Formal Models from Controlled Natural Language via Cognitive Grammar and Configuration

by

Matt Selway
Bachelor of Computer Science (Honours)

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

School of Information Technology & Mathematical Sciences
Division of Information Technology, Engineering and the Environment

February 2016
“The message is written, the meaning is missing.”

P.O.D., *Sleeping Awake*

Deanna Troi: “Computer, show me a table.”

Enterprise Computer: “There are 5047 classifications of tables on file. Specify design parameters.”

*Star Trek: The Next Generation, Season 6, Episode 5*
# Table of Contents

- **Table of Contents** iv
- **List of Figures** ix
- **List of Tables** xiii
- **List of Abbreviations** xv
- **Abstract** xvii
- **Declaration of Authorship** xix
- **Acknowledgements** xx
- **Preface** xxi

## 1 Introduction 1
- 1.1 Aim, Contributions, and Scope 5
- 1.2 Overview of the Thesis 8

## 2 Background 11
- 2.1 Model-Driven Engineering 11
  - 2.1.1 Models 12
  - 2.1.2 Model Transformations 12
  - 2.1.3 The Model-Driven Architecture 15
  - 2.1.4 Challenges in MDE 17
- 2.2 Requirements Engineering 17
- 2.3 Natural Language Processing 19
  - 2.3.1 Controlled Natural Languages 21
  - 2.3.2 Cognitive Linguistics 22
    - 2.3.2.1 Cognitive Grammar 23
- 2.4 Knowledge-based Configuration 26
- 2.5 Summary 28

## 3 Survey of existing Tools/Techniques 31
- 3.1 Overview of Related Work 32
# Table of Contents

3.1.1 Formalist Approaches ........................................ 32
3.1.2 Information Extraction Approaches .......................... 34
3.1.3 Naturalist Approaches ...................................... 35
3.2 Tool Comparison .................................................. 37
3.2.1 Suitability of Criteria ...................................... 44
3.2.2 Results .......................................................... 45
3.3 Summary ........................................................... 51

4 Knowledge-based Framework for CNL Understanding .......... 53
4.1 Proposed Solution and Hypotheses .............................. 54
4.2 Lexicon ............................................................. 56
4.3 Syntactic Analysis ................................................ 60
4.3.1 Pruning Strategy ............................................. 79
4.3.2 Parse Trees ................................................... 81
4.3.3 Tokenisation (and Internal Expectations) ................. 82
4.4 Semantic Analysis ................................................ 89
4.4.1 Configuration Model Knowledge ............................ 90
4.4.2 Configuration Solution Knowledge ......................... 95
4.4.3 Configuration Requirements Knowledge .................. 98
4.4.4 Configuration Result ........................................ 100
4.4.5 More Complex Language .................................... 101
4.4.6 Interactive Feedback, Alternatives, and Corrections .... 107
4.5 Summary ........................................................... 108

5 Application to Specifications using SBVR ......................... 111
5.1 Semantics as SBVR ............................................... 112
5.1.1 SBVR Metamodel ............................................ 113
5.1.2 SBVR Structured English .................................. 114
5.2 Process Overview and Architecture ............................ 117
5.3 Lexicon ............................................................. 122
5.4 Syntactic Analysis ................................................. 127
5.4.1 Tokenisation Strategy ........................................ 128
5.4.2 Pruning Strategy .............................................. 129
5.4.3 Maximum Catching Distance ............................... 132
5.4.4 Catch/Caught Weights ....................................... 132
5.4.5 Implemented Algorithm ..................................... 134
5.5 Semantic Analysis ................................................. 139
5.5.1 Configuration Model Knowledge ............................ 140
5.5.2 Configuration Solution Knowledge ......................... 145
5.5.3 Configuration Requirements Knowledge .................. 145
5.5.4 Configuration Result ......................................... 148
5.5.5 Interactive Feedback, Alternatives, and Corrections .... 148
5.6 Lexicon Acquisition .............................................. 151
5.6.1 From Glossary ................................................ 152
5.6.2 From Unrestricted Text ....................................... 155
Table of Contents

5.6.2.1 General Process ............................................ 156
5.6.2.2 Initial Parsing ........................................... 157
5.6.2.3 Pre-processing ........................................... 157
5.6.2.4 Candidate Extraction ................................. 161
5.6.2.5 Disambiguation of Compound Nouns .................. 168
5.6.2.6 Resolution of Verb Concepts ......................... 172
5.6.2.7 Object Type/Individual Concept Determination ...... 173
5.6.2.8 Final Filtering ........................................... 175
5.6.2.9 Presentation to the User .............................. 175

5.7 Summary .......................................................... 176

6 Evaluation of the Application ................................. 177

6.1 SBVR Lexicon Acquisition from Glossary ............... 178
   6.1.1 Criterion ............................................... 179
   6.1.2 Measure ............................................... 179
   6.1.3 Method ............................................... 180
   6.1.4 Results ............................................... 183
   6.1.5 Discussion ............................................ 184

6.2 SBVR Lexicon Acquisition from “Unrestricted” Text .... 185
   6.2.1 Preliminaries .......................................... 185
   6.2.2 Criterion ............................................. 187
   6.2.3 Measure ............................................. 188
   6.2.4 Method ............................................. 193
   6.2.5 Results ............................................. 211
   6.2.6 Discussion ........................................... 216
   6.2.7 Limitations of this Comparison ..................... 218

6.3 Cognitive Linguistic Rule Parsing of SBVR ............... 219
   6.3.1 Criterion ............................................. 219
   6.3.2 Measure ............................................. 220
   6.3.3 Method ............................................. 220
   6.3.4 Results ............................................. 223
   6.3.5 Discussion ........................................... 224

6.4 Run-time Performance ...................................... 225
   6.4.1 Criterion ............................................. 225
   6.4.2 Measure ............................................. 225
   6.4.3 Method ............................................. 226
   6.4.4 Results ............................................. 228
   6.4.5 Discussion ........................................... 228

6.5 Summary .......................................................... 229

7 Discussion and Conclusions .................................. 231

7.1 Comparison to Surveyed Approaches ..................... 233
7.2 Limitations and Future Work ............................... 238
Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bibliography</td>
<td>243</td>
</tr>
<tr>
<td>Appendix A: PENS Classification Framework</td>
<td>263</td>
</tr>
<tr>
<td>Appendix B: Configuration Art</td>
<td>267</td>
</tr>
</tbody>
</table>
# List of Figures

1.1 Relationships between Chapters of the Thesis .................................................. 8

2.1 Three-level Model Architecture ................................................................. 13

2.2 Concepts of Model Transformation ............................................................ 14

2.3 MDA Framework ......................................................................................... 15

2.4 Symbolic structures in Cognitive Grammar at varying levels of composition. The phonological poles are indicated by $P$ and the semantic poles by $S$ ................................................. 24

2.5 Example Semantic Structure ....................................................................... 25

2.6 Superimposition of Semantic Structures .................................................... 25

2.7 Intersection of research fields .................................................................... 29

4.1 Lexical entries using Langacker’s Image Schema style notation. (a) Basic lexical entry for something and the word that represents it, (b) A lexical entry with a grammatical expectation .................................................. 59

4.2 Example lexicon .......................................................................................... 61

4.3 Sketch of the syntactic analysis algorithm ................................................. 62

4.4 Sketch of the suggestion combination algorithm ....................................... 63

4.5 Suggestion list with catching distances for a sentence including a relative clause ........................................................................................................ 67

4.6 Suggestion list with catching distances for a sentence with a relative clause and an internal expectation ........................................................................ 69

4.7 Suggestion list displaying the binding energy of a sentence including a catching distance greater than zero. Interesting binding energies are indicated as BE ................................................................................................................................................. 70

4.8 Suggestion combination algorithm supporting negative catching distances .................................................................................................. 72

4.9 Different cases of the negative catching distances showing the catching distance (CD) and binding energy (BE). They are shown for each sub-row, e.g. (1.1), being caught by its main row, e.g. (1), independently of one another. Only cases 1.1, 1.2, and 2.2 are valid catches ......................................................................................................................... 73

4.10 Local coverage vs. binding energy of suggestions .................................... 75

4.11 Suggestion list displaying the distance to the current token (D) and the global coverage (GC) of each suggestion at the end of the token stream ................................................. 78

4.12 Comparison of an expectation-based parse tree (a), a traditional constituency tree (b), and a dependency tree (c) ....................................................................................... 81

4.13 Expectation-based analysis of multi-word expressions ................................ 85

4.14 Expectation-based parse trees for multi-word expressions: (a) analysis with separate lexical entries, and (b) analysis using the single lexical entry ......................................................... 85

4.15 Expectation-based analysis of a negated multi-word expression ............. 88
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.16</td>
<td>Simple Conceptual Graphs (constrained) meta-model in UML notation.</td>
<td>91</td>
</tr>
<tr>
<td>4.17</td>
<td>Simple Conceptual Graphs relaxed meta-model in UML notation.</td>
<td>91</td>
</tr>
<tr>
<td>4.18</td>
<td>Simple conceptual graph and its first-order logic equivalent for Matt lends John a car.</td>
<td>94</td>
</tr>
<tr>
<td>4.19</td>
<td>Final parse tree and suggestions for Some branch is included in a local area.</td>
<td>97</td>
</tr>
<tr>
<td>4.20</td>
<td>Basic suggestion mapping rules for the simple Conceptual Graph semantics.</td>
<td>98</td>
</tr>
<tr>
<td>4.21</td>
<td>Partial Conceptual Graph model for Some branch is included in a local area after applying the mapping rules.</td>
<td>98</td>
</tr>
<tr>
<td>4.22</td>
<td>Partial Conceptual Graph model for Some branch is included in a local area including catch constraints (dotted lines).</td>
<td>100</td>
</tr>
<tr>
<td>4.23</td>
<td>Conceptual Graph meta-model including constraint relations.</td>
<td>100</td>
</tr>
<tr>
<td>4.24</td>
<td>Configured Conceptual Graph model for Some branch is included in a local area.</td>
<td>101</td>
</tr>
<tr>
<td>4.25</td>
<td>Extract of the Conceptual Graph meta-model incorporating additional constraint relations and operations necessary to ensure correct negation scoping.</td>
<td>104</td>
</tr>
<tr>
<td>4.26</td>
<td>Negation term mapping rules for the simple Concept Graph semantics.</td>
<td>105</td>
</tr>
<tr>
<td>4.27</td>
<td>Conceptual Graph model of the sentence Some branch is included in each local area: top half shows the partial model, bottom half the configured model with correct quantifier scoping.</td>
<td>106</td>
</tr>
<tr>
<td>4.28</td>
<td>Conceptual Graph of the complex scoping of No branch is not included in each local area.</td>
<td>108</td>
</tr>
<tr>
<td>5.1</td>
<td>Extract of the SBVR metamodel [OMG08b]: Meanings and Representations</td>
<td>114</td>
</tr>
<tr>
<td>5.2</td>
<td>Extract of the SBVR metamodel [OMG08b]: Logical Formulations</td>
<td>114</td>
</tr>
<tr>
<td>5.3</td>
<td>EU-Rent case study overview</td>
<td>118</td>
</tr>
<tr>
<td>5.4</td>
<td>Semi-formalised EU-Rent business rules</td>
<td>118</td>
</tr>
<tr>
<td>5.5</td>
<td>Semi-formalised EU-Rent business vocabulary</td>
<td>118</td>
</tr>
<tr>
<td>5.6</td>
<td>EU-Rent case study, Locations extract [OMG08b] Annex E</td>
<td>119</td>
</tr>
<tr>
<td>5.7</td>
<td>A possible workflow using our approach and details of the parsing process</td>
<td>120</td>
</tr>
<tr>
<td>5.8</td>
<td>Component diagram of the CLUE4SBVR prototype implementation of the framework.</td>
<td>123</td>
</tr>
<tr>
<td>5.9</td>
<td>Derivation of lexical entries from object types and individual concepts.</td>
<td>123</td>
</tr>
<tr>
<td>5.10</td>
<td>Derivation of lexical entries for fact types.</td>
<td>124</td>
</tr>
<tr>
<td>5.11</td>
<td>Semantics of Business Vocabulary and Business Rules (SBVR) model of the example vocabulary and the lexicon derived from it. Grey dotted arrows indicate relations between different vocabulary entries.</td>
<td>125</td>
</tr>
<tr>
<td>5.12</td>
<td>Example lexical entries for the main categories of logical formulation: Quantifications (a), Modalities (b), and Logical Operations (c). The dashed line in the lexical entry for (always) indicates a constraint between the ends of the two relations.</td>
<td>126</td>
</tr>
<tr>
<td>5.13</td>
<td>Example annotations created from lexicon JAPE grammar.</td>
<td>128</td>
</tr>
<tr>
<td>5.14</td>
<td>Parse tree with deeply nested relative clauses.</td>
<td>131</td>
</tr>
<tr>
<td>5.15</td>
<td>Syntactic analysis of a modal term within the passive form of a fact type.</td>
<td>133</td>
</tr>
<tr>
<td>5.16</td>
<td>Initial suggestion for the evokation of &quot;includes.&quot;</td>
<td>137</td>
</tr>
</tbody>
</table>
List of Figures

5.17 Composite suggestions for ‘each branch’. .................................. 138
5.18 Composite suggestions for ‘each branch is [] included in *’. .......... 139
5.19 Parse tree for ‘each branch is included in *’. ............................... 139
5.20 SBVR meta-model as a COCOA knowledge-base. Component type taxonomy and attribute definitions only. ......................... 144
5.21 Suggestion mapping rules for SBVR ........................................... 146
5.22 Partial SBVR model with evoked structures. Black bordered elements are the evoked structures, empty boxes are elements from the vocabulary, filled boxes were created by the evocation of the lexical entries. Dashed arrows are relations between different vocabulary entries. ............................... 147
5.23 Partial SBVR model with evoked structures and catch and precedes constraints. Black bordered elements are the evoked structures, empty boxes are elements from the vocabulary, filled boxes were created by the evocation of the lexical entries. Dashed arrows are relations between different vocabulary entries. ............................... 147
5.24 Desired configuration result of the example rule. Evoked structures are outlined in black; existing vocabulary elements are empty boxes; elements created through evocation are filled boxes; and relationships created by the generative constraint solver are highlighted with a dot. ................. 148
5.25 Example suggestion list with disjoint suggestions ........................ 150
6.1 Example SBVR Glossary Test Data ............................................ 181
6.2 Example SBVR vocabulary and associated key annotations. ............ 182
6.3 Categories of the unrestricted text acquisitions results. ................... 191
6.4 Example system response (by correct key and extra) and answer key. ... 201
6.5 Example answer key for rule parsing ........................................ 222
# List of Tables

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Comparison of Related Work</td>
<td>46</td>
</tr>
<tr>
<td>3.2</td>
<td>Short-list of Related Work</td>
<td>50</td>
</tr>
<tr>
<td>6.1</td>
<td>Results of Vocabulary Acquisition from EU-Rent vocabulary</td>
<td>184</td>
</tr>
<tr>
<td>6.2</td>
<td>Summary of evaluation text characteristics</td>
<td>212</td>
</tr>
<tr>
<td>6.3</td>
<td>Summary of answer key properties</td>
<td>212</td>
</tr>
<tr>
<td>6.4</td>
<td>Original case study evaluation results</td>
<td>213</td>
</tr>
<tr>
<td>6.5</td>
<td>Case study evaluation: results using the framework</td>
<td>214</td>
</tr>
<tr>
<td>6.6</td>
<td>Case study evaluation: extra, missing, and error categorisation</td>
<td>215</td>
</tr>
<tr>
<td>6.7</td>
<td>Parsing accuracy results</td>
<td>223</td>
</tr>
<tr>
<td>6.8</td>
<td>Configuration Time</td>
<td>228</td>
</tr>
<tr>
<td>6.9</td>
<td>Process Execution Time</td>
<td>228</td>
</tr>
<tr>
<td>7.1</td>
<td>Comparison of Related Work including CLUE4SBVR</td>
<td>235</td>
</tr>
<tr>
<td>7.2</td>
<td>Short-list of Related Work including CLUE4SBVR</td>
<td>238</td>
</tr>
</tbody>
</table>
List of Abbreviations

ACE  Attempto Controlled English
ATL  Atlas Transformation Language
BPMN Business Process Model and Notation
CIM  Computation Independent Model
CNL  Controlled Natural Language
COCOA Constraint-based Configuration Architecture
DSL  Domain Specific Language
EMF  Eclipse Modelling Framework
EMOF Essential MOF
ER  Entity-Relationship
ETL  Epsilon Transformation Language
GATE General Architecture for Text Engineering
IE  Information Extraction
LoA  Level of Abstraction (wrt. the levels of the Model-Driven Architecture)
MDA  Model-Driven Architecture
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDE</td>
<td>Model-Driven Engineering</td>
</tr>
<tr>
<td>MDRE</td>
<td>Model-Driven Requirements Engineering</td>
</tr>
<tr>
<td>MOF</td>
<td>Meta Object Facility</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>NLU</td>
<td>Natural Language Understanding</td>
</tr>
<tr>
<td>OCL</td>
<td>Object Constraint Language [OMG10]</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language [W3C]</td>
</tr>
<tr>
<td>PIM</td>
<td>Platform Independent Model</td>
</tr>
<tr>
<td>POS</td>
<td>part-of-speech</td>
</tr>
<tr>
<td>PSM</td>
<td>Platform Specific Model</td>
</tr>
<tr>
<td>QVT</td>
<td>Query/View/Transformation</td>
</tr>
<tr>
<td>RE</td>
<td>Requirements Engineering</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Network</td>
</tr>
<tr>
<td>SBVR</td>
<td>Semantics of Business Vocabulary and Business Rules</td>
</tr>
<tr>
<td>SBVR SE</td>
<td>SBVR Structured English</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
</tbody>
</table>
Abstract

School of Information Technology & Mathematical Sciences
Division of Information Technology, Engineering and the Environment

Doctor of Philosophy

by Matt Selway

In the past decade the rise of Model-Driven Engineering has resulted in major changes to industrial software development. Using models for describing behaviours allows replacing part of the coding activities with higher-level definitions of concepts and behaviour: commonly cited as reducing costs, errors and time-to-market, while improving software quality and communication between stakeholders. However, we continue to see spectacular failures of large, expensive IT projects, often blamed on badly specified requirements. With high-level models of business environments—indeed, independent of technical considerations—now possible and standards like the Semantics of Business Vocabulary and Business Rules (SBVR) emphasising business and domain experts as the owners of their business requirements, we are reaching the point where formal modelling of requirements by the business experts themselves is necessary. However, requirements specifications written in natural language (e.g. English) still play a fundamental role in communicating the requirements.

