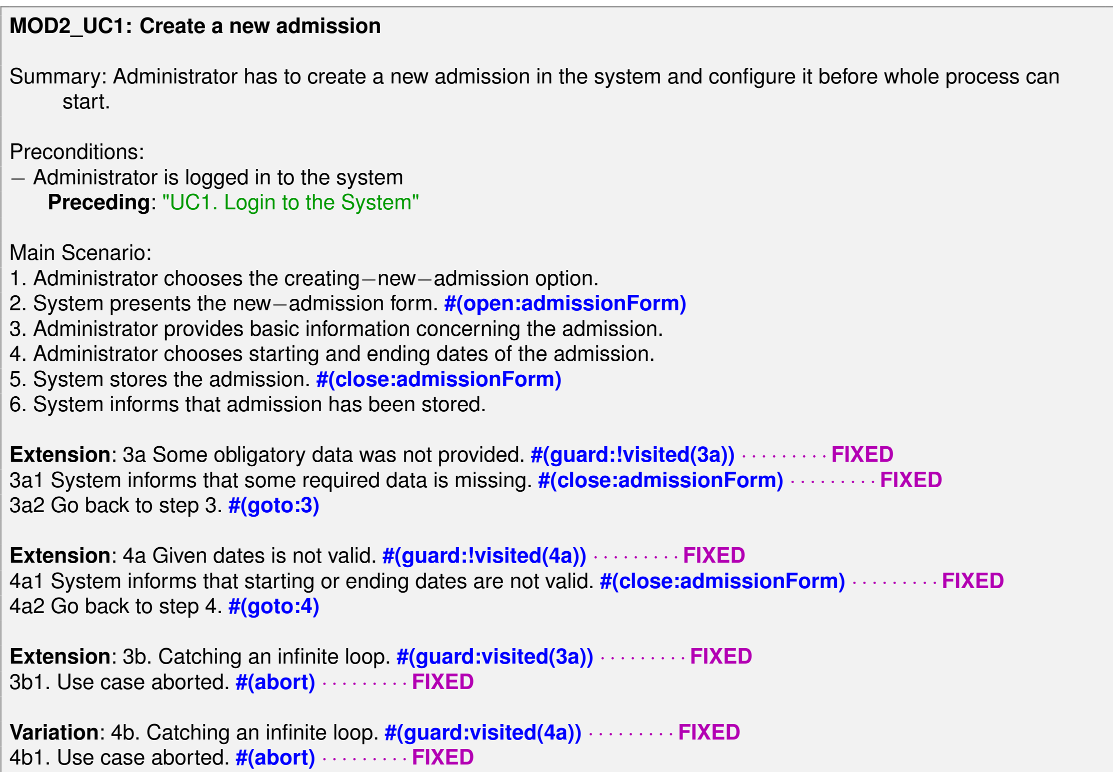
**MOD1_UC5: Check application status**

Summary:
System should provide information about current status of the Candidate's application.
For example Candidate can check whether his/her application is being processed or was accepted by Selection committee.

Preconditions:
– Candidate is logged in to the system
**Preceding**: "UC1. Login to the System"
– Candidate has chosen at least one major
**Preceding**: "MOD1_UC3: Choose a major"

Main Scenario:
1. Candidate chooses an option of presenting current status of his applications.
2. System presents a chosen–major list with information concerning current status of each application.
**#(use:chosenMajor)**

**Figure A.6:** MOD1_UC5: Check application status



**MOD2_UC1: Create a new admission**

Summary: Administrator has to create a new admission in the system and configure it before whole process can start.

Preconditions:
– Administrator is logged in to the system
**Preceding**: "UC1. Login to the System"

Main Scenario:
1. Administrator chooses the creating–new–admission option.
2. System presents the new–admission form. **#(open:admissionForm)**
3. Administrator provides basic information concerning the admission.
4. Administrator chooses starting and ending dates of the admission.
5. System stores the admission. **#(close:admissionForm)**
6. System informs that admission has been stored.

**Extension**: 3a Some obligatory data was not provided. **#(guard:!visited(3a))** · · · · · · · · · **FIXED**
3a1 System informs that some required data is missing. **#(close:admissionForm)** · · · · · · · · · **FIXED**
3a2 Go back to step 3. **#(goto:3)**

**Extension**: 4a Given dates is not valid. **#(guard:!visited(4a))** · · · · · · · · · **FIXED**
4a1 System informs that starting or ending dates are not valid. **#(close:admissionForm)** · · · · · · · · · **FIXED**
4a2 Go back to step 4. **#(goto:4)**

**Extension**: 3b. Catching an infinite loop. **#(guard:visited(3a))** · · · · · · · · · **FIXED**
3b1. Use case aborted. **#(abort)** · · · · · · · · · **FIXED**

**Variation**: 4b. Catching an infinite loop. **#(guard:visited(4a))** · · · · · · · · · **FIXED**
4b1. Use case aborted. **#(abort)** · · · · · · · · · **FIXED**

**Figure A.7:** MOD2_UC1: Create a new admission

**MOD2_UC8: Add a new user**

Summary: Administrator can add users.

Preconditions:
– Administrator is logged in to the system.
**Preceding**: "UC1. Login to the System"

Main Scenario:
1. Administrator chooses an option to add user.
2. System presents the new–user form.
3. Administrator fills the form. **#(open:newUserForm)**
4. Administrator grants roles to the user in the system.
5. System stores the user data. **#(close:newUserForm)**
6. System grants the user roles.
7. System displays confirmation message.