This thesis investigates an approach to semi-automatically transforming natural language specifications written by business/domain experts into formal models. The aim is to support the user in the process of formalising their requirements as models to reduce errors, ambiguities, inconsistencies, and the time taken.

Initially, two possible approaches to achieving this aim are identified: a precise Controlled Natural Language approach, and a more flexible Information Extraction-based approach. However, both are considered unsuitable for use by business experts as they require too much technical knowledge and time, or cannot express the necessary requirements. Therefore, a new approach that overcomes these limitations is developed in this thesis.
A general framework is presented, based on Cognitive Grammar and Knowledge-based Configuration, which focuses on the semantic analysis of controlled language. It is more flexible than a formal grammar for controlled language while preserving their desirable properties—e.g. reduced ambiguity for humans and computers—and can identify and help correct errors, ambiguities, and inconsistencies. Most of the knowledge required for analysing the text is embedded in the parser as models used for configuration, with the exception of some simple lexical information and mapping rules that evoke lexical entries and instantiate the elements required for parsing.

Following this, a prototype implementation for SBVR Structured English is created, called CLUE4SBVR, which provides concrete descriptions for each aspect of the general framework. Furthermore, two lexical acquisition are incorporated: one that learns vocabulary from a partly formalised glossary, and a another that learns candidate vocabulary entries from unrestricted text.

The prototype is evaluated on its performance in processing an SBVR SE specification. The lexical acquisition components are also evaluated, including the development of a comparative framework to more reliably compare existing IE approaches to the component for vocabulary extraction from unrestricted text.

The results show that CLUE4SBVR is capable of performing precise semantic interpretation of the CNL specification, while identifying ambiguity as intended. Furthermore, the lexicon acquisition from glossary performed very well, while acquisition of candidate vocabulary from unrestricted text obtained similar results to the compared approaches. However, the prototype displayed run-time performance issues that must be overcome before it can be used in practice; several possible solutions are identified.
Declaration of Authorship

I, Matt Ryan Selway, declare that:

• this thesis presents work carried out by myself and does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university;

• to the best of my knowledge it does not contain any materials previously published or written by another person except where due reference is made in the text;

• and all substantive contributions by others to the work presented, including jointly authored publications, is clearly acknowledged.

Signed: 

Date: 04/02/2016
Acknowledgements

I would like to thank my supervisors, Prof. Markus Stumptner and Dr. Wolfgang Mayer, for their support and guidance during my PhD candidature. I am especially grateful to Prof. Markus Stumptner for providing the opportunity to be a member of the Knowledge and Software Engineering Laboratory. My time in the lab has been thoroughly enjoyable with interesting projects and great personalities. In addition, thanks go to the other members of the lab for helping to make the experience as enjoyable as it was.

I also acknowledge the University of South Australia for supporting me through a Vice Chancellor and President’s Scholarship in addition to providing the facilities with which I could pursue my research. Moreover, a special thank you goes to the administrative staff of the School of Information Technology & Mathematical Sciences who handled all my petty requests and form submissions with a smile and aplomb.

I would like to thank my family for their constant and ongoing support. In particular, I am grateful to my brother who was very understanding while I was completing the write-up of this thesis.

Last of all, but certainly not least important, my gratitude goes to the girls at the local Caffé who kept me caffeinated and sane throughout my candidature.

A big “thank you” to you all,

Thank you!
Preface

This thesis presents research undertaken at the Advanced Computing Research Centre, University of South Australia as a member of the Knowledge and Software Engineering Laboratory under the supervision of Prof. Markus Stumptner and Dr. Wolfgang Mayer. The content of this thesis is based on and extends the work presented in the following peer-reviewed research papers published during my candidature:

  In press


Chapter 1

Introduction

Historically, Software Engineering research has heavily focused on improving the development and delivery of software projects. In particular, an emphasis is placed on key factors such as: time-to-market, cost, software quality, correctness, and requirements traceability. Over the years various methods of improving these factors have been considered, with a common theme being an increase in the level of abstraction used for the different artefacts of the software development process (e.g. specifications, designs, and implementations). Increasing the level of abstraction reduces the cognitive gap, thereby, reducing errors, speeding up the development process, and reducing costs.

One of the main areas where the level of abstraction has been increased over time is program code. There has been a shift from bits, to assembly languages (where shorter, easier to remember codes were given to identify operations), to higher-level procedural programming (where step-by-step instructions are given using data structures), to object-oriented programming (where the procedures are encapsulated as the behaviour of the data structures, i.e. objects), to model-based programming (where more abstract modelling of behaviour and objects occurs). This has resulted in an approach to software development known as Model-Driven Engineering (MDE\textsuperscript{1}), which has undergone much research in the last decade.

The unifying principle, as suggested by \cite{Bez05}, is that ‘everything is a model’ \cite{Bez05}, analogous to the object-oriented principle that ‘everything is an object.’ Models (usually

\textsuperscript{1}The term Model-Driven Development is often used synonymously, but should really be considered a subset \cite{BCW12}. We will refer only to MDE as it encompasses both either way.
object-oriented models) have been used to support the analysis and design of software for a long time—e.g. the Object Modelling Technique [RBPE+91], predecessor to the current de facto standard for object-oriented modelling (UML), has existed since the early 90’s. However, MDE promotes models to the status of the primary artefacts of the software development process: to the extent that developers are said to model, not code. Furthermore, the theme of increasing the level of abstraction is present within MDE itself, as it encourages the creation of models at higher-levels of abstraction and their refinement (ideally through automated transformations, which can be represented as models themselves) into more concrete, lower-level models. Such an approach is demonstrated in the Object Management Group’s Model-Driven Architecture (MDA) [MM03], which formalises three levels of abstraction.

Despite the increase in the level of abstraction and other improvements to implementation methods, software development has been plagued by spectacular failures of large, expensive projects [Bro11; Con14]. Problems with requirements, such as badly specified requirements or misunderstandings of requirements, are often cited causes of IT project failures are [Fir07; AAKA+09; AS09; Lin08; MYS12; NAT96; NFSR+14; HL01; ROW13]. Since the requirements form the basis of all the activities of the software development process, ill-defined and unrecorded requirements lead to incorrect decisions during the design, implementation, and testing phases of the development process. This has led to research in Software Engineering specifically aimed at the improvement of requirements resulting in the field of Requirements Engineering (RE).

Requirements Engineering research focuses on the improvement of the requirements phase of software development including the elicitation, specification, and formalisation of requirements [Lau02; SS97]. With the advent of MDE, Requirements Engineering can be tightly integrated with the remainder of the software development processes through formally modelling the requirements at a higher-level—in contrast to formal specification languages such as Z [ISO02] which are highly technical, are difficult to learn, and cannot be understood by non-technical experts [FKV91]. As a result, the software development process can be improved through automated analysis and through greater traceability using mappings, transformations, and refinements of the high-level requirements.

Although formalising and modelling requirements can improve the software development
process, the primary form of a requirements specification is still a document written in natural language, such as plain English [YBL11; MBM13]. The formalisation of these requirements is a slow, difficult, and error-prone manual process due to the inherent ambiguities of the language. To address this, numerous approaches have been developed; these range from developing methodologies for the manual process to fully automated approaches that make use of Natural Language Processing (NLP) techniques. By formalising the manual process, the level of errors can be reduced; however, such approaches remain time-consuming. On the other hand, the automated approaches save time in developing the initial models, but typically produce incomplete and erroneous models that require manual revision and refinement [MBM13].

While it has long been known that the customer is an integral part of the requirements process as the domain expert [SD00], recent work by the Object Management Group in standardising the Semantics of Business Vocabulary and Business Rules (SBVR) [OMG08b] has provided a push to put control of specifications into the hands of the domain experts themselves, rather than the technically minded developer. This is particularly important for the business domain, where the rules (or requirements) of a business can change rapidly due to internal process improvement or policy change, and outside factors such as legislative changes [HH00; BM11]. In both cases, the software developers/maintainers are not necessarily in a position to keep pace with such rapid change and, therefore, the business people (as the domain experts of their organisation) need to be able to create and maintain the requirements themselves.

While numerous tools are available to support the capture and modelling of requirements, including a number that utilise an SBVR-based notation, they are intended to be used by technical experts. The typical process is for the business/domain expert to develop an informal description in plain natural language and provide that to the technical experts—along with the outputs of interviews and other requirements elicitation techniques employed by the analysts [Lau02]. The technical expert then reads the specification and other documentation, analyses it to identify rules and, if using a formalisation tool, translates them into a formal language that the tool then translates into a formal model. The technical expert may then provide the resulting specification to the business/domain experts to have them validate that it is what they wanted. The benefit of using a controlled language in this process is that the domain experts
can more easily read the resulting specification, even if they would not have been able to write it.

This method inhibits the development of formal specifications by business/domain experts themselves and, therefore, restricts the creation and maintenance of the formal business specification to the technical experts.

In contrast, bringing the creation and maintenance of the formal business specification under the control of non-technical experts will mean a shift in the relationship between the two parties. Rather than providing informal specifications, business people will be able to provide formal, validated, models of their requirements that are immediately usable by technical experts. If additional requirements are elicited by the technical experts, the changes will easily be integrated by a business/slash domain expert and the updated formal model returned to the technical experts.

To date, however, there has been insufficient research into the translation of natural language requirements into formal models that reduces both the errors in the output and the time to perform the translation, in addition to being suitable for the shift of control from technical experts to domain experts. Both the manual and automated processes tend to be aimed at technical experts and, as such, use technical notations that are difficult for domain experts to use or validate. For example, manual or semi-automatic approaches typically require the natural language requirements be rewritten in a (semi-)technical notation, while fully automated approaches that do not require the specifications to be rewritten typically produce models that need to be manually revised and validated in a notation unfamiliar to domain experts.

In summary, it would be interesting to see if we could develop an approach to formalising requirements that is suitable for both domain and technical experts, and that addresses the seemingly conflicting concerns of reducing both the time to formalise and the number of errors in the specification.
1.1 Aim, Contributions, and Scope

The aim of this research is to investigate a method for the (semi-)automatic translation of natural language requirements into formal models that: (1) is suitable for use by domain experts, not only technical experts; (2) supports the user in performing the formalisation to reduce errors, ambiguity, inconsistencies, etc.; and (3) reduces the amount of time and manual effort involved. While we aim to achieve all three goals, some trade-offs may be necessary as the process is unlikely to be completely automated. For example, creating a well defined vocabulary (even when using natural language) takes more effort than would normally be used for a simple glossary of terms. However, we see this additional effort as positive since it is adding information to better formalise the specification, in contrast to expending a large amount of effort simply rewriting an entire specification into another notation. Moreover, the net effect should be a reduction of effort.

To achieve the stated aim, this thesis investigates the following questions:

1. What are the current approaches to (semi-)automatically transforming natural language (business) specifications into formal models and how suitable are they for use by (business) domain experts?

And, assuming current approaches are not suitable:

2. How can we better support business/domain experts to create formal models from natural language specifications?

3. How can we reduce both the errors and the time taken to perform such formalisation?

In order to answer these questions, we first perform a review of existing approaches to automatically creating formal models from specifications and investigate their suitability for use by business/domain experts. This is performed by identifying a number of relevant criteria and assessing the capacity of each approach to fulfil those criteria. The hypothesis is that no existing approach is entirely suitable for use by domain experts and, therefore, a novel approach must be developed.
We then propose a general, knowledge-based framework for the definition, processing, and understanding of Controlled Natural Languages (CNLs). This novel framework combines a theory from the field of Cognitive Linguistics with Knowledge-based Configuration to derive an understanding of the text, with respect to some target semantics. While it results in a framework for CNLs, it is designed to support relatively unrestricted CNL compared to others and to provide high-quality feedback that can assist users in refining their specifications to conform to the syntax and, more importantly, the semantics of the CNL thereby reducing or eliminating errors, inconsistencies, and ambiguities.

The general framework is then applied to natural language specifications using the SBVR standard as the target semantics. Using SBVR, the resulting CNL supports language suitable for business experts, while providing feedback and assistance to revise and refine natural language specifications. The applied framework is then extended with methods for acquiring the vocabulary from a glossary and the specification itself to help reduce the time and effort it takes a business expert to produce a formalised specification.

A performance evaluation of the SBVR-based implementation is then performed with respect to an SBVR-based case study. In addition, a comparative evaluation of the vocabulary acquisition from unrestricted text is performed against approaches for extracting analysis models from natural language specifications. The comparative analysis is performed on non-CNLS texts that are common to the evaluation of several works in the literature. The two aspects allow two different scenarios to be considered: (1) the situation where a new specification is being developed in accordance to the SBVR guidelines; and (2) the situation where a requirements specification (or other documentation) already exists but needs to be formalised. Since the application is based on SBVR semantics and the SBVR controlled language, it is expected that the former will show good performance in creating the formal model from the specification text, while the latter will require a larger number of feedback iterations to arrive at a suitably formalised specification and its model.

The contributions of this work are as follows:

- A general knowledge-based framework for defining, processing, and understanding CNLs. This novel framework is designed to support relatively unrestricted CNLs and
Section 1.1. *Aim, Contributions, and Scope*

to provide high-quality feedback that assists the user in refining the text so that it is acceptable to both the syntax and semantics of the CNL.

- An application/implementation of the general framework to the formalisation of natural language requirements (e.g. business requirements and software requirements) in an *MDE* environment.

- A method for acquiring a lexicon from a glossary accompanying a specification.

- A method for acquiring a candidate lexicon from dependency parses of unrestricted natural language specifications.

There exist some limits to the research that should be noted from the outset. The focus of the research is on the natural language to formal model translation/transformation. Once the requirements are represented in a formal model they can undergo additional transformations into other types of models, such as Unified Modelling Language (UML) or Business Process models as exemplified by [KAB09b; SNPS12; SPI10; RPH08; TC06]. Since many transformations already exist in the *MDE* literature, and since individual transformations between different types of models can have their own complexities, they are left for other work.

As such, the contributions made in this thesis do not constitute a complete process or tool for Business Modelling, Software Engineering, or Requirements Engineering. Rather, they constitute a prototype component that requires further development before being integrated into such tools and processes. A complete tool could include many more advanced features, such as the automatic generation of graphical models from the translation, view generation, revision management, etc.

Finally, due to time constraints and the incomplete nature of the prototype, the approach presented in this thesis has not undergone user evaluation. Instead the benefits of reduced manual effort, time, and errors are based on the assumption that the automation (by definition) will reduce manual labour and time, while errors will be reduced by providing user guidance for conflicts that cannot automatically be resolved.
1.2 Overview of the Thesis

The remainder of this thesis is structured into 7 chapters. An overview of the relationships between the them is shown in Figure 1.1.

![Figure 1.1: Relationships between Chapters of the Thesis](image)

Chapters 2 and 3 investigate literature related to this thesis. In Chapter 2, we introduce the basic foundations of this work, including: Model-Driven Engineering (MDE), Requirements Engineering (RE), Natural Language Processing (NLP), and Knowledge-based Configuration. Afterwards, Chapter 3 identifies previous approaches to formalising or otherwise creating models (controlled) natural language specifications. This includes a critical review, comparison, and categorisation of the numerous approaches and their underlying language processing techniques. The review covers a wide range of approaches over which a number of criteria are developed to differentiate and evaluate them with respect to our aim. As a result, gaps in the literature are identified and appropriate research questions formed.

Chapters 4 to 6 constitute the core of the thesis in which we present a novel approach to translating (controlled) natural language requirements into formal models and the results of
Section 1.2. Overview of the Thesis

applying the it to several case studies. First, a general framework for the knowledge-based processing of controlled languages is developed in Chapter 4. This is followed by an implementation of the general framework in Chapter 5. The described implementation makes use of SBVR to fulfil the identified criteria and achieve the aim of this thesis. Chapter 6 presents the results of applying the prototype tool to a controlled natural language (business) specification as well as a comparative evaluation of the vocabulary acquisition from unrestricted text component.

In Chapter 7, the results are discussed and conclusions presented. In particular, the criteria developed in Chapter 3 are applied to our new approach to illustrate where it fits with respect to the surveyed approaches. Finally, Chapter 7 concludes this dissertation with a discussion of future research directions.
Chapter 2

Background

The topic of this thesis transits a wide variety of fields. As we have mentioned in the introduction, this research is heavily based in areas such Model-Driven Engineering (MDE), Requirements Engineering (RE), and Natural Language Processing (NLP). In addition, it utilises the process of Configuration (specifically Knowledge-Based Configuration) to achieve its goal. This chapter briefly presents each of these fields in turn to provide the foundations on which the remainder of the thesis is built. Furthermore, it clarifies the motivation of the thesis by identifying the shortcomings of each field towards the aim of the thesis.

Structurally, it consists of five distinct sections, one for each field plus a summary, that inform the decisions discussed in the remainder of the thesis. In the first section, Model-Driven Engineering (MDE) is introduced. Next, the Requirements Engineering (RE) process and its importance in the software development process is discussed. Natural Language Processing (NLP) is then introduced and a number of different approaches, their benefits and disadvantages are discussed. This is followed by a section detailing Knowledge-Based Configuration and discussion of its application to NLP. In the final section, the links between the four fields are summarised.

2.1 Model-Driven Engineering

Model-Driven Engineering (MDE) is an approach to Software Engineering that promotes models—abstracted representations of a system—from a documentation role to the primary
artefacts of the software development process. It does so based on the principle of “everything is a model” [Bez05], treating models as the unifying concept, thereby supporting the uniform treatment of many aspects of MDE. The aim of MDE is to raise the level of abstraction of software development with the goal of gaining a number of benefits, including: (1) improved separation of concerns, (2) increased formalisation, (3) reduced complexity, (4) improved software quality, (5) increased productivity, (6) reduced time-to-market, and (7) improved communication between developers (and other stakeholders) [Bez05; HT06; BCW12]. In the following we briefly describe the main concepts of MDE and introduce some of its challenges.

2.1.1 Models

There are four main concepts of MDE: terminal models, meta-models, meta-meta-models, and model transformations. Terminal models are representations of a system under consideration; they capture some aspects of the system (i.e. they abstract from it) and are used to provide knowledge about it [Bez05; KDA10]. Typically the modelling language of a terminal model is provided by a meta-model; it is said that the terminal model conforms to the meta-model. Similarly, the meta-meta-model provides the language for defining meta-models; however, it also conforms to itself. Together, terminal models, meta-models, and meta-meta-models form a three-level architecture—with levels named M1, M2, and M3, respectively—with a fourth level (M0) for the system that is represented. This is shown in Figure 2.1 with some examples in different technical spaces.

Finally, since models provide an abstracted view of a system they exclude some details of it. Therefore, in MDE multiple models (e.g. structural and behavioural models) are typically required to provide a complete representation of a system.

2.1.2 Model Transformations

The last, and arguably the most important, concept of MDE is the concept of a model transformation: that is, the process of producing one or more output models from one or

1 Otherwise known as instance models, concrete model or simply models. For the purpose of this discussion terminal model will be used, while model by itself will be used more generally to refer to any of terminal models, meta-models, and meta-meta-models.
more input models [SK03]. The model transformation process is shown in Figure 2.2; note that model transformations themselves are considered models that conform to a model transformation language (meta-model).

There are several tasks for which model transformations are intended to be applied, including [SK03; CH06; BCW12]:

- Refinement of higher-level models into lower-level models and code (forward engineering)
- Reverse engineering higher-level models from lower-level models
- Synchronisation between models at the same or different levels
- Application of software patterns and refactoring for model evolution
- Creation of different views of a system to demonstrate a particular concern of the system
- Interoperability between systems
Automating these tasks maximises the benefits of Model Driven Engineering (MDE) by allowing the software engineer to develop the system at the higher-level of abstraction. Model transformations can be automated in a variety of ways, e.g. through direct manipulation with a general purpose programming language [SK03; CH06]. However, the most appropriate approach is to utilise a Domain Specific Language (DSL) designed for the manipulation of models, such as [ATL JK06], QVT [OMG08a], ETL [KPP08], or one of the many other transformation languages that exist.

Using a model transformation DSL, a developer specifies mappings (or transformation rules) between models that a transformation engine can then execute. Using multiple transformation definitions between the same or different meta-models allows a sequence of transformations to be automated: e.g., a refactoring transformation, followed by refinement, and then code generation.

Finally, an important distinction is made between model-to-model transformations and model-to-text transformations. Model-to-model transformations are typically used for the model refinement and refactoring tasks, while model-to-text transformations are used for code generation from low-level models [CH03]. This is mainly due to the pragmatic reason of reusing existing compilers.
2.1.3 The Model-Driven Architecture

To a certain degree, the MDA [MM03], developed by the Object Management Group, standardises the core concepts and processes of MDE around a set of Object Management Group standards. Moreover, the MDA specifies three levels of abstraction that separate the high-level specification of a system from the implementation details of that system, which are the: (1) Business Model or Computation Independent Model (CIM) level, which models the environment and requirements of a system without regard to how the system is implemented; (2) Platform Independent Model (PIM) level, which models the functionality of the system independent of any particular platform; and (3) Platform Specific Model (PSM) level, which models the implementation with respect to a particular platform. This framework is illustrated in Figure 2.3 including a number of the Object Management Group standards that are intended to be used within the framework.

The main standards developed by the Object Management Group to be used in the MDA include: SBVR [OMG08b], MOF [OMG06], UML [OMG09a, OMG09b], OCL [OMG10], and QVT [OMG08a]. The Meta Object Facility (MOF) standard specifies a meta-meta-model for the definition of meta-models in the MDA framework. MOF provides the unifying language for the entire group of MDA standards.
The UML standard provides a general purpose modelling language suitable for defining a number of software artefacts such as Class Diagrams, Activity Diagrams, and Sequence Diagrams. The artefacts defined by UML are typically at the PIM and PSM layers of the MDA framework. The difference between UML models at the PIM and PSM levels is that the PSMs include platform specific details, e.g. the use of a particular code library to realise certain functionality.

UML itself does not provide all the necessary constructs for defining business rules and constraints at the PIM and PSM layers. However, combining it with the Object Constraint Language (OCL) allows UML model elements to be annotated with constraints. For example, OCL constraints can be used to define preconditions and post-conditions on operations/methods, additional relations that hold between elements of the model, definitions of derived attributes, and even method/operation definitions (for side effect-free methods only).

The model transformation capabilities of the MDA framework are provided by the Query/View/Transformation (QVT) standard. It defines a group of related model transformation languages for model-to-model transformations. These languages—QVT Relations, Core, and Operational Mappings—allow specification of the transformations using a declarative (Relations and Core), imperative (Operational Mappings), or mixed syntax.

Finally, the SBVR standard is designed to bridge the gap between business analysts and software developers. It allows the description of a business model in a controlled natural language, SBVR Structured English (SBVR SE), which uses text stylisation markup to identify different aspects of the defined vocabulary and rules. In addition, the standard provides a MOF meta-model that acts as the formal basis of the specification to facilitate the transferal of the natural language business model to the software developers with a minimum of inconsistencies.

Business models, such as those defined in SBVR, exist at the CIM layer as they model the business domain, rules, activities, and policies, but not the system itself. Model transformations, preferably automated, are performed from the models of the CIM layer to the PIM layer and further to the PSM layer, which we can see in Figure 2.3. Ideally, business/domain

---

2Due to its general nature, UML can also be used to define a business model at the CIM layer, which is the case in conceptual modelling where a conceptual schema or domain model is defined to capture the knowledge required by the system [CPR09; KAB09a].
experts should control the CIM level models, while technical experts control the lower levels and the transformations between them. Indeed, the MDA specification agrees, stating that, ‘It is assumed that the primary user of the CIM, the domain practitioner, is not knowledgeable about the models or artifacts used to realize the functionality for which the requirements are articulated in the CIM.’ [MM03]

2.1.4 Challenges in MDE

While MDE shows a lot of promise, it is not without its challenges. Of prime importance to the aim of this thesis is the current focus of MDE at its lower levels, such as code generation [MD08; HWRK11; LAI11]. Indeed the MDA although it discusses CIMs describes model-to-model transformations entirely between PIMs and PSMs [MM03]. Therefore, not only are manual model transformations required from natural languages specifications into the highest level models, but manual transformations are required to refine them into lower-levels of abstraction. Although we focus on the former problem in this thesis, we believe that maximising the automation across all levels of abstraction will gain the most benefit from MDE.

2.2 Requirements Engineering

The transformation of business and requirements specifications into models falls in the domain of Requirements Engineering (RE), which focuses on improving the requirements phase of software development including the elicitation, specification, and formalisation of requirements [Lau02; SS97]. These tasks are usually performed during the elicitation—the discovery of requirements through stakeholder meetings, interviews, existing documentation etc.—and analysis—including analysis of the relevant domain objects/concepts and requirements—phases of software development [Lar05; Som11]. The outputs of RE form an important communication tool and typically consist of a requirements document in natural language and possibly a set of analysis models, which should be at the CIM level—commonly referred to

---

3 We say ‘should’ since RE done poorly can lead to the insertion of technical considerations into these high-level models, which is a common problem [Fir07].
as the Problem Domain, as opposed to the Solution Domain PIM level, in RE. These models are then revised and refined into PIMs during design.

Traditional RE approaches—i.e. those provided in textbooks and those embodied in more general Software Engineering processes such as the Rational Unified Process—take a very manual approach to the development of requirements specifications and analysis models and their transformation to PIMs. These manual transformations are time-consuming and error prone, which could be improved by the integration of MDE into the process. To do so, analysis models require a greater level of formalisation to better support the automated transformation to PIMs as much as possible. Model-Driven Requirements Engineering (MDRE) processes such as OpenUP/MDRE focus on the transformation from CIMs to PIMs. In this way the benefits of MDE can be applied to the earliest phases of software development. However, such approaches are few and far between.

Not only are RE processes that are integrated with MDE limited in number, they do not address the automation of the transformation from natural language specifications into the initial analysis models. Although more and more methods are being developed to perform automated analysis of requirements documentation, comparatively few focus on generating formal models. For example, in a survey carried out in 2013, only one-third (12 of 36) of the approaches fell into the ‘Requirements Model Generation’ category. A survey of commercial RE tools found that they were lacking in modelling features. Moreover, when the generation of models is considered, it is not generally automated. For example, in a survey of approaches to creating analysis models only 7 of the 16 identified were automated (a further 2 were semi-automated). In the 2013 study, of the 12 approaches in the model generation category, only 5 supported full automation, with the other 7 semi-automated. The limited modelling capabilities of commercial RE tools in particular suggests that MDRE (along with MDE itself) is not common in practice.

Finally, despite the importance of involving the users and business/domain experts in the RE process being well-known, their involvement is often relegated to initial informal input and final validation. Even using the existing tools, the requirements are developed by Requirements Engineers or Business Analysts and, as a consequence, the requirements must go through many iterations of elicitation, analysis, and validation. This results in a lot of
back-and-forth between customers and technical experts, with misunderstandings a common occurrence due to the imprecise and ambiguous nature of natural language. Considering a large number of software developers would like to more efficiently identify user requirements [MBM13], it would be ideal if the business/domain experts could specify and validate their requirements themselves—at least to a certain degree—in a formal form that maximises communication and minimises misunderstandings. However, this clashes somewhat with the formal nature of the analysis models required for the integration with MDE since business/domain experts are unfamiliar with the formal notations used by the software developers. Therefore, automated transformations between natural language requirements documentation and the formal models must be investigated.

2.3 Natural Language Processing

To create formal models from natural language specifications requires techniques from the field of Natural Language Processing (NLP). This field aims to make computers perform useful tasks with human language such as dialogue systems, language translation, Information Retrieval, and query interfaces to structured data (e.g. databases) or unstructured data (i.e. textual documents) [JM09]. To perform these tasks, a large number of NLP components, techniques, and tools have been developed which rely on several types of knowledge [All95, JM08]:

- **phonological and phonetic**: knowledge considers how words are related to their sounds; this is relevant to speech processing
- **morphological**: knowledge refers to the meaningful elements of words, called morphemes
- **syntactic**: knowledge regards how sentences are formed and structural relationships between words
- **semantic**: knowledge considers the meanings of words and sentences in a context independent manner, e.g. as a basic logical form

---

4We believe traditional Requirements Engineers will still be necessary to help elicit additional requirements that have potentially been left implicit, or that the business/domain experts are not aware of.
**pragmatic**: knowledge concerns the meaning of sentences in the context of situations and the goals of the speaker

**discourse**: knowledge regards the effect preceding sentences have on those that follow

**world**: knowledge considers the general knowledge that a speaker may have, including that of other people’s beliefs and goals

In order to perform deep Natural Language Understanding (NLU), a system needs to successfully integrate all of these forms of knowledge. However, many useful applications have been developed by relying on only some of them. We can differentiate the approaches that use different levels of knowledge through three broad categories [Zou11]: (1) shallow syntactic, (2) shallow semantic, and (3) deep semantic. Shallow syntactic approaches only consider syntactic (and possibly morphological) knowledge to provide partial (e.g. as in the case of chunking) or complete syntactic analyses of texts (e.g. those output by constituent and/or dependency parsers). Shallow semantic approaches take it a little further by integrating some context independent semantic knowledge through the use of such things as semantic/thematic roles [MCLS08], discourse representation structures [KR93], and frames [DCMS+14]. In contrast, deep semantic analysis utilises much more of the knowledge hierarchy—including pragmatic, discourse, and even world knowledge—to perform complex language tasks such as context dependent anaphora (or co-reference) resolution (i.e. linking elements between sentences that refer to the same entity).

A large focus of NLP is in resolving the many different types of ambiguity that occur in natural language at all levels [Ali95]. For example, noun-verb ambiguities (in part-of-speech (POS)-tagging and syntactic analysis), quantification scope ambiguities in semantic analysis, and co-reference ambiguities in pragmatic analysis. Modern methods for solving these problems, particularly for shallow approaches, make extensive use of statistical/probabilistic methods; the rise of which has been supported by the increased availability of large corpora and computer processing power over the last 20 years [JM08]. While this approach has been very successful, it is inadequate to achieve the aim of this thesis: that is, creating precise, unambiguous formal models from natural language specifications.
Unfortunately general NLP and NLU systems are too imprecise, achieving variable accuracies depending on the task (e.g. POS tagging achieves up to 97%, while syntactic dependency parsing achieves up to 92%) [DCMS+14; PDG12; Bos08; CM14; ASB08; ACL14]. Part of the issue is the propagation of errors from earlier subtasks (e.g. POS tagging) through to subsequent stages of analysis (e.g. syntactic and semantic analysis) [NVS12]. In addition, in the presence of ambiguity they utilise the statistics and heuristics to determine the expected interpretation; however, to formalise specifications we need a precise approach that can detect ambiguity, but rather than “guessing” the intended interpretation, assists the user in eliminating the ambiguity. One way of reducing or removing ambiguity from natural language is to utilise a Controlled Natural Language.

### 2.3.1 Controlled Natural Languages

Controlled Natural Languages (CNLs) or simply controlled languages, have had many names and definitions, often restricted to specific purposes [Kuh14]. In this thesis we adopt the definition of CNL proposed by Kuhn [Kuh14], that a CNL:

- is based on a natural language (such as English),
- is more restrictive than its base language in terms of lexicon, syntax and/or semantics,
- preserves most of the natural properties of the base language, and
- is a constructed language, i.e., it has been explicitly defined rather than a result of an implicit natural process.

Languages that fulfil these conditions have been defined for many different purposes, such as: communication between humans, automated translation between languages, and as simplified representations of formal languages [Kuh14]. Moreover, such languages can be classified as taking a naturalist (or human-oriented) approach or a formalist (or computer-oriented) approach [CMHT10; Hui98]. Naturalist approaches focus on human comprehensibility and communication. In contrast, formalist approaches attempt to simplify a formal language to make it more accessible to non-experts by giving it a natural language based notation. In
general, formalist approaches are strictly defined using formal grammars, while naturalist approaches are often defined by sets of restrictions to the base language and guidelines on how to use the controlled language.

Kuhn [Kuh14] presents a survey and classification of 100 English-based CNLs from 1930–2013. Many of these languages were created for very specific purposes and/or organisations. Moreover, while controlled languages can reduce ambiguity to varying degrees, they are often more difficult to write (i.e. the writability problem [Kuh10, PSE98, SLH03]) or may not be expressive enough to capture what is needed for a specific application (when considering reusing an existing controlled language). This is a potential problem for our aim of putting the control of formal specifications into the hands of business and domain experts. Therefore, we searched for inspiration and alternative approaches to processing natural language from related fields.

### 2.3.2 Cognitive Linguistics

The field of Cognitive Linguistics is a framework of disparate theories, unified by one basic principle and four characteristics [Gee06]:

- P Language is about meaning
  - C1 Meaning is based on perspective
  - C2 Meaning is dynamic and flexible
  - C3 Meaning is encyclopedic and non-autonomous
  - C4 Meaning is based on usage and experience

As such, Cognitive Linguistic theories typically cover most (if not all) of the types of knowledge required in NLP and often encompass methods for the representation and processing of concepts and knowledge. Of interest to our work is the emphasis on meaning that Cognitive Linguistics has, which we believe to be important in the analysis of natural language. In particular, one theory in Cognitive Linguistics appears to embody this principle really well: Cognitive Grammar [Lan08].

22
2.3.2.1 Cognitive Grammar

Cognitive Grammar is a theory of grammar that blends morphology, syntax, semantics, and pragmatics into a holistic view of language. The basis of Cognitive Grammar, described in \[\text{Lan08}\], is that grammar (i.e. the structure of language) has meaning. This is in contrast to traditional theories which relegate grammar to syntax and separate it from semantics. Furthermore, Cognitive Grammar defines grammar as symbolic—i.e. each grammatical element is a pairing between a semantic structure and a phonological structure—in the same way as basic lexical units. Therefore, symbols on their own are enough to characterise complex expressions, thereby allowing a uniform processing of all elements of language \[\text{Ste04}\].

In lexical terms, Cognitive Grammar advocates a lexicon defined as ‘the set of fixed expressions in a language’ \[\text{Lan08}\]. Therefore, learning a language is the learning of a large number of expressions, including multi-word expressions, that represent the conventional ways of conveying concepts. This is similar to the memory-based learning methods described in \[\text{DV99}\]; however, Cognitive Grammar also advocates the varying schematicity of expressions in the lexicon, moving reasoning beyond similarity against previous experiences to abstraction over those experiences. As a result, the lexicon encompasses what is traditionally considered grammar. This view of lexicon is potentially useful in allowing non-technical users to define their domain terminology in a natural way.

Looking up a lexical entry through its phonological structure (i.e. the sound or text) evokes, or activates, its associated semantic structure \[\text{Lan08}\]. However, the same phonological structure may activate multiple semantic structures. Resolving this ambiguity, roughly equivalent to word-sense disambiguation, occurs as additional semantic structures are evoked and combined during the processing of an expression. Figure 2.4 illustrates some abstract symbolic structures at varying levels of composition.