**Extension**: 3a. Not all obligatory data was given.
3a1. System informs that some data is missing. **#(close:newUserForm)** · · · · · · · · · **FIXED**
3a2. System highlights the missing fields.
3a3. Go back to step 3. **#(goto:3)**

**Figure A.8:** MOD2_UC8: Add a new user

**MOD2_UC12: Import admissions fees**

Primary: FALSE · · · · · · · · · **FIXED**

Summary: Information about payments which are done via money transfer procedure ( outside the system ) have to be imported from the Bank system.

Preconditions:
– Administrator is logged in to the system.
**Preceding**: "UC1. Login to the System"

Main Scenario:
1. Administrator chooses an option to import payments from the bank system.
2. System imports payment entries from the bank.
3. System displays a list containing information about all imported admission fees.
**#(create:registeredMoney)**

**Extension**: 2a. Error occurred during the import.
2a1. System displays error message with the detailed information concerning the source of the failure.
2a2. Use case finishes. **#(abort)**

**Figure A.9:** MOD2_UC12: Import admissions fees

# Appendix B

# Measured prediction performance

| FsetID | FallOut | F1 | Precision | Recall | SP | MCC | Context | #ctx |
|---|---|---|---|---|---|---|---|---|
| set51 | 34.7% | 22.4% | 14.6% | 55.0% | 65.3% | 1.02E-03 | rel_depOnRoot:src rel_ivf:pos | 2 |
| set3 | 2.9% | 20.7% | 30.0% | 19.2% | 97.1% | 4.56E-03 | rel_depOnRoot:src rel_depOnRoot:dest rel_ivf:lemma rel_ivf:pos rel_passroot | 5 |
| set15 | 2.8% | 16.2% | 25.3% | 15.3% | 97.2% | 2.80E-03 | rel_depOnRoot:src rel_depOnRoot:dest rel_ivf:lemma rel_passroot | 4 |
| set49 | 32.8% | 15.7% | 10.1% | 42.3% | 67.2% | 3.88E-04 | rel_depOnRoot:src rel_ivf:lemma | 2 |
| set2 | 2.7% | 15.2% | 40.0% | 9.5% | 97.3% | 2.59E-03 | rel_depOnRoot:src rel_depOnRoot:dest rel_semrootlemma rel_ivf:lemma rel_passroot | 5 |
| set20 | 1.8% | 14.5% | 35.0% | 9.2% | 98.2% | 2.49E-03 | rel_depOnRoot:src rel_depOnRoot:dest rel_semrootlemma rel_ivf:lemma | 4 |
| set46 | 26.4% | 14.4% | 10.8% | 27.7% | 73.6% | 1.81E-05 | rel_depOnRoot:src rel_depOnRoot:dest | 2 |
| set36 | 1.0% | 14.2% | 35.0% | 9.3% | 99.0% | 2.41E-03 | rel_depOnRoot:src rel_depOnRoot:dest rel_ivf:pos | 3 |
| set28 | 1.8% | 11.7% | 23.3% | 8.3% | 98.2% | 1.19E-03 | rel_depOnRoot:src rel_depOnRoot:dest rel_ivf:lemma | 3 |
| set14 | 28.8% | 11.6% | 8.2% | 20.8% | 71.2% | -2.20E-04 | rel_depOnRoot:src rel_depOnRoot:dest rel_ivf:lemma rel_ivf:pos | 4 |
| set1 | 2.8% | 9.8% | 17.5% | 13.7% | 97.2% | 1.59E-03 | rel_depOnRoot:src rel_depOnRoot:dest rel_semrootlemma rel_ivf:lemma rel_ivf:pos | 5 |
| set13 | 1.8% | 7.3% | 13.3% | 5.8% | 98.2% | 7.96E-04 | rel_depOnRoot:src rel_depOnRoot:dest rel_semrootlemma rel_ivf:pos | 4 |
| set29 | 1.1% | 6.7% | 10.0% | 5.0% | 98.9% | 1.32E-03 | rel_depOnRoot:src rel_depOnRoot:dest rel_passroot | 3 |
| set31 | 2.5% | 5.0% | 10.0% | 3.3% | 97.5% | 5.08E-04 | rel_depOnRoot:src rel_depOnRoot:dest rel_semrootlemma | 3 |
| set19 | 0.7% | 3.3% | 10.0% | 2.0% | 99.3% | 4.28E-04 | rel_depOnRoot:src rel_depOnRoot:dest rel_semrootlemma rel_passroot | 4 |
| set25 | 1.8% | 2.5% | 5.0% | 1.7% | 98.2% | -1.27E-04 | rel_depOnRoot:dest rel_semrootlemma rel_ivf:lemma | 3 |
| set53 | 1.5% | 2.2% | 3.3% | 1.7% | 98.5% | -1.35E-04 | rel_depOnRoot:dest rel_ivf:lemma | 2 |
| set56 | 7.0% | 1.6% | 1.0% | 5.0% | 93.0% | -9.70E-05 | rel_ivf:lemma rel_ivf:pos | 2 |