In knowledge representation terms, Cognitive Grammar utilises semantic structures, representing concepts, that contain a set of domains called its matrix. A domain is a ‘... realm of experience’ \[\text{Lan08}\] that can be basic or nonbasic\(^5\) and consists of one or more dimensions. Examples of basic domains include time, space and colour, while nonbasic domains are

\(^5\)In previous work Langacker referred to nonbasic domains as abstract domains \[\text{Lan08}\]
Figure 2.4: Symbolic structures in Cognitive Grammar at varying levels of composition. The phonological poles are indicated by $P$ and the semantic poles by $S$.

Built on top of them. The domains in a matrix have different degrees of centrality, i.e. some domains are more salient than others; however, the salience of domains changes based on context, which profiles a certain domain or aspect of the semantic structure [Lan08]. The profile of a semantic structure is simply the most salient aspect of it. Relations can also have other highly salient aspects, referred to as the trajector—usually the most salient participant of the relation—and the landmark—another highly salient aspect of the semantic structure that provides a reference point for the trajector.

Semantic structures are also linked to additional structures that fall under their scope; called its parts [Lan08]. For example, an ATM may have a display, keypad, and cash dispenser as parts. Parts of relationship structures that are explicitly involved in the composition processes are termed elaboration sites (or just sites for simplicity). Furthermore, in [Hol93], the converse of parts, i.e. wholes, is distinguished so as to utilise full semantic structures rather than nonbasic domains for the purpose of scoping. For example, an ATM may have ‘banking’ as a whole, rather than creating a ‘banking’ domain and including it in the matrix of ATM. An example semantic structure using the notation of [Hol93] is displayed in Figure 2.5. Briefly, in the notation ‘Dn:cn’ represent domains of a semantic structure and their centrality; the ‘predications’ are categories of the domains, while ‘dn’ and ‘vn’ indicate the different dimensions of a domain and their values, respectively; and ‘sn’ indicates that the part is an elaboration site.

When processing an expression in Cognitive Grammar, the meaning is arrived at through the composition of the evoked semantic structures into a single coherent structure [Lan08]. Correspondences are found between the domains, parts, wholes and sites of the structures being composed, which are then superimposed to form a new semantic structure for the
conception. If correspondences cannot be found and the semantic structures cannot be superimposed, then the expression is ungrammatical or otherwise not understandable. An example of the superimposition process for the fragment ‘ATM requests password’ is displayed in Figure 2.6—at the top is the composite structure, with the individual structures below, and the dotted lines indicate the correspondences.
The unified approach of Cognitive Grammar has several desirable properties in the context of this thesis. The emergence of traditional syntactic elements (e.g. the POS categories and grammar rules) from meaning raises the possibility of making semantic analysis the primary aspect of the process, with (pure) syntactic analysis reduced to word-order information evoking possible meanings.\(^6\) As a result, a Cognitive Grammar-based language processing approach could take traditional syntax into account (directly or indirectly) within the semantic analysis process in parallel with other disambiguation tasks. This could prevent errors propagating from POS-tagging, for example, throughout further analysis. In addition, the usage based lexicon presents the possibility of simplifying the definition of vocabulary for domain experts.

The challenge with Cognitive Grammar, however, is that it is a complex descriptive theory that is difficult to implement [Hol93; Hei94; Ste04]. However, there have been a few attempts at developing computational models of Cognitive Grammar to start from [Hol93; Hei94; Ste04]. Of particular import is the work of Holmqvist [Hol93], which is the most faithful to the theory on which it is based and is not specific to a cognitive architecture (e.g. the work of [Ste04] for ACT-R). As a result it preserves the desirable properties of Cognitive Grammar, unlike L-spaces [Hei94], which makes use of a complex technical specification of lexical entries. While the processes described by Holmqvist [Hol93] are still very complex and lexical acquisition (along with it the associated semantic structures) will be a challenge for unrestricted natural language processing, this thesis considers leveraging some aspects of Cognitive Grammar and Holmqvist’s computational model to achieve its aim.

### 2.4 Knowledge-based Configuration

Knowledge-based Configuration is a constraint-based search method traditionally used to compose a customised system from generic components [Jun06]; however, it has also been used in a various domains such as software services, software product lines, and constraint-based language parsing [HW13]. There a number of approaches to configuration such as rule-, structure-, resource-, and connection-based, as well as component-oriented, which combines the advantages of many of the others [STMS98; Stu97]. While there a many

---

\(^6\)This does not necessarily mean that syntax and word-order has no meaning, only that it is at the least semantic end of the continuum.
formalisms for configuration, they each require two fundamental elements: (1) a description of the problem and its constraints, i.e. the configuration model; and (2) user preferences and constraints on the desired configuration, i.e. the configuration goal [Jun06]. With this information defined, a configurator can produce one or more configurations that satisfy the constraints, or a description of why one cannot be found.

The general flexibility of configuration and its successful application in varied domains suggests its usefulness for this work. Two applications in particular stand out. The first is its application to MDE for which model search is defined as an advanced model transformation technique [KDA10]. In this approach, rather than specifying deterministic mapping rules using a model transformation language like ATL, configuration is used to search for a solution. To do so, two versions of a meta-model are defined: a relaxed meta-model, and the constrained meta-model. The relaxed meta-model serves as the source meta-model of the transformation and is simply a version of the original meta-model in which the constraints have been relaxed: e.g., minimum cardinalities are reduced to zero and predicates are removed. The constrained meta-model, which is the original meta-model with all of its constraints intact, is used as the target of the model transformation.

The model transformation uses the configurator to search for a terminal model that satisfies the constraints of the constrained meta-model based on an input, relaxed terminal model. In this situation the description of the problem and its constraints is provided by the meta-model(s), while the configuration goal is provided by a partial configuration that is the relaxed terminal model used as input. This transformation method is useful in situations where deterministically generating target models is inadequate and for bringing traditional configuration tasks in the MDE domain.

The second application area of import is the use of configuration for language processing, for which it has been shown to be a flexible, non-deterministic approach for parsing natural language [EH04; KAB09b]. For example, Estratat and Henocque [EH04] translated Property Grammars (a constraint-based linguistic formalism) into configuration models to process a simple context-free grammar ($a^n b^p$) and a subset of French. Moreover, Kleiner et al. [KAB09b] adapted the approach to English for the purpose of analysing SBVR-based texts in the MDE context. In addition, Duchier et al. have translated Dependency Grammars [Duc99]...
Chapter 2. Background

and later Property Grammars [DDPL12] into standard constraint-satisfaction problems (which are also a formalism used for configuration) to process natural language.

The limitation of all these approaches has been their focus on traditional\[7\] syntactic analysis. Although Estratat and Henocque [EH04] suggest that semantics can be integrated into the configuration model, none were incorporated. Similarly, Kleiner et al. [KAB09b] add some SBVR elements into the syntax model, they are primarily to enable a simpler transformation to an SBVR model and do not appear to affect the outcome of the configuration process.

Since configuration is usually applied to complex domain knowledge, it suggests the possibility of using configuration to perform semantic parsing of natural language. Using configuration, many of the different forms of knowledge required for NLP could be embedded directly in the parser and extended during processing. In the context of this thesis, that knowledge would be the formal model of the natural language specification.

2.5 Summary

In summary, the four fields discussed in this chapter—Model-Driven Engineering (MDE), Requirements Engineering (RE), Natural Language Processing (NLP), and Knowledge-based Configuration—contribute to the aim and outcomes of this thesis, as illustrated in Figure 2.7.

MDE is the foundation and application area of this thesis, demonstrating the desire, need, and benefits of formalisation in software development. As part of the traditional software development life-cycle, RE still has its place in MDE; however, the natural language artefacts of RE need to be formalised and integrated into MDE to gain the most benefit. This formalisation should be performed using automated NLP techniques to produce consistent and correct models while minimising the amount of effort required. Furthermore, using appropriate NLP techniques will enable non-technical users to perform the formalisation themselves, which allows the specifications to be validated at the source by the domain experts who have the necessary knowledge. Finally, knowledge-based configuration appears to be a promising approach to NLP which can potentially incorporate that knowledge into an understanding of the domain and form the basis of the formalisation of requirements specifications.

\[7\] As opposed to Cognitive Grammar style syntactic analysis discussed recently.
In the following chapter we survey a large number of approaches to creating formal models automatically, or semi-automatically, from natural language specifications and identify their limitations with respect to achieving the aim of this thesis.
Chapter 3

Survey of existing Tools/Techniques

As we have shown in the previous chapter, the formalisation of natural language specifications plays a major role in improving the software development process, particularly for MDE. By formalising the specifications, it allows consistency checking, transformations and traceability between artefacts, and improves the overall quality of the software and specification (if changes to the formal model are fed back into the specification). Ideally, automated tools would be able to perform an accurate, complete, and correct formalisation of natural language specifications; however, such NLP techniques are still a long way off (if they can ever be achieved).

In this chapter we consider existing tools and approaches to formalising natural language specifications as formal models. In doing so, we answer the first research question,

What are the current approaches to (semi-)automatically transforming natural language (business) specifications into formal models and how suitable are they for use by (business) domain experts?

While we have previously discussed some of the existing manual approaches, here we focus on automated and semi-automated approaches as they are most likely to achieve the aim of the thesis, in particular in reducing the time and manual effort involved in creating the models.
Chapter 3. Survey of existing Tools/Techniques

This chapter is broken down into two sections. Section 3.1 provides an overview of approaches in three categories that we have identified. In Section 3.2, a comparative review of approaches is performed based on a number of important comparison criteria.

3.1 Overview of Related Work

In this section we provide an overview of approaches to transforming natural language specifications into formal models. We have identified three main categories of approaches: (1) those that perform a complete parse of a specification using a formal grammar, termed Formalist approaches; (2) those that process unrestricted texts using Information Extraction (IE) techniques; and (3) those that perform a complete parse of the sentence without necessarily using formal grammars, which we refer to as Naturalist approaches. Most approaches fall into the first and second categories, while only a few fall into the third.

3.1.1 Formalist Approaches

The first category is characterised by parsing controlled natural language using a formalist approach, in which a formal language is simplified to make it more accessible to non-experts \cite{CMHT10}. Formalist approaches provide complete parsing of a specification with respect to their formal grammar, which results in reduced naturalness of the text. Due to the strict requirements of the grammar and its basis in a formal language, greater manual effort is necessary to translate specifications into the language and the user must have more technical knowledge of the underlying formalism.

To make up for this, Formalist approaches are often tool-driven. They provide editors that embed parsers generated from their formal grammar (see \cite{TC06, RPH08, NSSS+10, SPI10, NCGE13}) or graphical user interfaces that guide the user through the creation of rules (see \cite{Lin06, RCV12}). One advantage of such approaches is that they often include text prediction or ‘code-completion’ features (e.g. those in \cite{TC06, NSSS+10, NCGE13}), which help to create valid sentences. Moreover, while they perform no semantic analysis to parse a sentence, they can often validate the produced model either after each rule is entered or at the request
of the user (e.g. \cite{TC06, NSSS+10, NCGE13}). In the case of \cite{SNPS12}, the authors use an OWL (Web Ontology Language) reasoner on the transformed model to check its consistency.

The more general knowledge representation approaches, PENG Light \cite{Sch08, Sch10} and Attempto Controlled English (ACE) \cite{FKK08}, also provide predictive editors. Where they differ is in the complexity of the language that they process, allowing elements such as anaphoric reference in the text. Moreover, they take some semantic information into account to produce their output formalisms, but leave the complete semantics for further processing and reasoning. As general knowledge representation languages, PENG Light and ACE are not particularly suited to business specifications (even though ACE was initially applied to software specifications, see \cite{FS95}). In particular, PENG Light lacks expressiveness with respect to modality, while ACE supports only necessity/possibility modality (while business rules often use permission/slash obligation as well) and has the characteristic that all sentences are unambiguous even if deemed ambiguous to the user. This characteristic requires the user to learn a specific set of interpretation rules to ensure they are interpreting the sentences in the same way as ACE, which can be unintuitive to the user. Instead, to support formalisation by domain experts, a better approach would be to identify the ambiguity and reformulate it into an unambiguous form that is intuitive to the user.

Another approach in the formalist category is NL-OOPS \cite{Mic96}, which attempted to produce object-oriented models from unrestricted text. It is considered formalist as the underlying natural language processing engine, LOLITA, depended primarily on a very large formal grammar (rather than the IE approaches of the following section). Where NL-OOPS differs from the other formalist approaches is that it performed a deep semantic analysis of the text (courtesy of LOLITA’s extensive semantic network containing many lexical and semantic relations), from which it extracted the object-oriented elements. However, the approach was never fully evaluated and, along with the underlying NLP engine, is no longer available to the best of our knowledge.
3.1.2 Information Extraction Approaches

The second category encompasses approaches that use IE techniques to process natural language specifications. These approaches utilise rules, patterns, or templates to identify objects and relations in unrestricted textual specifications, usually to produce initial models that can be used as a basis of understanding and refinement (see [HG03, AU06, IO06, FKMS+07, ABB11, FMP11, AS12, FE12, LZC13, SDGG13, SA14]). While the exact rules vary, they typically recognise common nouns or noun phrases as candidate types, proper nouns as individuals, verbs as possible relations, and adjectives as possible attributes. Moreover, the approaches tend to differ on how they filter the candidates to remove spurious types and relations, which leads to different performances between approaches. Although the exact models produced are varied, many approaches utilise intermediate models from which they produce the final models (see [HG03, IO06, FKMS+07, FMP11, AS12]). This is important as the final model, for example a UML Class or Activity model, does not necessarily capture all of the information contained in the text.

The advantage of IE approaches is that they perform completely automated analysis of the documents (except [FKMS+07] and the semi-automatic mode of [HG03]), so there is no manual translation required on the part of the user. However, the IE approaches typically perform shallow and incomplete parsing as they extract elements using rules that search for syntactic patterns and ignore parts of a sentence. As a result, it is difficult to provide in-depth feedback to the user about possible errors, ambiguities, or inconsistencies in the documents or produced models. Moreover, the produced models need manual validation and refinement to ensure that they are a true representation of information of the documents [MBM13].

To circumvent this, some approaches require that the user provide a domain model (e.g. [BBL10, BLB11, SDGG13] or glossary (e.g. [AS12, FE12, SSA12]). In particular, the work of [SDGG13] uses a domain model to provide deep analysis; however, it requires increased effort and technical knowledge to provide a complete and consistent domain model in the first place, thereby eliminating the benefits gained by using IE techniques.

Another means of improving the results of IE approaches is to restrict the forms of input, as in [LKM10, CS08, YBL10, YBL13, SSA12]. Each of these approaches restricts the input text
Section 3.1. Overview of Related Work

to a lesser or greater extent. For example, in \cite{CS08, YBL13, YBL10} use case templates are used for specifying the requirements, while \cite{SSA12} the input texts are restricted to “simple sentences in active voice.” In each case the simplified input allows the approaches to improve their results by focusing their efforts on the specified type of input rather than trying to process all kinds of text.

One particularly promising approach is that of \cite{IO06}, in which structural and behavioural models are created from a pair of intermediate models: a tabular representation of subject, predicate, object triples and a semantic network of directed relations. While their approach appears to utilise the desired information of rules (e.g. modality, quantification, etc.) in order to identify them, it appears that they are not formalised in the produced models. For example, their Hybrid Activity Diagrams (a cross between UML Activity and Sequence diagrams) appear to include conditions only as simple text. As such it is useful as a simple analysis tool, but not a method of formalising business specifications.

Finally, the major drawback for \cite{IE} based approaches is their variable performance. For example, in \cite{HG03} precision and recall across several simple case study texts ranged between 40-100% for recall and 57-80% for precision, while in \cite{ABB11} ranges of 75-93% recall and 82-93% precision were reported. While such measures are a contentious issue, since there is usually not a single correct model, it highlights the difficulty of acquiring complete and precise models from text. If an approach aims to use the models for further automated transformations, it is perhaps better to ensure that precise and correct models are first created that more directly represent the logical meaning of the text.

3.1.3 Naturalist Approaches

The third category contains relatively few approaches, which are characterised by performing complete parsing while not relying on a typical formal grammar. To various degrees, the approaches of this category attempt to gain the benefits of both the formalist and \cite{IE} approaches, such as:

- more natural and fluent languages for users
Chapter 3. Survey of existing Tools/Techniques

- precise analysis
- robust feedback on errors and inconsistencies
- higher expressiveness
- the ability to process a wider variety of documentation

The approach of [KAB09b] produces SBVR models and UML Class diagrams through a series of transformations starting from natural language text. Their parsing is performed using configuration; however, they utilise configuration as a more flexible means of generating traditional syntactic parse trees while performing semantic analysis as an additional step. Moreover, the method of processing the text into the models for configuration is left somewhat trivialised and unspecified in [KAB09b]. Lastly, their approach requires a domain specific lexicon containing detailed linguistic information that goes beyond what a business user should be expected to have.

The CPL language [CMHT10] is similar to ACE and PENG Light in that it is more of a general knowledge representation language. In contrast to those formalist approaches, it surrounds a formalist core with a naturalist shell to gain the benefits of both languages. As a result it performs complete parsing, but can be less natural if the user must resort to using the core language. Like the other knowledge representation languages, it does not support the level of expressiveness required for business specifications. Finally, CPL relies on a pre-defined ontology to correctly interpret sentences, rather than attempting to acquire the vocabulary and relations from the business documentation itself.

An approach that is very similar to our own in both motivation and technique is CIRCE [AG06]. Their approach uses matching rules augmented with simple semantic tags to process natural language specifications; however, it performs complete parsing by reporting unparsed portions of text. Furthermore, they aim to provide significant feedback to support the user in developing a complete, correct, and unambiguous specification. While their tool is much more mature than ours, supporting various output models and analyses to provide feedback, CIRCE requires a domain specific glossary and rules (which they called definitions) provided in a technical notation to perform the parsing of natural language. Therefore, while their approach would still be highly usable for a more technical user, it is deemed inappropriate for
Section 3.2. Tool Comparison

use by business and domain experts. Finally, the publicly available version of the tool that was promised\(^1\) has not materialised to the best of our knowledge.