**Figure B.1:** Measured results for the "relcl" classification. The table contains top 50 results sorted by $F1$.

| FsetID | FallOut | F1 | Precision | Recall | SP | MCC | Context | #ctx |
|---|---|---|---|---|---|---|---|---|
| set12 | 10.3% | 76.7% | 73.0% | 80.9% | 89.7% | 3.95E-04 | pos pos:1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:2 wsuffix:3 wsuffix:4 sprel sppos | 19 |
| set19 | 11.0% | 76.4% | 71.1% | 83.1% | 89.0% | 4.00E-04 | pos pos:1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel sppos | 19 |
| set30 | 11.7% | 75.9% | 70.1% | 83.4% | 88.3% | 3.85E-04 | pos pos:1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:6 whascl whasdigit wprefix:1 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel sppos | 18 |
| set48 | 10.8% | 75.6% | 71.6% | 81.0% | 89.2% | 3.91E-04 | pos pos:1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:6 whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:2 wsuffix:3 wsuffix:4 sprel | 16 |
| set74 | 10.8% | 75.2% | 71.2% | 80.2% | 89.2% | 3.81E-04 | pos:1 wminlen:2 wminlen:3 wminlen:4 wminlen:5 wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:2 wsuffix:3 wsuffix:4 sprel | 13 |
| set83 | 10.4% | 75.1% | 71.7% | 79.1% | 89.6% | 3.81E-04 | pos:-1 lemma wminlen:4 wminlen:5 whascl whasdigit wprefix:1 wprefix:2 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:4 | 12 |
| set50 | 12.4% | 75.1% | 69.2% | 83.0% | 87.6% | 3.87E-04 | pos pos:1 pos:-1 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl wprefix:1 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:4 sprel sppos | 16 |
| set69 | 12.1% | 75.0% | 68.8% | 82.6% | 87.9% | 3.80E-04 | pos pos:-1 wminlen:2 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wsuffix:2 wsuffix:3 wsuffix:4 sprel | 14 |
| set23 | 11.9% | 74.9% | 69.0% | 82.3% | 88.1% | 3.79E-04 | pos pos:1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl wprefix:1 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel sppos | 18 |
| set33 | 12.1% | 74.9% | 69.3% | 82.2% | 87.9% | 3.75E-04 | pos pos:1 pos:-1 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:2 wsuffix:3 wsuffix:4 sprel | 17 |
| set40 | 11.5% | 74.9% | 70.1% | 82.2% | 88.5% | 3.91E-04 | pos pos:1 lemma wminlen:3 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel | 17 |
| set21 | 12.7% | 74.9% | 68.2% | 83.8% | 87.3% | 3.80E-04 | pos pos:1 pos:-1 lemma wminlen:2 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:2 wsuffix:4 sprel sppos | 18 |
| set86 | 12.2% | 74.6% | 69.4% | 81.8% | 87.8% | 3.74E-04 | pos pos:-1 lemma wminlen:2 wminlen:3 wminlen:5 whascl whasdigit wprefix:1 wprefix:2 wsuffix:1 wsuffix:3 | 12 |
| set6 | 12.1% | 74.5% | 68.8% | 82.4% | 87.9% | 3.78E-04 | pos pos:1 pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:3 wsuffix:4 sprel sppos | 20 |
| set64 | 11.6% | 74.4% | 69.4% | 80.6% | 88.4% | 3.78E-04 | pos pos:1 lemma wminlen:3 wminlen:4 wminlen:6 whascl wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 sprel | 14 |
| set104 | 11.5% | 74.4% | 69.1% | 81.2% | 88.5% | 3.89E-04 | pos pos:1 whascl whasdigit wprefix:1 wprefix:2 wprefix:4 wsuffix:3 wsuffix:4 sprel | 10 |
| set100 | 11.3% | 74.4% | 71.0% | 78.9% | 88.7% | 3.75E-04 | pos:-1 lemma wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl wprefix:1 wprefix:3 wsuffix:2 wsuffix:4 | 11 |
| set7 | 11.2% | 74.3% | 69.7% | 80.2% | 88.8% | 3.74E-04 | pos pos:1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel sppos | 20 |
| set58 | 12.9% | 74.3% | 67.7% | 83.3% | 87.1% | 3.73E-04 | pos pos:1 pos:-1 wminlen:3 wminlen:5 wminlen:6 whascl wprefix:1 wprefix:2 wprefix:3 wsuffix:1 wsuffix:3 wsuffix:4 sprel sppos | 15 |
| set60 | 12.8% | 74.3% | 67.7% | 84.2% | 87.2% | 3.80E-04 | pos pos:1 pos:-1 lemma wminlen:2 wminlen:4 wminlen:5 wminlen:6 whascl wprefix:1 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 sppos | 15 |
| set76 | 11.2% | 74.3% | 69.6% | 80.2% | 88.8% | 3.80E-04 | pos:-1 lemma wminlen:2 wminlen:5 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:3 sprel sppos | 13 |
| set20 | 11.7% | 74.1% | 69.6% | 79.9% | 88.3% | 3.76E-04 | pos:1 pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel sppos | 19 |
| set5 | 11.9% | 74.0% | 68.2% | 81.7% | 88.1% | 3.80E-04 | pos pos:1 pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel sppos | 20 |