3.2 Tool Comparison

The overview of approaches to transforming natural language business specifications into formal models shows that the different approaches have their advantages and disadvantages. To better characterise them and identify whether or not any existing approaches are suitable for our aim, we perform a more detailed comparison to answer the question:

What are the current approaches to (semi-)automatically transforming natural language (business) specifications into formal models and how suitable are they for use by (business) domain experts?

To aid in answering this question and derive some criteria for comparison, we decompose this question into several sub-questions as follows:

1. Is the approach/language expressive enough to describe the user’s requirements?

2. Is the approach/language natural enough for a domain expert to use intuitively?

3. Is the approach fully or semi-automatic?

4. Can/does the approach provide detailed feedback on errors and/or inconsistencies in the specification?

5. Does the approach require any inputs other than the specification?
   (a) If so, what are they?
   (b) —, do they require any special notations?

6. What type of models does the approach create?

\(^1\)http://circe.di.unipi.it/Eclipse/resources.html – viewed 5 September 2014
7. At what level of abstraction in the Model-Driven Architecture are the produced model(s) situated?

8. Is it possible for the approach to be tailored to the business/domain expert’s needs?

These questions led to the definition of the following criteria.

**CNL Class** In partial answer to questions (1) and (2) we consider the *class* of CNL processed by an approach in terms of the PENS scheme proposed by Kuhn [Kuh14]. Each of its four dimensions—precision, expressiveness, naturalness, and simplicity—is separated into 5 distinct categories along a continuum, with formal language (e.g. propositional logic) at one end and natural language (e.g. English) at the other. The categories are defined objectively such that (given enough information) any language can easily be situated within the appropriate category. In the PENS scheme all natural languages are of the class $P^1 E^5 N^5 S^1$, while formal languages such as propositional logic are of the class $P^5 E^1 N^1 S^5$, and SBVR SE is classified\(^2\) as $P^3 E^4 N^4 S^2$ [Kuh14]. The four dimensions are defined as follows\(^3\):

- **Precision**: describes the degree to which the language can be interpreted without ambiguity and ranges from imprecise languages ($P^1$), in which complex sentences are almost always ambiguous, to languages with fixed semantics ($P^5$), in which each sentence has exactly one meaning.

- **Expressiveness**: categorises languages based on the minimal set of propositions that they support; ranging from inexpressive languages ($E^1$) that are limited to universal quantification and/or unary predicates, to maximally expressive languages ($E^5$) that can express anything.

- **Naturalness**: is the degree to which a language can be read and understood intuitively, with respect to natural language, and can range from unnatural languages ($N^1$), which make extensive use of symbols, brackets, and unnatural keywords, to

---

\(^2\)This is the classification of SBVR SE as defined in the SBVR specification, not the variations that are processed by any specific approach.

\(^3\)For brevity, we provide only a brief description of each dimension here. A more complete explanation of the dimensions can be found in Appendix A.
Section 3.2. Tool Comparison

languages with natural texts \((N_5)\), in which even long texts appear completely natural.

**Simplicity:** describes the complexity of a complete language description (syntax and semantics) based on the number of pages it would take to describe it, for which the categories range from *very complex* \((S_1)\), i.e. the language does not have such a description, to *languages with very short descriptions* \((S_5)\) that can be defined in a single page.

To answer question (1), however, we need to incorporate some finer distinctions to the *expressiveness* dimension. An important element of specifications is to specify what should, must, or may be required; however, such modalities are not covered in Kuhn’s definition of expressiveness. Therefore, we indicate support for modality with the subcategory \(E^{(2,3,4)}_m\).

Another important aspect that impacts both expressiveness and naturalness is support for n-ary relations. While n-ary relations can be decomposed into binary relations [RPH08, NSSS+10], such decomposition must be performed by the user if the language does not support relations of any arity. This negatively impacts the naturalness of the specification and leads to more effort on the part of the user for a task that, ideally, would be automated in a model transformation. Since many languages are restricted to relations of arity \(\leq 2\) and the standard PENS scheme only differentiates between relations of arity 1 and \(\geq 1\), we define another subcategory \(E^{(2,3,4)}_{\leq 2}\) to indicate that a language is restricted to relations with an arity of at most 2. This can be generalised to any specification of allowed arity as using an arbitrary relation operator (e.g. \(\leq, =\)) and the arity; for example, the category \(E^{(2,3,4)}_{=2}\) would indicate that a language is restricted to binary relations only (i.e. all relations have an arity of 2).

Note that the subcategories can be combined (i.e. \(E^{(2,3,4)}_{\leq 2m}\)) to indicate that a language is restricted to unary and binary relations, while supporting modal logic.

Since the naturalness dimension of Kuhn [Kuh14] exclusively refers to readability and understandability, we define an additional dimension: writability naturalness. This relates to the “writability problem” discussed by Kuhn [Kuh10]. The problem is that writing documents conforming to the restrictions of the CNL is more difficult than reading it, that it can be a slow process, and that domain experts require training in the CNL to become proficient with
We define the writability naturalness dimension similarly to the others of the PENS scheme with the following 5 categories:

\( N^1_w \) Languages requiring complete specialised knowledge are unintuitive to write, require extensive training (e.g. more than 1 week of intensive workshops), and need tools to correctly write sentences and full texts in the language.

\( N^2_w \) Languages requiring a high-level of specialised knowledge are difficult to write intuitively (simple sentences may be intuitive, but more complex sentences become unintuitive), require training (e.g. up to 1 week of intensive workshops), and tools are necessary to correctly and efficiently write sentences and full texts in the language.

\( N^3_w \) Languages requiring moderate specialised knowledge can be written reasonably intuitively; however, the user must take into account various syntactic and/or semantic restrictions that reduce their efficiency in writing sentences and full texts in the language. Some training (e.g. a single day intensive workshop) and tool support is necessary to become proficient at writing in these languages.

\( N^4_w \) Languages requiring little specialised knowledge require that the user learn some simple restrictions or guidelines of the language but can otherwise write sentences and full texts intuitively and efficiently without the need for specialised tools. Training and tool support may be beneficial but not necessary for languages in this class.

\( N^5_w \) Languages requiring no specialised knowledge are completely natural to write and the user does not need to take into consideration any restrictions of the language (this does not necessarily mean that there are not any, only that they are transparent to the user).

According to these definitions we extend the classification of SBVR SE to include \( N^4_w \) as we consider the documentation on SBVR SE good enough for a user to not require training and SBVR SE documents can be written using standard Word Processing Software\(^4\).

\(^4\)The need for specialised SBVR tools is only necessary if the user wishes to create an SBVR model from the text; however, users can still make use of SBVR SE to help better communicate their specifications without creating a model from them.
Section 3.2. Tool Comparison

**Supported Rule Types**  The types of rules supported by an approach affects the expressiveness of the approach, question (1). An approach can support *structural* rules, *operational* rules, or both. Structural rules typically define constraints on relationships between concepts, such as cardinality constraints, while operational rules can define behaviour in a declarative fashion, such as stating that some action must be performed when a certain situation occurs [OMG08b]. Without support for both types of rules, a user may not be able to appropriately formalise their requirements.

When determining if an approach supports a type of rule where it is not explicitly specified, it is considered to support both if it has a means of differentiating between them. If an approach cannot differentiate, it is determined to be one or the other depending on how the rules are handled.

**Degree of Automation (DoA)**  To answer question (3) we define the Degree of Automation criterion to be either *automatic* or *semi-automatic*. An approach is automatic if the entire process of creating the model(s) can be performed without user interaction (once initiated by a user); otherwise, the approach is considered semi-automatic. Manual approaches are not included in the comparison.

**Analysis Type**  To provide insight into question (4), we consider the type of analysis performed by an approach. The types of analysis are: shallow, shallow+, deep, and wizard. Shallow (syntactic) analysis only looks at the surface structure (syntax) of a sentence, while deep analysis looks at the semantics of a sentence combined with context and background [CFBM+02; KL09]. The type shallow+ (i.e. shallow semantic) denotes approaches that use techniques such as *semantic/thematic role labelling* [MCLS08] and *discourse representation structures* [KR93] to incorporate some semantics into the analysis, but do not incorporate domain (or general) knowledge, context, etc. The wizard type does not perform any language analysis at all. Instead, graphical user interfaces are employed to guide the user in entering only valid constructions.
Deep analysis is capable of providing the most relevant feedback on errors and inconsistencies in a specification, while the restrictions of wizard approaches attempt to prevent errors and inconsistencies from occurring in the first place.

**Parsing Completeness**  The parsing completeness criterion assesses how much each sentence or business rule is processed, which also helps to answer question (4). For example, *incomplete* approaches that parse sentences by scanning them for elements of interest and ignoring the rest are unlikely to report erroneous statements reliably. On the other hand, approaches that perform *complete* processing of a sentence can more reliably report erroneous statements.

**Additional Inputs**  This criterion directly answers question (5a) by identifying additional inputs required by an approach (if any). The types of inputs required may be one or more of the following:

- *(Domain Specific) Glossary:* a list of words and their definitions, possibly more formal than a standard glossary of terms.
- *(Domain Specific) Lexicon:* a list of words and lexical information such as category (noun, verb, etc.), voice (passive or active), plurality, gender, etc.
- Templates or Patterns: rules with some schematic component that are matched against sentences to define the allowable forms that they can take or to extract specific information from them.

To answer question (5b), each additional input is identified as requiring a notation other than natural language or not (denoted by an asterisk `*`). Intuitively, special notations reduce the suitability of an approach to be used directly by business people; therefore, special notations should be kept to a minimum. The notations themselves can differ widely, from formal languages to more natural [CNLs](http://www.cnl.org), so some may be more acceptable to business users than others.
Section 3.2. Tool Comparison

Output Models  In answer to question (6), the output models criterion specifies what types of models are created by an approach. Some possibilities include UML models, including Class and/or Activity Diagrams, Business Process Model and Notation (BPMN) models, Entity-Relationship (ER) models, etc. The type of models created determines what information can be represented by an approach; whether it be static/structural information, or dynamic/behavioural information. As a result, this criterion partly ties in with the supported rule types and supports question (1) as well. However, the two are independent of one another in that an approach may, for example, identify structural information during the analysis of operational rules.

Level of Abstraction (LoA)  This criterion explicitly addresses question (7) to identify at what level of the Model-Driven Architecture (MDA) the approach targets, the: Computation Independent (or Business Model) Layer; Platform Independent Layer; or Platform Specific Layer [MM03]. Ideally, transformations between layers should be as automated as possible, with business or domain experts specifying their requirements at the top-most level. While not all approaches use the MDA classifications, deciding what level of abstraction an approach targets is relatively straightforward. References to problem domain, domain knowledge, conceptual or business model, and similar are classified as CIM level, while mentions of solution domain, system model, detailed design, or similar, are considered to target the PIM level. It seems reasonable to assume no approach targets PSMs as they are usually created by refinement from PIMs. If no specific level is indicated, a judgement is made as to what the highest-level the approach could support.

Adaptability  In order for an approach to be easily adopted by an organisation for representing its specifications it is important that it be easily adaptable to the documents used by that organisation. This criterion addresses question (8) and gives an indication as to whether or not complete rewriting of existing organisational documentation is necessary. Since business documentation can vary widely in the type of documents and language used, we have identified two important ways in which an approach should be adaptable by domain experts.
• **Document structure adaptability** means the approach can be adapted by users to process different forms of document rather than restricting users to, say, the SBVR SE document structure, and

• **Language adaptability** means the approach can be adapted by users to allow alternative phrases for keywords, e.g. ‘at most’ could also be said as ‘no more than’, or allowable sentence patterns. This does not include new semantics, only different ways of expressing the semantics.

Previous surveys, such as [YBL11; MBM13], address similar questions and utilise similar criteria. While valuable in their own right, they are either too high-level and describe approaches in broad categories, as in [MBM13], or provide too much unnecessary detail, as in [YBL11], without answering the question that is most important in our context: i.e., is the approach suitable for use directly by business and domain experts?

### 3.2.1 Suitability of Criteria

The criteria described above are considered suitable as they cover many of the aspects that serve as points of differentiation and commonality between different approaches to transforming natural language specifications into formal models. Moreover, in the absence of user studies to determine whether an approach is appropriate for use by domain experts, we believe these criteria can serve as indicators to that effect.

Although it may seem inappropriate to apply the CNL Class criterion to IE-based approaches, which process unrestricted text, we do not believe that to be the case. IE-based approaches place restrictions on the words and semantics they can extract through the rules, templates, or training they use. As such, it conforms to the definition of a CNL as ‘...a constructed language that is based on a certain natural language, being more restrictive concerning lexicon, syntax and/or semantics while preserving most of its natural properties’ [Kuh14]. Furthermore, some of the IE-based approaches explicitly restrict the form of the input text, making them even more controlled than standard IE approaches.
Finally, while some criteria overlap with others, none completely subsume one another. For example, the type of rules supported is quite strongly related to support for modalities; however, an approach may support modalities while restrict certain types of rules or vice versa if the rule types are indicated by some other means. Therefore, this list of criteria cannot be reduced without losing some distinctions between approaches. On the other hand, this list of criteria is not complete as, for example, we could include a criterion that describes the exact pipeline of NLP tools used by an approach (as was done in [YBL11]); however, such detail would not provide any additional information with respect to the question(s) being asked.

### 3.2.2 Results

The results of the comparison are displayed in Table 3.1 (on pages 46 to 48), with the different approaches grouped into the categories discussed in Section 3.1: Formalist, Naturalist, and Information Extraction. The Formalist approaches are characterised by their use of a formal grammar that performs a complete parse of each sentence. On the other hand Information Extraction approaches are characterised by their processing of unrestricted text using rules, patterns, or templates to identify elements of interest without performing a comprehensive parse of each sentence. The Naturalist approaches sit between the two, performing complete parsing of a sentence without relying on a restrictive formal grammar.
<table>
<thead>
<tr>
<th>Approach</th>
<th>CNL Class</th>
<th>Rule Types</th>
<th>DoA</th>
<th>Analysis Type</th>
<th>Complete</th>
<th>Inputs</th>
<th>Outputs</th>
<th>LoA</th>
<th>Adaptability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Formalist</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[TC06]</td>
<td>(P^5 E^4 \leq 2m N^3 N^3 S^4)</td>
<td>S+O</td>
<td>Semi</td>
<td>S</td>
<td>✓</td>
<td>G*</td>
<td>SBVR, Prolog</td>
<td>CIM</td>
<td>×</td>
</tr>
<tr>
<td>[RPH08]</td>
<td>(P^5 E^4 \leq 2m N^3 N^3 S^4)</td>
<td>S+O</td>
<td>Semi</td>
<td>S</td>
<td>✓</td>
<td>G*</td>
<td>UML (Class, Activity, Sequence)</td>
<td>PIM</td>
<td>×</td>
</tr>
<tr>
<td>[NSSS+10]</td>
<td>(P^5 E^4 \leq 2m N^3 N^3 S^4)</td>
<td>S+O</td>
<td>Semi</td>
<td>S</td>
<td>✓</td>
<td>G*</td>
<td>SBVR, UML/OCL (Class)</td>
<td>CIM</td>
<td>×</td>
</tr>
<tr>
<td>[SPI10]</td>
<td>(P^5 E^4 \leq 2m N^3 N^3 S^4)</td>
<td>S</td>
<td>Semi</td>
<td>S</td>
<td>✓</td>
<td>G*</td>
<td>sSBVRMM, UML (Class), BPMN</td>
<td>CIM</td>
<td>×</td>
</tr>
<tr>
<td>[NCGE13]</td>
<td>(P^5 E^4 \leq 2m N^3 N^3 S^4)</td>
<td>S+O</td>
<td>Semi</td>
<td>S</td>
<td>✓</td>
<td>G*</td>
<td>SBVR</td>
<td>CIM</td>
<td>×</td>
</tr>
<tr>
<td>[RCV12]</td>
<td>(P^5 E^4 \leq 2m N^3 N^3 S^4)</td>
<td>S+O</td>
<td>Semi</td>
<td>W</td>
<td>✓</td>
<td>G, T*</td>
<td>ECA-rules</td>
<td>PIM</td>
<td>D+L</td>
</tr>
<tr>
<td>[Lin06]</td>
<td>(P^5 E^4 \leq 2m N^3 N^3 S^4)</td>
<td>S+O</td>
<td>Semi</td>
<td>W</td>
<td>✓</td>
<td>DM*</td>
<td>OCL</td>
<td>CIM</td>
<td>×</td>
</tr>
<tr>
<td>[SP14]</td>
<td>(P^5 E^4 \leq 2m N^3 N^3 S^4)</td>
<td>S+O</td>
<td>Semi</td>
<td>W</td>
<td>✓</td>
<td>G*, T*</td>
<td>OWL2</td>
<td>PIM</td>
<td>×</td>
</tr>
<tr>
<td>[SNPS12]</td>
<td>(P^5 E^4 \leq 2m N^3 N^3 S^4)</td>
<td>S</td>
<td>Semi</td>
<td>S</td>
<td>✓</td>
<td>G*</td>
<td>SBVR, OWL2/SWRL/SPARQL</td>
<td>CIM</td>
<td>×</td>
</tr>
<tr>
<td>[Sch08; Sch10]</td>
<td>(P^5 E^4 \leq 2m N^3 N^3 S^3)</td>
<td>S</td>
<td>Semi</td>
<td>S+</td>
<td>✓</td>
<td>L*</td>
<td>TPTP</td>
<td>CIM</td>
<td>×</td>
</tr>
<tr>
<td>[FKK08]</td>
<td>(P^5 E^4 \leq 2m N^3 N^3 S^3)</td>
<td>S</td>
<td>Semi</td>
<td>S+</td>
<td>✓</td>
<td>L*</td>
<td>DRS</td>
<td>CIM</td>
<td>×</td>
</tr>
<tr>
<td>[Mic96]</td>
<td>(P^5 E^4 \leq 2m N^3 N^3 S^3)</td>
<td>S</td>
<td>Semi</td>
<td>S+</td>
<td>✓</td>
<td>L*</td>
<td>DRS</td>
<td>CIM</td>
<td>×</td>
</tr>
</tbody>
</table>

Continued on next page.
### Table 3.1 – continued.