| set88 | 10.6% | 73.9% | 70.5% | 78.7% | 89.4% | 3.81E-04 | pos:-1 lemma wminlen:3 wminlen:4 wminlen:5 whascl wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:3 | 12 |
| set106 | 10.5% | 73.8% | 71.1% | 77.3% | 89.5% | 3.78E-04 | pos:-1 lemma wminlen:2 wminlen:4 wminlen:5 whascl whasdigit wprefix:2 wprefix:3 sprel | 10 |
| set13 | 11.7% | 73.8% | 68.4% | 80.8% | 88.3% | 3.78E-04 | pos pos:1 lemma wminlen:2 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel sppos | 19 |
| set32 | 12.2% | 73.7% | 68.5% | 80.4% | 87.8% | 3.77E-04 | pos pos:-1 wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel | 17 |
| set97 | 11.9% | 73.7% | 70.2% | 79.6% | 88.1% | 3.78E-04 | pos:-1 wminlen:2 wminlen:6 whascl wprefix:1 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 | 11 |
| set117 | 13.5% | 73.7% | 67.2% | 81.9% | 86.5% | 3.63E-04 | pos:-1 wminlen:3 wminlen:4 wminlen:6 wprefix:1 wprefix:4 wsuffix:2 wsuffix:3 wsuffix:4 | 9 |
| set36 | 13.2% | 73.7% | 67.0% | 82.6% | 86.8% | 3.69E-04 | pos pos:1 pos:-1 lemma wminlen:2 wminlen:4 wminlen:5 whascl whasdigit wprefix:1 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 sprel sppos | 17 |
| set4 | 13.1% | 73.7% | 66.8% | 83.0% | 86.9% | 3.71E-04 | pos pos:1 pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel sppos | 20 |
| set11 | 13.9% | 73.7% | 65.8% | 84.0% | 86.1% | 3.73E-04 | pos pos:1 pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 sprel sppos | 19 |
| set81 | 11.2% | 73.6% | 69.7% | 79.6% | 88.8% | 3.81E-04 | pos:1 wminlen:2 wminlen:3 wminlen:6 wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:3 wsuffix:4 sprel | 12 |
| set28 | 13.7% | 73.6% | 66.6% | 83.4% | 86.3% | 3.72E-04 | pos pos:1 pos:-1 lemma wminlen:2 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:4 sppos | 18 |
| set41 | 12.9% | 73.6% | 66.8% | 82.6% | 87.1% | 3.76E-04 | pos pos:-1 lemma wminlen:3 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sppos | 16 |
| set110 | 14.0% | 73.5% | 66.1% | 83.9% | 86.0% | 3.73E-04 | pos:1 wminlen:2 wminlen:3 wminlen:5 wminlen:6 wprefix:1 wprefix:2 wsuffix:1 wsuffix:2 sprel | 10 |
| set9 | 11.7% | 73.5% | 68.6% | 80.3% | 88.3% | 3.79E-04 | pos pos:1 pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:4 sprel sppos | 20 |
| set0 | 12.9% | 73.5% | 67.1% | 82.1% | 87.1% | 3.73E-04 | pos pos:1 pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel sppos | 21 |
| set78 | 10.0% | 73.4% | 71.8% | 75.8% | 90.0% | 3.85E-04 | pos lemma wminlen:2 wminlen:4 wminlen:5 whasdigit wprefix:2 wprefix:3 wprefix:4 wsuffix:2 wsuffix:4 sprel sppos | 13 |
| set53 | 12.4% | 73.4% | 68.0% | 80.3% | 87.6% | 3.73E-04 | pos pos:1 wminlen:2 wminlen:3 wminlen:4 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:2 wsuffix:3 sprel | 15 |
| set24 | 12.0% | 73.4% | 67.9% | 80.4% | 88.0% | 3.79E-04 | pos:1 pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:2 wsuffix:4 sprel sppos | 18 |
| set22 | 12.9% | 73.4% | 67.5% | 81.7% | 87.1% | 3.72E-04 | pos pos:1 pos:-1 lemma wminlen:2 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:4 sprel sppos | 18 |
| set18 | 10.8% | 73.4% | 70.1% | 77.9% | 89.2% | 3.72E-04 | pos pos:1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sppos | 19 |
| set14 | 13.2% | 73.4% | 66.4% | 82.4% | 86.8% | 3.70E-04 | pos pos:1 pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel sppos | 19 |
| set68 | 10.5% | 73.3% | 70.2% | 77.4% | 89.5% | 3.82E-04 | pos lemma wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:3 wprefix:4 wsuffix:2 wsuffix:3 sprel | 14 |
| set26 | 13.4% | 73.3% | 66.4% | 82.2% | 86.6% | 3.63E-04 | pos pos:1 pos:-1 wminlen:2 wminlen:3 wminlen:4 wminlen:5 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:3 wsuffix:4 sprel sppos | 18 |
| set105 | 11.0% | 73.2% | 70.2% | 76.8% | 89.0% | 3.64E-04 | pos:-1 lemma wminlen:2 wminlen:3 wminlen:6 whasdigit wprefix:3 wsuffix:1 wsuffix:2 wsuffix:3 | 10 |
| set15 | 13.2% | 73.1% | 66.3% | 82.6% | 86.8% | 3.72E-04 | pos pos:1 pos:-1 lemma wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel sppos | 19 |
| set161 | 10.5% | 73.1% | 70.7% | 76.4% | 89.5% | 3.75E-04 | pos:-1 wprefix:4 wsuffix:2 wsuffix:4 | 4 |
| set102 | 12.1% | 73.1% | 67.7% | 79.9% | 87.9% | 3.70E-04 | pos:-1 lemma wminlen:3 wminlen:4 whasdigit wprefix:1 wprefix:2 wprefix:4 wsuffix:3 sppos | 10 |