<table>
<thead>
<tr>
<th>Approach</th>
<th>CNL Class</th>
<th>Rule Types</th>
<th>DoA Analysis Type</th>
<th>Complete</th>
<th>Inputs</th>
<th>Outputs</th>
<th>LoA</th>
<th>Adaptability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Naturalist</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[KAB09b]</td>
<td>$P^3 E^4 N^4 N_w^4 S^2$</td>
<td>S+O</td>
<td>Auto</td>
<td>D</td>
<td>✓</td>
<td>L*</td>
<td>SBVR, UML (Class)</td>
<td>CIM</td>
</tr>
<tr>
<td>[CMHT10]</td>
<td>$P^3 E^3 N^4 N_w^3 S^2$</td>
<td>S</td>
<td>Semi</td>
<td>S+</td>
<td>✓</td>
<td>DM*</td>
<td>Prolog</td>
<td>CIM</td>
</tr>
<tr>
<td>[AG06]</td>
<td>$P^3 E^4 N^4 N_w^4 S^3$</td>
<td>S+O</td>
<td>Semi</td>
<td>S+</td>
<td>✓</td>
<td>G*, T*</td>
<td>Various static and dynamic</td>
<td>CIM</td>
</tr>
<tr>
<td>[GH09]</td>
<td>$P^3 E^3 N^4 N_w^4 S^4$</td>
<td>O</td>
<td>Semi</td>
<td>S</td>
<td>✓</td>
<td>×</td>
<td>Live Sequence Charts (LSCs)</td>
<td>PIM</td>
</tr>
<tr>
<td><strong>Information Extraction</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[AU06]</td>
<td>$P^1 E_{≤2m}^3 N^5 N_w^5 S^3$</td>
<td>S+O</td>
<td>Auto</td>
<td>S</td>
<td>×</td>
<td>×</td>
<td>UML (Class)</td>
<td>PIM</td>
</tr>
<tr>
<td>[BBL10]</td>
<td>$P^1 E_{≤2m}^4 N^5 N_w^5 S^3$</td>
<td>S+O</td>
<td>Auto</td>
<td>S+</td>
<td>×</td>
<td>DM*</td>
<td>OCL</td>
<td>CIM</td>
</tr>
<tr>
<td>[ABB11]</td>
<td>$P^4 E_{≤2m}^4 N^5 N_w^5 S^3$</td>
<td>S</td>
<td>Auto</td>
<td>S+</td>
<td>×</td>
<td>×</td>
<td>UML (Class)</td>
<td>PIM</td>
</tr>
<tr>
<td>[BLB11]</td>
<td>$P^1 E_{≤2m}^4 N^5 N_w^5 S^3$</td>
<td>S+O</td>
<td>Auto</td>
<td>S+</td>
<td>×</td>
<td>DM*</td>
<td>SBVR</td>
<td>CIM</td>
</tr>
<tr>
<td>[FE12]</td>
<td>$P^1 E_{≤2m}^4 N^5 N_w^5 S^3$</td>
<td>S+O</td>
<td>Auto</td>
<td>S+</td>
<td>×</td>
<td>G</td>
<td>SBVR</td>
<td>CIM</td>
</tr>
<tr>
<td>[AS12]</td>
<td>$P^1 E_{≤2m}^4 N^5 N_w^5 S^3$</td>
<td>S+O</td>
<td>Auto</td>
<td>S</td>
<td>×</td>
<td>G, T*</td>
<td>RSL-IL</td>
<td>CIM</td>
</tr>
<tr>
<td>[FMP11]</td>
<td>$P^1 E_{≤2m}^4 N^5 N_w^5 S^3$</td>
<td>O</td>
<td>Auto</td>
<td>S+</td>
<td>×</td>
<td>×</td>
<td>World Model, BPMN</td>
<td>CIM</td>
</tr>
<tr>
<td>[FKMS+07]</td>
<td>$P^1 E^3 N^8 N_w^6 S^3$</td>
<td>S+O</td>
<td>Semi</td>
<td>S+</td>
<td>×</td>
<td>×</td>
<td>KCPM, UML (Class, Activity, Statechart)</td>
<td>CIM</td>
</tr>
<tr>
<td>[SDGG13; SDGG14]</td>
<td>$P^1 E^3 N^5 S^2$</td>
<td>S+O</td>
<td>Semi</td>
<td>S</td>
<td>×</td>
<td>DM*, T*</td>
<td>OWL/Reified Requirement Templates</td>
<td>PIM</td>
</tr>
</tbody>
</table>

*Continued on next page.*
Table 3.1 - continued.

<table>
<thead>
<tr>
<th>Approach</th>
<th>CNL Class</th>
<th>Rule Types</th>
<th>DoA</th>
<th>Analysis Type</th>
<th>Complete</th>
<th>Inputs</th>
<th>Outputs</th>
<th>LoA</th>
<th>Adaptability</th>
</tr>
</thead>
<tbody>
<tr>
<td>[LZC13]</td>
<td>$P_1^1 E^3 N_5^5 N_5^5 S^3$</td>
<td>S</td>
<td>Auto</td>
<td>S</td>
<td>×</td>
<td>×</td>
<td>UML (Class)</td>
<td>PIM</td>
<td>D+L</td>
</tr>
<tr>
<td>[SA14]</td>
<td>$P_1^1 E_{\leq 2}^3 N_5^5 N_5^5 S^3$</td>
<td>S</td>
<td>Auto</td>
<td>S</td>
<td>×</td>
<td>×</td>
<td>UML (Class)</td>
<td>CIM</td>
<td>D</td>
</tr>
<tr>
<td>[SSA12]</td>
<td>$P_1^1 E_3^3 N_5^5 N_5^4 S^3$</td>
<td>S+O</td>
<td>Auto</td>
<td>S</td>
<td>×</td>
<td>G</td>
<td>UML (Class, Component)</td>
<td>PIM</td>
<td>D</td>
</tr>
<tr>
<td>[HG03]</td>
<td>$P_1^1 E_3^3 N_5^5 N_5^4 S^3$</td>
<td>S</td>
<td>Semi</td>
<td>S</td>
<td>×</td>
<td>×</td>
<td>UML (Class)</td>
<td>CIM</td>
<td>D</td>
</tr>
<tr>
<td>[HG03]</td>
<td>$P_1^1 E_3^3 N_5^5 N_5^4 S^3$</td>
<td>S</td>
<td>Auto</td>
<td>S+</td>
<td>×</td>
<td>×</td>
<td>XI World Model, UML (Class)</td>
<td>CIM</td>
<td>D</td>
</tr>
<tr>
<td>[IO06]</td>
<td>$P_1^1 E_{m}^4 N_5^5 N_5^4 S^3$</td>
<td>S+O</td>
<td>Auto</td>
<td>S</td>
<td>×</td>
<td>×</td>
<td>UML-like (Class, Activity)</td>
<td>CIM</td>
<td>D</td>
</tr>
<tr>
<td>[EVR11]</td>
<td>$P_1^1 E_{\leq 2}^3 N_5^5 N_5^4 S^3$</td>
<td>S</td>
<td>Auto</td>
<td>S</td>
<td>×</td>
<td>×</td>
<td>ER</td>
<td>CIM</td>
<td>D</td>
</tr>
<tr>
<td>[LKM10]</td>
<td>$P_1^1 E_2^2 N_5^4 N_5^4 S^3$</td>
<td>S</td>
<td>Auto</td>
<td>S</td>
<td>×</td>
<td>G</td>
<td>ER</td>
<td>PIM</td>
<td>D+L</td>
</tr>
<tr>
<td>[CS08]</td>
<td>$P_1^1 E_3^3 N_4^4 N_4^4 S^3$</td>
<td>O</td>
<td>Semi</td>
<td>D</td>
<td>×</td>
<td>L*, DM*, T</td>
<td>CSP process algebra</td>
<td>PIM</td>
<td>×</td>
</tr>
<tr>
<td>[YBL10]</td>
<td>$P_1^1 E_3^3 N_4^4 N_4^4 S^2$</td>
<td>O</td>
<td>Auto</td>
<td>S</td>
<td>×</td>
<td>×</td>
<td>UCMeta (intermediate model), UML (Class, Activity, Sequence)</td>
<td>PIM</td>
<td>×</td>
</tr>
</tbody>
</table>

Legend:
- **Rule Types**: Structural (S), Operational (O);
- **Analysis Type**: Shallow (S), Shallow+ (S+), Deep (D), Wizard (W);
- **Inputs**: glossary (G), lexicon (L), domain model (DM), rules or templates (T), special notation (*);
- **Adaptability**: document structure (D), language (L);
- $^a$ Has the specified features only if appropriate patterns/templates have been provided;
- $^b$ Restricted permission rules only;
- $^c$ Various via transformations from logical representation;
Section 3.2. Tool Comparison

In order to answer the main question posed at the beginning of the survey, we apply constraints on the various criteria. This will allow us to identify which approaches, if any, are appropriate for use by business experts to help formalise their specifications as models.

First of all, we focus on the approaches that can be used to produce models at the business (or CIM) level of abstraction. Most approaches can be used for this purpose (even if not initially designed for it) as only 12 of the 35 approaches surveyed are specific to the PIM level. Moreover, since CIM level models should capture requirements of the business environment and its operations, it is important for an approach to support both structural and operational rules. This more than halves the number of candidate approaches to 15 of the 35 surveyed.

In terms of the PEN(N)S classes, we use SBVR SE as a guide to suitable classes since it has been designed to be used by business users to express the types of rules necessary for business specifications: i.e. high-level models and the CIM level. Since we want to be able to capture and formalise as much of a specification as possible, the Expressiveness dimension is of prime importance. Therefore, approaches with an expressiveness less than $E^4$ are considered unsuitable; this excludes 4 of the remaining 15 candidates. In addition, modality should be supported to allow the correct expression of business policies and rules, which all 11 remaining candidates do. While support for n-ary relations is preferred, we do not exclude those approaches that are restricted to binary relations from being candidates.

The other two important PEN(N)S dimensions are those for naturalness. Since the formalised specifications are validated by domain experts, the meaning of the text must be as intuitive as possible. Therefore, we consider the lowest acceptable class of Naturalness to be $N^3$ since, according the definitions of the classes from [Kuh14], below this class the meanings become unintuitive. Similarly, we consider a Writing Naturalness class of $N^3_w$ or above to be suitable. According to our definition, class $N^3_w$ requires limited training, which is acceptable, whereas the lower classes require much more extensive training potentially hampering the adoption of such approaches. Moreover, the difference between classes $N^3_w$ and $N^4_w$ is the requirement of tool support for writing a specification. While the necessity of a tool may hamper the ability to directly reuse or import existing documentation (and so a higher class is preferred), it does not exclude an approach from being suitable outright. These conditions leave the suitable candidates unchanged.
Part of the purpose of processing specifications at the CIM level is to support the creation of a domain specific model describing the business environment of an organisation. Therefore, approaches that require a Domain Model as input (particularly if they require a technical notation or language) are considered unsuitable. This reduces the potential candidates to 9. Moreover, approaches that require rules, patterns, or templates to be specified as part of the input are deemed unsuitable as it requires linguistic and/or technical knowledge of the tool that domain experts should not be expected to have. Similarly, the requirement of lexicons containing detailed linguistic information is considered unsuitable; the users should be able to focus on the meaning of their specifications. This leaves us with 4 approaches that require only a glossary of terms, for which a special notation is allowable as it typically provides only structure to the glossary, and 1 approach that requires no additional inputs.

The resultant short-list is displayed in Table 3.2 ordered by the preferences discussed so far and which shows the criteria not yet considered. Most of the approaches output SBVR models, which is not particularly surprising as we are investigating the purpose that SBVR was developed for. Moreover, as discussed earlier, SBVR is intended as an intermediate model and, hence, it provides a suitable output to be used in further processing and transformations. The one non-SBVR approach outputs a combination of structural and behavioural models at the CIM level that can similarly be used for further refinement.

At the top we have two IE approaches, which emphasise fully automated parsing of unrestricted texts. However, they achieve this by using incomplete and shallow parsing techniques, which reduces their ability to indicate errors, inconsistencies, or ambiguities to the user. At the bottom we have three almost identical Formalist approaches. They each provide the user with an editor that parses business specifications using a strict formal grammar for SBVR SE.

---

**Table 3.2: Short-list of Related Work**

<table>
<thead>
<tr>
<th>CNL Class</th>
<th>DoA</th>
<th>Analysis Type</th>
<th>Complete</th>
<th>Adaptability</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>[IO06]</td>
<td>$P^1 E^4 N^5 N^5 S^3$</td>
<td>Auto</td>
<td>S</td>
<td>×</td>
<td>D</td>
</tr>
<tr>
<td>[FE12]</td>
<td>$P^1 E^4 \leq 2 N^5 S^3 N^5 S^3$</td>
<td>Auto</td>
<td>S+</td>
<td>×</td>
<td>D</td>
</tr>
<tr>
<td>[TC06]</td>
<td>$P^5 E^4 \leq 2 N^3 S^4 S^4$</td>
<td>Semi</td>
<td>S</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>[NSSS+10]</td>
<td>$P^5 E^4 \leq 2 N^3 S^3 S^3$</td>
<td>Semi</td>
<td>S</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>[NCGE13]</td>
<td>$P^5 E^4 \leq 2 N^3 S^3 S^3$</td>
<td>Semi</td>
<td>S</td>
<td>✓</td>
<td>×</td>
</tr>
</tbody>
</table>

50
which supports a semi-automatic approach to developing business specifications in a partially formal language. As such, they have the minimum acceptable level of naturalness and limit the ability to import existing specifications that are written in less restricted language.

We conclude that none of these are approaches are necessarily better nor more suitable than one another, but rather, that they are each suited to different situations. For example, if an organisation wishes to “start from scratch” and define their business policies and rules using a formalised approach, then using the editor-based Formalist approaches would be the most appropriate. If, on the other hand, an organisation wanted to spend minimal effort in order to gain a clearer understanding of the information contained in their textual documentation, then the IE approaches are most suitable. However, these approaches are at opposite ends of the spectrum. There is a gap somewhere in the middle for approaches that aim to actively support business users in formalising (possibly existing) specifications into a form more suited to integration with the software development process. While such an approach would require more initial effort on the part of the user than the IE approaches, the possibility of identifying issues in existing documentation can help to focus the attention of the user to particular problems rather than requiring them to manually read and formalise everything themselves; the latter approach would be necessary with the Formalist tools.

### 3.3 Summary

The importance of assisting users to formalise natural language specifications cannot be understated. Having performed an extensive search, to the best of our knowledge, there is no existing approach to transforming natural language business specifications into formal models that can achieve the aims outlined in [Section 1.1](#). The formalist approaches require too much rework to be done on behalf of the business experts we aim to support. On the other hand, IE approaches are only suitable for the creation of “first-cut” models that must be manually validated and refined as they could be incomplete, incorrect, or imprecise.

While CIRCE [AG06] is close candidate for achieving the middle ground (they even share a similar motivation to our own), their aim is to assist more technical users in specifying software specifications. As such, their approach is targeted towards the PIM level, although it
can be applied to the CIM Level of Abstraction (LoA), requires additional technical knowledge of the tool in order to be used effectively, and makes use of technical notations to define the additional inputs. As a result, CIRCE is not suitable for use by business users, which is the key criterion of this study. In the chapters that follow, we present and evaluate our approach to addressing this gap.
Chapter 4

Knowledge-based Framework for CNL Understanding

With the continued desire to increase the level of abstraction in the specification and development of software, it is important that methods and tools be provided that are suitable for use by business and domain experts. However, the literature review identified a gaping hole in current approaches in that none fulfil the combination of criteria required to adequately support non-technical users in formalising their requirements. In general they are aimed at technical experts at the system level rather than the business level, require extensive manual revisions, or are not accurate enough to provide complete formal specifications.

The aim of this thesis is to investigate such a method and finally bridge the gap between natural language specifications and formal models. To that end, this chapter proposes an approach intended to fulfil the criteria identified in Section 3.2. Moreover, it describes the theoretical framework of the approach. Its implementation and evaluation is left for Chapters 5 and 6, respectively.

In Section 4.1 the proposed solution is sketched, while Sections 4.2 to 4.4 describe the general framework of the solution in detail. The general framework is described with the aid of an example making use of a simple set of Conceptual Graph semantics. Finally, the main points of the chapter are summarised in Section 4.5.
4.1 Proposed Solution and Hypotheses

In order to fulfil the criteria we believe a deep, NLU approach is necessary, as the traditional shallow NLP techniques are inadequate. However, typical approaches to NLU are not suitable either. Predominantly statistical methods do not provide the accuracy nor the detailed feedback required, while ontology-based approaches require technical expertise in knowledge engineering and/or linguistics. A possible solution is to use a CNL; however, most CNLs are too restrictive and do not allow the reuse of existing documentation. Therefore, we propose an approach based on non-traditional NLP methods.

In particular, we propose the use of Cognitive Grammar to fulfil the stated criteria. With its emphasis on meaning, the theory of Cognitive Grammar is a natural fit for NLU. Moreover, the restricted domain of a requirements specification (formal documentation) simplifies the problem of acquiring the symbolic structures of Cognitive Grammar. However, it requires many complex cognitive processes to arrive at the meaning of an utterance/sentence. To reduce this complexity, we propose the use of Knowledge-based Configuration as an alternative to the many complex processes. The idea is that using Knowledge-based Configuration will allow the domain knowledge to be embedded in the parser, facilitating NLU and will unify various disambiguation tasks by searching for the correct solution. Furthermore, the Configuration process will be able to detect errors (through constraint violations) and ambiguities (through multiple solutions), which will support the refinement of the natural language specifications and their resultant models by domain and business experts.