**Figure B.2:** Measured results for the "linktype" classification. The table contains top 50 results sorted by $F1$.

| FsetID | FallOut | F1 | Precision | Recall | SP | MCC | Context | #ctx |
|---|---|---|---|---|---|---|---|---|
| set12 | 5.2% | 88.1% | 87.3% | 89.6% | 94.8% | 4.75E-04 | roleInLink:-1 roleInLink:-2 pos pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel | 20 |
| set96 | 4.9% | 88.1% | 88.6% | 88.0% | 95.1% | 4.75E-04 | roleInLink:-1 roleInLink:-2 pos:-1 wminlen:4 wminlen:5 wminlen:6 wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 | 12 |
| set67 | 5.9% | 88.0% | 86.3% | 90.0% | 94.1% | 4.65E-04 | roleInLink:-1 roleInLink:-2 pos pos:-1 lemma wminlen:2 wminlen:4 wminlen:6 whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:4 sprel | 15 |
| set29 | 5.6% | 87.4% | 86.3% | 89.1% | 94.4% | 4.59E-04 | roleInLink:-1 pos pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 sprel | 19 |
| set26 | 6.1% | 87.3% | 86.2% | 89.0% | 93.9% | 4.60E-04 | roleInLink:-1 roleInLink:-2 pos:1 pos:-1 lemma wminlen:2 wminlen:3 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel | 19 |
| set115 | 4.7% | 87.3% | 87.6% | 87.5% | 95.3% | 4.70E-04 | roleInLink:-1 roleInLink:-2 pos:-1 wminlen:2 whasdigit wprefix:1 wprefix:3 wprefix:4 wsuffix:1 wsuffix:4 | 10 |
| set55 | 5.9% | 87.3% | 85.5% | 89.6% | 94.1% | 4.62E-04 | roleInLink:-1 roleInLink:-2 pos pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wprefix:1 wprefix:3 wprefix:4 wsuffix:2 wsuffix:3 wsuffix:4 sprel | 16 |
| set41 | 6.8% | 87.1% | 84.8% | 89.9% | 93.2% | 4.56E-04 | roleInLink:-1 roleInLink:-2 pos pos:-1 lemma wminlen:4 wminlen:5 wminlen:6 whascl wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:3 wsuffix:4 sprel | 17 |
| set19 | 5.6% | 87.1% | 86.3% | 88.4% | 94.4% | 4.61E-04 | roleInLink:-1 roleInLink:-2 pos pos:1 pos:-1 lemma wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel | 20 |
| set60 | 6.0% | 87.1% | 85.8% | 89.0% | 94.0% | 4.65E-04 | roleInLink:-1 roleInLink:-2 pos:-1 lemma wminlen:3 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:2 wsuffix:3 sprel | 16 |
| set8 | 6.6% | 87.0% | 84.7% | 89.7% | 93.4% | 4.62E-04 | roleInLink:-1 roleInLink:-2 pos pos:1 pos:-1 wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel | 21 |
| set62 | 6.4% | 86.8% | 85.4% | 88.7% | 93.6% | 4.61E-04 | roleInLink:-1 pos pos:1 pos:-1 wminlen:2 wminlen:4 whascl whasdigit wprefix:1 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 sprel | 15 |
| set64 | 6.2% | 86.8% | 84.8% | 89.7% | 93.8% | 4.59E-04 | roleInLink:-1 roleInLink:-2 pos pos:-1 lemma wminlen:2 wminlen:4 wminlen:5 wminlen:6 whasdigit wprefix:1 wprefix:4 wsuffix:2 wsuffix:3 sprel | 15 |
| set50 | 5.6% | 86.8% | 86.0% | 88.1% | 94.4% | 4.64E-04 | roleInLink:-1 roleInLink:-2 pos pos:-1 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:2 wsuffix:4 sprel | 17 |
| set5 | 5.8% | 86.8% | 86.0% | 87.9% | 94.2% | 4.57E-04 | roleInLink:-1 roleInLink:-2 pos pos:1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel | 21 |
| set4 | 6.3% | 86.8% | 84.9% | 88.8% | 93.7% | 4.55E-04 | roleInLink:-1 roleInLink:-2 pos pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel | 21 |
| set13 | 6.6% | 86.6% | 84.7% | 88.9% | 93.4% | 4.59E-04 | roleInLink:-1 roleInLink:-2 pos pos:1 pos:-1 lemma wminlen:2 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:2 wsuffix:3 wsuffix:4 sprel | 20 |
| set40 | 5.7% | 86.6% | 86.6% | 87.1% | 94.3% | 4.56E-04 | roleInLink:-1 pos pos:1 pos:-1 wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:3 wsuffix:4 sprel | 18 |
| set37 | 6.4% | 86.6% | 84.2% | 89.5% | 93.6% | 4.62E-04 | roleInLink:-1 roleInLink:-2 pos pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wsuffix:1 wsuffix:3 wsuffix:4 sprel | 18 |
| set59 | 5.5% | 86.5% | 86.0% | 87.5% | 94.5% | 4.65E-04 | roleInLink:-1 roleInLink:-2 pos:-1 lemma wminlen:3 wminlen:4 wminlen:5 wminlen:6 whasdigit wprefix:1 wprefix:2 wprefix:4 wsuffix:1 wsuffix:3 wsuffix:4 sprel | 16 |
| set30 | 6.4% | 86.5% | 84.4% | 89.0% | 93.6% | 4.61E-04 | roleInLink:-1 roleInLink:-2 pos pos:1 pos:-1 lemma wminlen:2 wminlen:4 wminlen:5 wminlen:6 whascl wprefix:1 wprefix:2 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel | 19 |
| set132 | 4.4% | 86.5% | 87.9% | 85.6% | 95.6% | 4.70E-04 | roleInLink:-1 pos:-1 lemma wminlen:2 wminlen:4 wminlen:6 whascl wprefix:4 | 8 |
| set34 | 5.7% | 86.5% | 86.1% | 87.1% | 94.3% | 4.62E-04 | roleInLink:-1 roleInLink:-2 pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 sprel | 18 |
| set80 | 6.2% | 86.5% | 84.4% | 89.0% | 93.8% | 4.53E-04 | roleInLink:-1 pos pos:1 pos:-1 wminlen:3 wminlen:6 whasdigit wprefix:1 wprefix:2 wprefix:3 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 | 14 |
| set35 | 6.7% | 86.5% | 84.1% | 89.4% | 93.3% | 4.52E-04 | roleInLink:-1 roleInLink:-2 pos pos:1 pos:-1 lemma wminlen:2 wminlen:3 wminlen:5 whasdigit wprefix:1 wprefix:2 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel | 18 |
| set3 | 6.8% | 86.5% | 83.8% | 89.8% | 93.2% | 4.54E-04 | roleInLink:-1 roleInLink:-2 pos pos:1 pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel | 21 |
| set11 | 6.0% | 86.4% | 85.1% | 88.7% | 94.0% | 4.55E-04 | roleInLink:-1 roleInLink:-2 pos:1 pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 | 20 |
| set57 | 6.4% | 86.4% | 84.7% | 88.6% | 93.6% | 4.48E-04 | roleInLink:-1 pos:1 lemma wminlen:2 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:4 sprel | 16 |
| set49 | 6.4% | 86.4% | 84.6% | 88.5% | 93.6% | 4.50E-04 | roleInLink:-1 roleInLink:-2 pos pos:1 pos:-1 lemma wminlen:2 wminlen:3 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:3 sprel | 17 |
| set77 | 5.2% | 86.4% | 86.7% | 86.6% | 94.8% | 4.59E-04 | roleInLink:-1 roleInLink:-2 pos pos:-1 lemma wminlen:3 wminlen:5 wminlen:6 whasdigit wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:4 | 14 |
| set58 | 6.9% | 86.3% | 83.7% | 89.5% | 93.1% | 4.50E-04 | roleInLink:-1 roleInLink:-2 pos pos:1 pos:-1 wminlen:2 wminlen:3 wminlen:4 wminlen:5 whasdigit wprefix:1 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 sprel | 16 |
| set44 | 6.7% | 86.2% | 84.5% | 88.4% | 93.3% | 4.53E-04 | roleInLink:-1 pos pos:1 pos:-1 lemma wminlen:2 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 sprel | 17 |
| set25 | 6.7% | 86.2% | 84.2% | 88.7% | 93.3% | 4.46E-04 | roleInLink:-1 pos pos:1 pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:3 wsuffix:4 | 19 |
| set28 | 6.4% | 86.2% | 84.6% | 88.3% | 93.6% | 4.48E-04 | roleInLink:-1 roleInLink:-2 pos pos:1 pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:2 wsuffix:3 wsuffix:4 | 19 |