The combination of Cognitive Grammar and Knowledge-Based Configuration results in a general, knowledge-based framework for natural language understanding. However, we focus on using it to specify CNLs rather than unrestricted natural language, since the semantics are restricted by the configuration model used. However, with careful selection or development of the configuration model, the flexibility of the framework allows the CNL to be less restrictive than formal approaches. Moreover, with the predominantly semantics-based analysis of Cognitive Grammar, the definition of the knowledge in the lexicon can be simplified and more accessible to non-linguistic or non-technical users over ontology-based approaches.\footnote{While the language and its semantics must be defined by a technical expert/language designer, only the closed class words relevant to an application of the framework must be specified in this manner.}
Using configuration for parsing has been shown to be an effective and flexible technique by [EH04] and [KAB09b], in which traditional parse trees were generated by the configuration of property grammars. However, we go further by performing configuration on the semantic model directly, which ensures that parsed sentences are semantically sound based on the existing domain knowledge, rather than just syntactically or grammatically correct.

The hypothesis is that the proposed Cognitive Grammar/Configuration approach will fulfil the criteria as follows:

H1 The approach supports the CIM LoA (the LoA appropriate for business/domain experts) through an appropriately selected configuration model/meta-model.

H2 The approach provides a language (reading and writing) naturalness of $N_3^3 w$ or higher, according to the PEN(N)S classification scheme. That is, the language is intuitive to read and write but may require some training.

H3 The approach can express the requirements of the user, including:

H3.1 both structural and operational rules;

H3.2 a PEN(N)S classification of $E_4$ (i.e. support for higher-order logic); and

H3.3 different modalities (i.e. necessity, possibility, obligation, and permission).

H4 The approach supports the processing of existing documentation in its original form (rather than requiring it to be rewritten from the outset to conform to a prescribed format).

H5 The approach does not require any technical notations or specialised knowledge (other than the appropriate domain knowledge) for all user generated inputs.

H6 The approach provides high-quality feedback on errors, inconsistencies, and ambiguities identified in the natural language specifications.

There are three aspects to the framework that will be considered in turn: (1) the lexicon, (2) the syntactic analysis process, and (3) the semantic analysis process (and its requirements).
4.2 Lexicon

One of the most important aspects of a knowledge-based approach to NLP is its lexicon, i.e., the mapping between words and their meaning(s). Traditional lexicons typically focus on linguistic information where the syntactic category of a word—i.e., its part-of-speech as a noun, verb, etc.—is the central piece of information. Additional features are usually included as well, to support various aspects of grammar, including [All95]:

- the **number** of a word, plural or singular;
- whether a verb is in the **passive** or **active voice**;
- whether a word is in the **first**, **second**, or **third person**;
- the **tense** of a verb, e.g., *past*, *present*, or *future*;
- the **transitivity** of a verb, i.e. whether not it takes complements, which can be **intransitive** (no complements, e.g., *he ran*), **transitive** (one complement, e.g., *he ran home*), or **ditransitive** (two complements, e.g., *he lent him the car*);
- the prepositions for which a verb **sub-categorises**, e.g., in *he lent the car to him* the verb (*lent*) sub-categorises for *to*;
- and others.

A simple lexicon explicitly containing this information may appear as follows:

<table>
<thead>
<tr>
<th>Word</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>( indefinite article, 3rd person, singular )</td>
</tr>
<tr>
<td>branch</td>
<td>( noun, 3rd person, singular )</td>
</tr>
<tr>
<td>includes</td>
<td>( verb, transitive, present, active )</td>
</tr>
<tr>
<td>included</td>
<td>( verb, transitive, past, +passive, subcat = {in} )</td>
</tr>
<tr>
<td>in</td>
<td>( preposition )</td>
</tr>
<tr>
<td>is</td>
<td>( aux verb, present, 3rd person )</td>
</tr>
<tr>
<td>lent</td>
<td>( verb, ditransitive, past, +passive, subcat = {to} )</td>
</tr>
<tr>
<td>to</td>
<td>( preposition )</td>
</tr>
</tbody>
</table>

Where +passive indicates that the word can be used in the passive voice (in the right construction with a variant of the auxiliary verb *be, is, etc*.), and subcat indicates the prepositions for which the word sub-categorises.
In contrast, the lexicons of our framework are based on semantic structures, elaboration sites, and grammatical expectations, following the approach of Cognitive Grammar [Lan08; Hol93]. In our approach, a lexical entry is primarily associated with a semantic structure, i.e., an element or elements from the knowledge-base (or meta-model in the MDE context); the elaboration sites are locations in those semantic structures that are incomplete or underspecified and can be combined with other semantic structures to arrive at composite structures; and the grammatical expectations are indicators that an elaboration site should (or must) be realised at the syntactic level.

With a primary aspect of Cognitive Grammar being that ‘syntax has meaning’ [Lan08], all of the syntactic information stored in traditional lexicons may become part of the semantic structures, if that is what is desired. With the traditional information moved into the semantics, the only syntactic information left (realised by grammatical expectations) is that of word-order. This simplifies the lexicon, potentially making it much easier for users to specify lexicon entries using simple notations and features (depending on the semantic structures used), rather than needing all of the linguistic information. The definition of our lexicon structure and associated elements is presented in the following.

**Definition 4.1 (Lexicon).** A lexicon is a tuple \( l = (E, LE, evoke) \) where:

- \( E \) is a set of expressions;
- \( LE \) is a set of lexical entries; and
- \( evoke : E \rightarrow 2^{LE} \setminus \emptyset \)

**Definition 4.2 (Expression).** An expression, \( e = \langle t_1, \ldots, t_n \rangle \), is a sequence of tokens, \( t_i \), corresponding to a sequence that may appear in a token stream and that forms a phonological structure.

Since an expression is a phonological structure as described by Cognitive Grammar, each expression represents a word, phrase, or morpheme: possibly in schematic form. Moreover, the level of granularity of the tokens is left underspecified in the general framework. This means

---

3Not all elaboration sites must be associated with a grammatical expectation, and not all sites that are must be realised syntactically, e.g., where a verb can be intransitive or transitive.

4Excluding the semantic structures as they will be discussed in Section 4.4 and are simply references here.
a token could be a complete word or an individual morpheme. This makes the framework more flexible and is possible since the syntactic analysis works the same way at the word or morpheme level. For example, the following are all valid expressions in the framework:

- \(\langle\text{branch}\rangle\),
- \(\langle\text{branches}\rangle\),
- \(\langle\text{branch}, -s\rangle\),
- \(\langle\text{sending},\text{branch}\rangle\),
- \(\langle\text{send}, -ing,\text{branch}\rangle\)

Expressions also appear in the lexical entries, which represent symbols in Cognitive Grammar. As a symbol, a lexical entry comprises a phonological structure (i.e. the expression), a semantic structure (e.g. a logical predicate), and the links between the two poles. Lexical entries are defined as follows.

**Definition 4.3 (Lexical Entry).** A lexical entry is a tuple \(le = (e, ss, GE)\), such that \(e\) is an expression, \(ss\) is its associated semantic structure, and \(GE\) is an ordered set of grammatical expectations.

At this point the semantic structures are underspecified and could be of arbitrary complexity. Depending on the application, a semantic structure could be a concept from an ontology or a reference to a model element such as we use for our SBVR-based application (see Section 5.1). The grammatical expectations, which represent phonological elaboration sites, are used during syntactic analysis to catch other suggested elements (either basic lexical entries or larger compositions) to create new suggestions. Moreover, they provide the link between the phonological and semantic poles of a symbol.

This is illustrated by the abstract lexical entries displayed in Figure 4.1: the bottom parts represent the phonological poles, while the upper sections represent the semantic structures. Moreover, Figure 4.1a illustrates a simple lexical entry for, say, a noun in which a some word

\[^5\text{This will be demonstrated in Section 4.3.3}\]
is associated with some thing. A more complex lexical entry demonstrating a grammatical expectation (the *) is shown in Figure 4.1b. This form of lexical entry could represent an adjective, for example, where another word is expected to follow the adjective itself. The semantic structure of this lexical entry indicates that the thing corresponding to the word that fills the expectation (indicated by the dotted line) is in some relationship with another entity.

**Definition 4.4 (Grammatical Expectation).** A grammatical expectation is a tuple $ge = (index, type, required, es)$, where $index \in \mathbb{Z}^+$ denotes a unique identifier for the expectation within its set, $type \in \{\text{backward}, \text{internal}, \text{forward}\}$ indicates the direction of search to fill the expectation, $required \in \{0, 1\}$ is a boolean indicating whether or not the expectation must be filled for a parse to be valid, and $es$ is the elaboration site of the semantic structure that is linked to the phonological elaboration site represented by the expectation.

A total ordering of each $ge_1, ge_2 \in le.GE$, corresponding to the tokens on the stream, is defined by: $ge_1 \prec ge_2 \iff ge_1.index < ge_2.index$, subject to the following restrictions:

1. $ge_1.index = ge_2.index \iff ge_1 = ge_2$; and

2. $ge_1.index < ge_2.index \iff ge_1.type \preceq ge_2.type$, where the precedence of types is $\text{backward} \prec \text{internal} \prec \text{forward}$.

In contrast to the model of Holmqvist [Hol93], we introduced the explicit specification of required and optional expectations. If an expectation is required, the suggestion incorporating it cannot be caught unless the expectation is filled (i.e. catches another suggestion), while optional expectations do not have that restriction. Holmqvist [Hol93, Section 3.2.5] discusses
the topic of optionality of expectations in relation to completely natural language, which leads to the conclusion that all grammatical expectations should be optional to let the semantic processes decide rather than allow any syntactic filtering. However, in the context of controlled languages, specifying required and optional expectations is practically useful to help limit the size of the suggestion list and arrive at the correct interpretation more quickly. In general, we expect that the keywords (i.e. closed class words) that map to specific semantic structures will have required expectations, while the lexical entries for open class words that are added by end users will have optional expectations.

For example, Figure 4.2 presents a lexicon containing the terms illustrated as a traditional lexicon at the beginning of the section, but utilising the definitions just introduced. The semantic structures are illustrative only, simply represented as predicates in a first-order logic style notation. Note the two lexical entries for the verb includes and its passive form associated are with the same predicate, but with the grammatical expectations reversed. Moreover, note how the focus of the lexicon is on the semantic elements (e.g. the indefinite article a being associated with a quantifier), rather than all of the linguistic information included in the traditional example.

### 4.3 Syntactic Analysis

To generate syntactic parse trees, we utilise and extend the expectation-based parsing technique of Holmqvist [Hol93]. Expectation-based parsing uses the expectations of lexical entries and the word/token order of the text to suggest possible composite structures. This approach is purely syntactic as it leaves it to the semantic analysis (termed accommodation in [Hol93]) to determine the efficacy of the suggested structures. While expectations are linked to semantic structures, their semantic content is not considered when suggesting the composites. However, the syntactic analysis is intended to be performed iteratively with semantic analysis.

This approach provides a flexible means of performing syntactic analysis that is generic with respect to the semantics. Since it is driven purely by word order and the grammatical expectations of the known lexical entries, it can handle a variety of inputs including fragments of text that are not complete sentences or phrases. Whether or not any meaning can be derived
Section 4.3. Syntactic Analysis

\[ l = (E = \{ (a), (\text{branch}), (\text{includes}), (\text{included}), (\text{is}, \text{included}, \text{in}), (\text{in}), (\text{lent}, \text{to}), (\text{is}, \text{lent}, \text{to}) \}, \]

\[ LE = \{ \]

\[ le_1 = ((a, \ast), \exists x. P(x), \{(1, \text{forward}, \text{required}, P)\}), \]

\[ le_2 = ((a, \ast), \forall x. P(x), \{(1, \text{forward}, \text{required}, P)\}), \]

\[ le_3 = ((\text{branch}), \text{Branch}(x), 0), \]

\[ le_4 = ((\ast, \text{includes}, \ast), \text{Includes}(x, y), \{(1, \text{backward}, \neg \text{required}, x), \]

\[ \quad (2, \text{forward}, \neg \text{required}, y)\}), \]

\[ le_5 = ((\ast, \text{included}, \ast), \text{Past}(\text{Includes}(x, y)), \{(1, \text{backward}, \neg \text{required}, x), \]

\[ \quad (2, \text{forward}, \neg \text{required}, y)\}), \]

\[ le_6 = ((\ast, \text{is}, \text{included}, \text{in}, \ast), \text{Includes}(x, y), \{(1, \text{backward}, \neg \text{required}, y), \]

\[ \quad (2, \text{forward}, \neg \text{required}, x)\}) \]

\[ le_7 = ((\ast, \text{in}, \ast), \text{InLocation}(x, y), \{(1, \text{backward}, \text{required}, x), (2, \text{forward}, \text{required}, y)\}), \]

\[ le_8 = ((\ast, \text{lent}, \ast, \text{to}, \ast), \text{Past}(\text{Lend}(x, y, z)), \{(1, \text{backward}, \neg \text{required}, x), \]

\[ \quad (2, \text{internal}, \neg \text{required}, y), (3, \text{forward}, \neg \text{required}, z)\}) \]

\[ le_9 = ((\ast, \text{is}, \text{lent}, \text{to}, \ast), \exists x. \text{Lend}(x, y, z), \{(1, \text{backward}, \neg \text{required}, y), (2, \text{forward}, \neg \text{required}, z)\}) \}

\]

\[ \text{evoked : } E \rightarrow 2^{LE} \]

\[ (a) \mapsto \{le_1, le_2\}, (\text{branch}) \mapsto \{le_3\}, (\text{includes}) \mapsto \{le_4\}, (\text{included}) \mapsto \{le_5\}, \]

\[ (\text{is}, \text{included}, \text{in}) \mapsto \{le_6\}, (\text{in}) \mapsto \{le_7\}, (\text{lent}, \text{to}) \mapsto \{le_8\}, (\text{is}, \text{lent}, \text{to}) \mapsto \{le_9\} \]

**Figure 4.2:** Example lexicon.

from such fragments is left to the semantic analysis. Due to the incremental nature of the analysis, such fragments are routinely provided with (partial) meaning that is combined with other suggestions in later iterations.

Expectation-based parsing is a bottom-up parsing algorithm that makes use of a *suggestion list*—similar to a *chart* in chart parsing [All95, JM08]—and an *agenda* to keep track of the suggestions that have been and need to be processed, respectively. There are several notable difference to chart-parsing:

1. the addition of new suggestions/constituents is driven by the expectations of lexical entries rather than the rules of a context-free grammar;

2. due to backward expectations the search can proceed both forward and backward—if reformulated as a context-free grammar as discussed in [Hol93] many productions would be left recursive due to the conceptual nature of the expectations and their links to the semantic structure, which is undesirable;
Chapter 4. Knowledge-based Framework for CNL Understanding

3. a suggestion contains only other suggestions, unfilled expectations, and tokens (i.e. terminal symbols) rather than the many possible syntactic categories (e.g. noun, noun phrase, verb, verb phrase) and terminal symbols—this is a restriction over Cognitive Grammar in general that allows schematic elements in the parse as we are focusing on well defined vocabularies of lexical elements.

The algorithm operates on a token stream with positions in between words, with the first position in front of the first token. A total order is defined on any two positions, \( p_1, p_2 \in P \subseteq \mathbb{Z}^+ \), as \( p_1 < p_2 \iff p_1 < p_2 \). A sketch of the algorithm is provided in Figures 4.3 and 4.4.

Do until there is no input left:

1. If the agenda is empty, look-up the lexical entries of the next token (word or morpheme) and add them to the agenda.
2. Select a suggestion from the agenda (\( S \)) and add it to the suggestion list.
3. If \( S \) contains any backward expectations then, using the suggestion combination algorithm, combine \( S \) with every suggestion in the suggestion list that is within the maximum catching distance (using only the first backward suggestion if \( S \) has more than one). Add any new suggestions to the agenda.
4. If \( S \) has no backward expectations, or its backward expectations are optional (i.e. the required property is false), then combine \( S \) with every suggestion in the suggestion list with forward expectations and that is within the maximum catching distance using the suggestion combination algorithm. Add any new suggestions to the agenda.
5. Prune the suggestion list.

**Figure 4.3**: Sketch of the syntactic analysis algorithm.

For example, consider parsing the sentence

\[
\text{1 Each 2 branch 3 is included in 4 exactly one 5 local area 6}
\]

assuming the following lexicon, a maximum catching distance of zero (i.e. only neighbouring suggestions can be combined), all expectations are required, and no pruning:

- **each** \( \mapsto (\langle \text{‘each’}, \ast \rangle, \text{quantifier}, \ldots) \)
- **exactly one** \( \mapsto (\langle \text{‘exactly’}, \text{‘one’}, \ast \rangle, \text{quantifier}, \ldots) \)
- **branch** \( \mapsto (\langle \text{‘branch’}, \text{noun concept}, \ldots) \)
- **local area** \( \mapsto (\langle \text{‘local’}, \text{‘area’}, \text{noun concept}, \ldots) \)
- **is included in** \( \mapsto (\langle \ast, \text{‘is’}, \text{‘included’}, \text{‘in’}, \ast \rangle, \text{verb concept}, \ldots) \)
To incorporate a suggestion $S$ from position $p_1$ to $p_2$:

1. If $S$ is of the form: $*_1 \ldots *_i \bullet X_k \ldots X_n$, where $*_i \ldots _i$ represent (unfilled) backward expectations; $X_k \ldots X_n$ represent caught suggestions, tokens, or (unfilled) forward or internal expectations; and $\bullet$ is at position $p_1$,
   - (a) For any suggestion, $S_2$, in the suggestion list from $p_0$ to $p_2$ that is within the maximum catching distance to $p_1$ and $p_x \preceq p_1$, create a new suggestion: $*_1 \ldots *_{i-1} \bullet S_2 X_k \ldots X_n$
   - (b) If the suggestion is valid, add it to the suggestion list from $p_0$ to $p_2$ and add it to the agenda.