| set46 | 6.0% | 86.2% | 85.3% | 87.5% | 94.0% | 4.56E-04 | roleInLink:-1 pos pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:3 wprefix:4 wsuffix:1 wsuffix:4 sprel | 17 |
| set84 | 6.0% | 86.2% | 85.8% | 86.9% | 94.0% | 4.52E-04 | roleInLink:-1 roleInLink:-2 pos pos:-1 lemma wminlen:2 wminlen:3 wminlen:5 wminlen:6 whascl whasdigit wsuffix:1 wsuffix:3 | 13 |
| set119 | 6.9% | 86.2% | 84.3% | 88.4% | 93.1% | 4.58E-04 | roleInLink:-1 pos pos:-1 lemma wminlen:3 whascl wprefix:1 wprefix:2 wsuffix:1 wsuffix:4 | 10 |
| set52 | 6.0% | 86.1% | 84.7% | 87.9% | 94.0% | 4.52E-04 | roleInLink:-1 pos pos:-1 lemma wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:2 wprefix:3 wsuffix:1 wsuffix:2 wsuffix:4 sprel | 16 |
| set33 | 6.2% | 86.1% | 84.9% | 88.1% | 93.8% | 4.59E-04 | roleInLink:-1 roleInLink:-2 pos:1 pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whasdigit wprefix:1 wprefix:3 wprefix:4 wsuffix:2 wsuffix:3 wsuffix:4 sprel | 18 |
| set36 | 7.4% | 86.1% | 83.7% | 89.1% | 92.6% | 4.47E-04 | roleInLink:-1 pos pos:1 pos:-1 lemma wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 sprel | 18 |
| set86 | 5.7% | 86.1% | 86.9% | 85.7% | 94.3% | 4.69E-04 | roleInLink:-1 roleInLink:-2 pos:-1 lemma wminlen:2 wminlen:3 wminlen:6 whascl wprefix:2 wsuffix:1 wsuffix:2 wsuffix:3 wsuffix:4 | 13 |
| set42 | 6.5% | 86.0% | 84.4% | 88.0% | 93.5% | 4.50E-04 | roleInLink:-1 roleInLink:-2 pos:1 pos:-1 lemma wminlen:3 wminlen:4 wminlen:5 whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:2 wsuffix:3 wsuffix:4 sprel | 17 |
| set97 | 5.3% | 85.9% | 87.2% | 85.5% | 94.7% | 4.58E-04 | roleInLink:-1 roleInLink:-2 pos wminlen:2 wminlen:3 wminlen:5 wminlen:6 whascl whasdigit wprefix:2 wprefix:4 wsuffix:4 | 12 |
| set51 | 7.1% | 85.9% | 83.2% | 89.0% | 92.9% | 4.46E-04 | roleInLink:-1 roleInLink:-2 pos pos:1 pos:-1 lemma wminlen:4 whascl wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:2 wsuffix:3 wsuffix:4 sprel | 16 |
| set45 | 7.2% | 85.9% | 83.4% | 89.5% | 92.8% | 4.49E-04 | roleInLink:-1 roleInLink:-2 pos pos:1 pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 whascl whasdigit wprefix:1 wprefix:3 wprefix:4 wsuffix:2 wsuffix:3 sprel | 17 |
| set94 | 6.3% | 85.9% | 84.9% | 87.1% | 93.7% | 4.52E-04 | roleInLink:-1 roleInLink:-2 pos pos:1 lemma wminlen:3 wminlen:4 wminlen:5 wminlen:6 wprefix:2 wprefix:3 wsuffix:4 | 12 |
| set99 | 5.8% | 85.9% | 85.7% | 86.3% | 94.2% | 4.54E-04 | roleInLink:-1 roleInLink:-2 pos:-1 lemma wminlen:4 wminlen:5 wminlen:6 whasdigit wsuffix:2 wsuffix:3 wsuffix:4 sprel | 12 |
| set39 | 6.5% | 85.8% | 83.8% | 88.4% | 93.5% | 4.50E-04 | roleInLink:-1 roleInLink:-2 pos pos:1 pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 whasdigit wprefix:1 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:4 sprel | 18 |
| set10 | 7.0% | 85.8% | 83.2% | 89.1% | 93.0% | 4.50E-04 | roleInLink:-1 roleInLink:-2 pos pos:1 pos:-1 lemma wminlen:2 wminlen:3 wminlen:4 wminlen:5 wminlen:6 whascl whasdigit wprefix:1 wprefix:2 wprefix:3 wprefix:4 wsuffix:1 wsuffix:2 wsuffix:3 sprel | 21 |
| set82 | 7.2% | 85.7% | 83.0% | 88.7% | 92.8% | 4.50E-04 | roleInLink:-1 roleInLink:-2 pos:1 pos:-1 lemma wminlen:2 wminlen:6 whascl wprefix:1 wprefix:2 wprefix:4 wsuffix:3 sprel | 13 |

**Figure B.3:** Measured results for the "roleInLink" classification. The table contains top 50 results sorted by $F1$.

# Appendix C

# Training data used for the evaluation

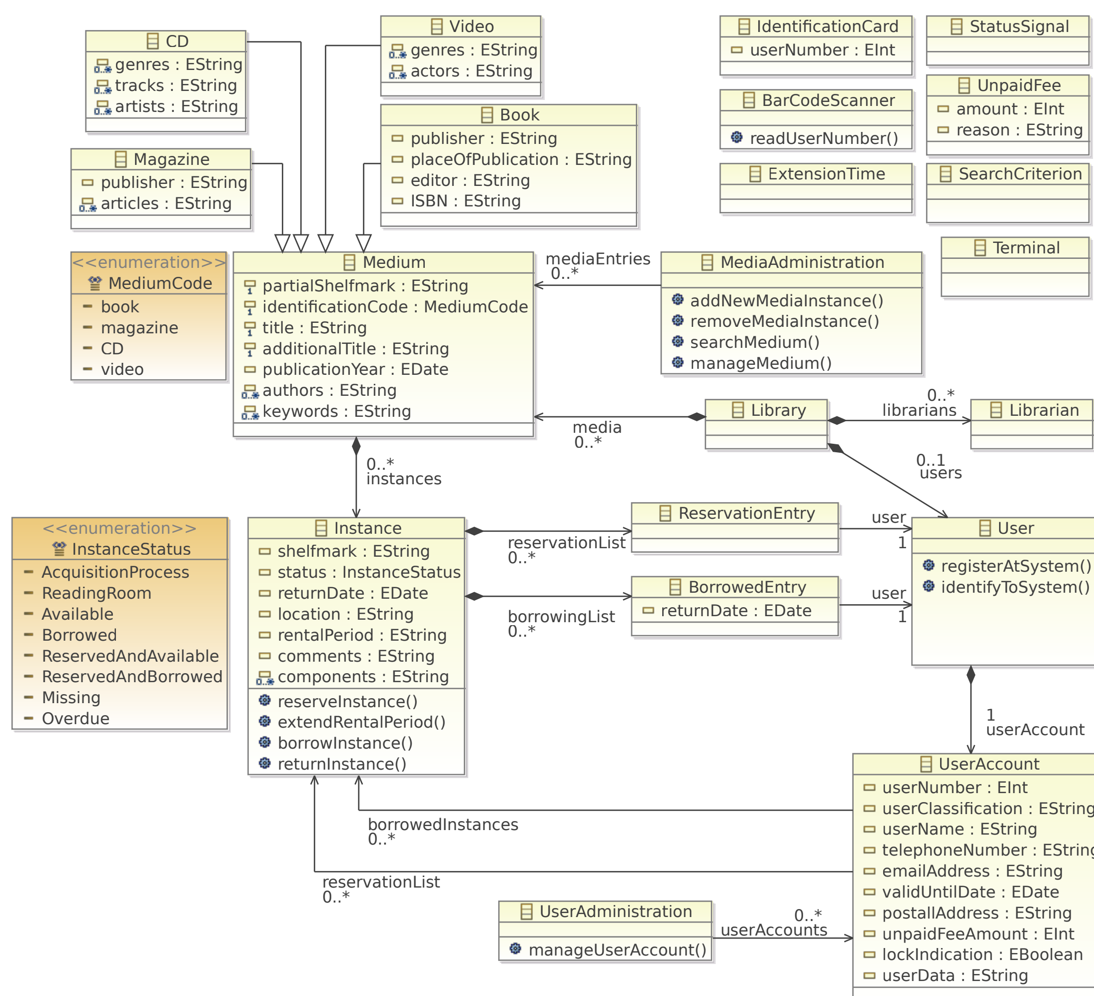



**Figure C.1:** Domain model for the Library System used for training in our experiment.

# Library System Specification

## 1. Objective

Some Media and user of a library are managed by the library system.

## 2. Operational Area

The library system is operated by the library staff (librarian) and library user (user) through terminals.

## 3. Product overview

The library system contains a user administration and a media administration.

### 3.1 User administration

The user administration contains a user account for each user which contains all user data. A librarian is able to create a new user account, to edit and to delete an existing user account. A user is able to register at the system with his user number, to manage his user account, and to extend the media's rental period. A password is not necessary because the user number on the identification card is read with a bar code scanner.

### 3.2 Media administration

The media administration contains an entry for each medium in the library. Several instances of each medium may be available, and they may have different locations. A librarian is able to add a new media instance to the media administration, change the status of an instance and remove an instance from the media administration. A user is able to search for a medium by specifying one or more features of the media. User can choose and reserve a found instance. To this end, the user number on the identification card is scanned. If a user borrows an instance, it is added to his account and will only be deleted when it is returned.

## 4. Product function

A user account must be available for each user. A librarian has access to all accounts of all users. A user has only access to his own user account by using his user number.

- A user is able to search a medium (search medium).
- A user is able to reserve one or more instances (reserve instance).
- A user is able to borrow one or more instances (borrow instance).
- A user is able to extend the rental period of one or more instances (extend instance).
- A user is able to return one or more instances (return instance).
- A librarian is able to create, edit and delete a user account (manage user).
- A librarian is able to create, edit and delete a medium (manage medium).

### 4.1. Use Case: Search medium

*The user specifies one or more titles and obtains a list of media which contain the titles. The search can be restricted to designated media types and attributes.*

Procedure:

- 1. The user specifies one or more search criteria.
- 2. It is searched for matching media in the media administration.
- 3. The user receives a list of media that match the search criteria.

### 4.2 Use Case: Reserve instance

*The user specifies an instance which he would like to reserve (e.g. by selection after a search) and has to identify himself to the system. The instance is registered in the reservation list of his user account. The status of the instance*

*is set to “available and reserved” or to “borrowed and reserved”.*

Procedure:

- 1. The user enters the shelfmark of the requested medium.
- 2. The user receives a list of instances of the medium.
- 3. The user chooses an instance.
- 4. The user identifies himself to the system.
- **Extension**: if the user account is not valid any more or closed
- The procedure is terminated with a status signal.
- 6. The status of the instance is set from “available” to “available and reserved” or from “borrowed” to “borrowed and reserved”.

### 4.3. Use Case: Borrow instance

*The user has to identify himself to the system and specifies a list of instances he wishes to borrow. The instances are listed in the borrowing list of his user account. Only instances can be borrowed which are “available”, or “available and reserved” and the user is the first on the reservation list.*

Procedure:

- 1. The user identifies himself to the system and chooses the ’borrow’ menu.
- **Extension**: If the user account is not valid any more or closed.
- 1. The procedure is terminated.
- 3. The user specifies an instance which should be borrowed (shelfmark of the instance is scanned).
- 4. If the instance’s status is “available” or “reserved and available”, and the user is the first on the reservation list, the instance can be borrowed otherwise borrowing the instance is rejected.
- 5. The instance is registered in the list of borrowed media of the user account.
- 6. The user is registered in the list of users who have/had borrowed the instance.
- 7. The status of the instance is changed to “borrowed” or “reserved and borrowed”.
- 8. The instance is unlocked.
- **Extension**: If a further instance shall be borrowed
- 1. Continue at step 3.
- 10. The user receives a receipt for the newly borrowed instance.