2. If $S$ does not contain any free, required expectations then:
   - (a) For any suggestion, $S_2$, in the suggestion list from position $p_0$ to $p_x$ such that $p_x \preceq p_1$, $S_2$ is within the maximum catching distance, and has the form: $X_1 \ldots X_n \circ *_i \ldots *_k$, where $*_i \ldots _k$ represent forward expectations; $X_1 \ldots X_n$ represent caught suggestions, tokens, or backward expectations that are optional; and $\circ$ is at $p_x$, create a new suggestion: $X_1 \ldots X_n S \circ *_{i+1} \ldots *_k$
   - (b) For any suggestion, $S_2$, in the suggestion list from position $p_0$ to $p_3$ such that $p_0 \prec p_1$, $p_2 \prec p_3$, $S_2$ is within the maximum catching distance, and has the form: $X_1 \ldots X_m \circ *_i \ldots *_k T Y_1 \ldots Y_n$, where $*_i \ldots _k$ represent internal expectations; $X_1 \ldots X_m$ represent caught suggestions, tokens, or backward expectations that are optional; $T$ is a token; $Y_1 \ldots Y_n$ represent tokens, internal or forward expectations; and $\circ$ is at $p_x$ with $p_0 \prec p_x \preceq p_1$, create a new suggestion: $X_1 \ldots X_m S \circ *_{i+1} \ldots *_k T Y_1 \ldots Y_n$
   - (c) If the suggestion is valid, add it to the suggestion list from $p_0$ to $p_2$ (or $p_3$ for the latter) and add it to the agenda.

**Figure 4.4:** Sketch of the suggestion combination algorithm.

The suggestion list and agenda initially start empty, so the first token, *each* is read and its lexical entries evoked and added to the agenda and to the suggestion list as ‘each $\circ *$’. The boundaries of what has been incorporated into a suggestion so far is denoted by ‘$\circ$’ for forward search and ‘$\bullet$’ for backwards search—one or both indicators may be excluded if there are no expectations, i.e. no search is required. Since ‘each $\circ *$’ has no backward expectations and there are no suggestions in the list ending at position 1, no additional suggestions are created.

Next *branch* is read and a new suggestion is created from its evoked lexical entry. This time the new suggestion can be combined with (i.e. caught by) an existing suggestion in step 4. As ‘each $\circ *$’ is of the form $X_1 \ldots X_n \circ *_i \ldots *_k$ (combination case 2a) ‘each $\circ$’ is said to catch ‘branch’ and the new suggestion ‘each [branch] $\circ$’ is added to the agenda and suggestion list.

At this point the state of the suggestion list is as follows:

1. `<Each>` 2. `<branch>` 3. `<is included in>` 4. `<exactly one>` 5. `<local area>` 6. `each $\circ$` 7. `branch` 8. `each branch $\circ$`
The next word, *is included in* contains both forward and backward expectations; therefore, the initial suggestion ‘* • is included in o *’ is added to the suggestion list. This leads to the introduction of two new suggestions according to step 3: ‘ • branch is included in o *’ and ‘ • each branch is included in o *’. In addition, an attempt to combine ‘each o *’ with ‘ • branch is included in o *’ is made. However, the latter contains a required expectation and, therefore, the suggestion is invalid and excluded.

After reading the next word, *exactly one*, only one new entry is added as the possible combinations with other suggestions would create invalid suggestions.

Reading the final word, however, causes several new suggestions to be created until the final parse, or parses, are found. Similar to when ‘branch’ was evoked, *local area* is caught by *exactly one* in step 4 of the algorithm. The resulting suggestion *exactly one local area* is then caught by other suggestions in step 4 when it is selected from the agenda, and so on until the agenda is empty and all possible parses have been added to the suggestion list as shown below.
Section 4.3. Syntactic Analysis

The final analysis produces two complete analyses: in (1), above, each \( \circ \ast \) catches the remainder of the sentence while in (2) it catches earlier in the analysis. This form of syntactic ambiguity is a result of the simplicity of the expectation-based parsing as all the expectations look the same from the syntactic perspective. Unlike traditional parsing methods there is no category information, e.g. in terms of parts-of-speech, to constrain the parses at this stage. While this allows for the simplicity of the lexical entries it can lead to many undesired syntactic analyses: for example, assuming that each branch is the desired result, only (2) above is a correct parse.

Even though the correct parse is determined by semantic analysis, it should be easy to see that the size of the suggestion list will grow exponentially. This is especially the case given optional expectations and a maximum catching distance greater than zero, where every possible combination of suggestions will occur. For example, each \( \circ \ast \) would be able to catch \( \ast \bullet \) is included in \( \circ \ast \), and vice versa, with a catching distance of one. Therefore, pruning the suggestion list using only syntactic information is beneficial to keep its size small and to help find the best parse quickly.

The heuristics suggested by Holmqvist [Hol93] for pruning the suggestion list make use of several metrics: catching distance, binding energy, local coverage and global coverage\(^6\), and the distance to the current token. In addition, we incorporated two lexical metrics: catch

\(^6\)In [Hol93] local coverage and global coverage are termed morpheme explanation local and morpheme explanation global, respectively.
weight and caught weight. In the following we describe the different metrics and their effects on the suggestion list.

**Catching Distance** The catching distance measures how close two suggestions are in the token stream. Rather than the simple Euclidean Distance, it is calculated using the Behaghelian Distance\(^7\)\[^8\][Hol93]. This distance measure is designed to take into account the clausal structure of language by updating the distances between expectations and the tokens in the stream as suggestions are evoked and caught. As suggestions are caught the distance to subsequent suggestions is reduced. This is demonstrated in the suggestion catching algorithm of Figure 4.4 by the propagation of the symbols ‘◦’ and ‘•’ when a suggestion is caught. More formally, the catching distance between two suggestions is defined as follows:\(^8\)

**Definition 4.5** (Catching Distance). For a suggestion, \(S_1\), from position \(p_1\) to \(p_2\) and another suggestion, \(S_2\), from \(p_3\) to \(p_4\) the catching distance, \(cd\), between an expectation, \(e\), of \(S_1\) to the suggestion \(S_2\) is:

\[
cd = \begin{cases} 
  p_1 - p_4 & \text{if } e.type = \text{backward} \\
  p_3 - p_2 & \text{if } e.type = \text{forward} \\
  p_3 - p_x & \text{if } e.type = \text{internal}
\end{cases}
\]

where \(p_x\) is the position prior to the first internal expectation of \(S_1\), if it has internal expectations.

Using this formulation of the distance, the ideal catching distance is zero (0) while negative values indicate an inversion of the direction, which is generally not allowed but could be allowed to support certain modifiers or text correction.

The suggestion list in Figure 4.5 illustrates the distances between suggestions as subclauses are constructed and caught. In particular, it demonstrates the handling of a relative clause occurring between the suggestions that should be caught by the two forward expectations of a word (i.e. the two objects of a ditransitive verb). The numbers in the each row indicate the

\(\text{So named due to its basis in Behaghel’s principle, which states that if two schemata are semantically close, then their morphemes will appear in the morpheme stream close to one another [Hol93, p. 235].}\)

\(\text{While Holmqvist [Hol93] defines the distance based on each morpheme in the stream having a position, we reformulate it to operate on the common method of the positions being between the morphemes/tokens.}\)
Figure 4.5: Suggestion list with catching distances for a sentence including a relative clause.

distances for the first expectation that would be handled by the algorithm sketched previously. For brevity, the exact construction of the relative clause has been excluded and the square brackets around the backward expectation of was indicate that it is optional—the semantic analysis would allow this expectation to be combined appropriately with John.

Without the relative clause the sentence would be simply ‘Matt lent John the car.’ This would be straightforwardly analysed to produce the final parse by combining the suggestions 1 \( \text{Matt lent John} \) \( \circ \) 3 \( \text{and the car} \) \( \circ \); however, the relative clause separating the two suggestions prevents this straightforward parse. The Behaghelian Distance performs two functions with this sort of parenthesised relative clause [Hol93]: (1) it allows the relative clause to be constructed, and (2) it protects the elements of the subclause from being caught by external suggestions. For example, 6 \( \text{his friend} \) \( \circ \) cannot be caught by 1 \( \text{Matt lent John} \) \( \circ \) as the catching distance between the two is 2. However, once the relative clause 3 \( \text{John who was his friend} \) \( \circ \) has been constructed completely, the older
suggestion \( \cdot \) Matt lent \( \cdot \) * \( \cdot \) can catch the completed relative clause. This then makes the catching distance between the new suggestion and \( \cdot \) the car \( \cdot \) zero, allowing it to be caught by the second forward expectation.

The introduction of internal expectations maintains this property. Consider an alternative formulation of the same sentence where the variant \((*, \text{lent}, *, \text{to}, *)\) (as in ‘someone lent something to someone’) is used. An example suggestion list with catching distances is displayed in Figure 4.6. As with the previous example, the distance measure prevents inappropriate suggestions from being created while the relative clause is being constructed. This is an important property to preserve, as it limits the number of suggestions added to the suggestion list.
**FIGURE 4.6:** Suggestion list with catching distances for a sentence with a relative clause and an internal expectation.
FIGURE 4.7: Suggestion list displaying the binding energy of a sentence including a
catching distance greater than zero. Interesting binding energies are indicated
as BE.

**Binding Energy** The *binding energy* of a suggestion is the summation of all catching
distances involved in the suggestion [Ho199]. The binding energy becomes important when
allowing catching distances greater than zero. That is suggestions that catch another at
distance 1 or more will have a *gap* and a binding energy greater than zero. If another word or
suggestion within the gap catches the previous suggestion it will *fill* the gap and the binding
energy of the new suggestion will be less due to the negative catching distance.

One of the main situations in which a catching distance greater than zero will occur is due to
the preference for checking backwards expectations of the current suggestion before forward
expectations of previous suggestions. Consider the situation where a verb is preceded by an
adverb, when the verb is first processed its backward expectation will try to catch a previous
suggestion. However, rather than catch the adverb directly preceding it at distance zero, it
should catch the noun before the adverb at distance 1. The forward expectation of the adverb
can then catch the noun/verb composite suggestion, filling the gap and resulting in binding
energy of zero. An example of this situation is displayed in [Figure 4.7] for the sentence ‘Matt
quickly lent John who was his friend the car.’
Section 4.3. Syntactic Analysis

To support this type of gap-filling catch\[^7\] the parsing and suggestion combination algorithms need to be modified slightly to allow for catches with a negative catching distance. In addition, the calculation of catching distance must be updated to correctly calculate the negative catching distances. The new catch distance calculation is based on the overlap between the suggestions as follows:

**Definition 4.6 (Catching Distance with Negatives).** For a suggestion, $S_1$, from position $p_1$ to $p_2$ and another suggestion, $S_2$, from $p_3$ to $p_4$ the catching distance, $CD$, between an expectation, $e$, of $S_1$ to the suggestion $S_2$ is:

$$
CD = \begin{cases} 
  p_1 - p_2 & \text{if } p_3 < p_1 < p_2 < p_4 \\
  p_1 - p_4 & \text{if } e\.type = \text{backward} \land p_1 \geq p_4 \\
  p_1 - p_4 + 1 & \text{if } e\.type = \text{backward} \land p_3 < p_1 < p_4 - 1 \land p_1 < p_2 \\
  p_3 - p_2 & \text{if } e\.type = \text{forward} \land p_2 \leq p_3 \\
  p_3 - p_2 + 1 & \text{if } e\.type = \text{forward} \land p_1 < p_3 \land p_3 + 1 < p_2 < p_4 \\
  p_3 - p_x & \text{if } e\.type = \text{internal} \land p_x \leq p_3 \\
  p_3 - p_x + 1 & \text{if } e\.type = \text{internal} \land p_1 < p_3 \land p_3 + 1 < p_x < p_4 \\
  \pm \infty & \text{otherwise}
\end{cases}
$$

where $p_x$ is the position prior to $e$, if $e$ is an *internal* expectation.

If the catching suggestion is entirely within another suggestion then the catch distance is the negative span of the catching suggestion. If there is only a partial overlap, then the catching distance is the negative span of the overlap, excluding the end tokens of the catching suggestion. However, the current algorithm sketched in Figure 4.4 does not allow these combinations; it only allows expectations to catch suggestions that strictly precede or follow the suggestion with the expectation. Therefore, the algorithm must be updated to support these cases without adding superfluous suggestions, including those containing duplicate tokens. This is done by disallowing composite suggestions in the partially overlapping cases.

\[^7\]These gap-filling catches are not the same as catching with internal expectations. Although they can be viewed similarly, internal expectations are linked to the semantics of the lexical entry they are incorporated in. In contrast, a gap-filling catch adds information on top of a suggestion that is already understandable, but skipped over some of the content of the sentence.
To incorporate a suggestion $S$ from position $p_1$ to $p_2$: 

1. If $S$ is of the form: $*_1 \ldots *_i \bullet X_k \ldots X_n$, 
   where $*_{1 \ldots i}$ represent (unfilled) backward expectations; $X_k \ldots n$ represent caught suggestions, 
   tokens, or (unfilled) forward or internal expectations; and $\bullet$ is at position $p_1$, 
   
   (a) For any suggestion, $S_2$, in the suggestion list from $p_0$ to $p_x$ that is within the maximum catching distance to $S_1$ and $p_0 < p_1$ and $(p_x \leq p_1 \lor p_1 \lor p_0 < p_x \lor p_1 \lor p_x - 1 \land p_x < p_2)$, create a new suggestion: $*_1 \ldots *_{i-1} \bullet S_2 X_k \ldots X_n$ 
   
   (b) If the suggestion is valid, with a binding energy $\geq 0$, add it to the suggestion list from $p_0$ to $p_2$ and add it to the agenda.

2. If $S$ does not contain any free, required expectations then:
   
   (a) For any suggestion, $S_2$, in the suggestion list from position $p_0$ to $p_x$ such that $p_x < p_2 \land (p_x \leq p_1 \lor p_1 < p_0 \lor p_0 \lor p_1 \lor p_x - 1)$, $S_2$ is within the maximum catching distance, and has the form: $X_1 \ldots X_n \bigcirc *_{i+1} \ldots *_k$, 
   where $*_{1 \ldots k}$ represent forward expectations; $X_1 \ldots n$ represent caught suggestions, 
   tokens, or backward expectations that are optional; and $\bigcirc$ is at $p_x$, 
   create a new suggestion: $X_1 \ldots X_n S \bigcirc *_{i+1} \ldots *_k$ 
   
   (b) For any suggestion, $S_2$, in the suggestion list from position $p_0$ to $p_3$ such that $p_3 < p_2 \land p_2 < p_3 \land (p_2 \leq p_1 \lor p_1 < p_0 \lor p_0 \lor p_1 \lor p_3 - 1)$, $S_2$ is within the maximum catching distance, and has the form: $X_1 \ldots X_m \bigcirc *_{i+1} \ldots *_k T Y_1 \ldots Y_n$, 
   where $*_{1 \ldots k}$ represent internal expectations; $X_1 \ldots m$ represent caught suggestions, 
   tokens, or backward expectations that are optional; $T$ is a token; $Y_1 \ldots n$ represent tokens, 
   internal or forward expectations; and $\bigcirc$ is at $p_x$ with $p_0 < p_x < p_2$, 
   create a new suggestion: $X_1 \ldots X_m S \bigcirc *_{i+1} \ldots *_k T Y_1 \ldots Y_n$ 
   
   (c) If the suggestion is valid, with a binding energy $\geq 0$, add it to the suggestion list from $p_0$ to $p_2$ (or $p_3$ for the latter) and add it to the agenda.

Figure 4.8: Suggestion combination algorithm supporting negative catching distances.

where the ends of the suggestions are shared and by ensuring that the new binding energy is 
non-negative. The updated suggestion combination algorithm is displayed in [Figure 4.8]. It is 
important to note that for internal expectations, the suggestions being caught cannot extend 
past the next token in the expression. This is because the tokens are typically markers, so 
suggestions should not occur across these markers without involving them. Most cases of 
icorrect suggestions are excluded by these constraints, see [Figure 4.9] for common cases; 
however, there others that may still be admitted. These are avoided using the next metric, 
local coverage.

**Local Coverage** The local coverage of a suggestion measures how many of the tokens in 
the stream are used by the suggestion within its span [Hol93]. It is defined as the ratio of 
tokens involved in the suggestion to the words spanned by the suggestion. This ratio allows the 
detection of gaps in a suggestion as well as the detection of tokens used more than once. That
Figure 4.9: Different cases of the negative catching distances showing the catching distance \((CD)\) and binding energy \((BE)\). They are shown for each sub-row, e.g. (1.1), being caught by its main row, e.g. (1), independently of one another. Only cases 1.1, 1.2, and 2.2 are valid catches.

Note that the resulting binding energy for case 2.2 would be 1, as the new suggestion would incorporate two suggestions with a binding energy of 1 each and only \(-1\) catching distance for the new suggestion, i.e. \(1 + 1 - 1 = 1\)

is, a local coverage of \(<1\) indicates gaps, while \(>1\) indicates duplicated tokens. Of course, the ideal local coverage is 1, where each token that occurs in the stream within the span of the suggestion is incorporated in the suggestion exactly once. For example, the suggestions

\[
\begin{align*}
\text{Matt} & \rightarrow \text{2} \\
\text{Matt lent} & \rightarrow \text{3}
\end{align*}
\]

have a local coverage of one, while

\[
\begin{align*}
\text{Matt . . . lent} & \rightarrow \text{4}
\end{align*}
\]

has a local coverage of \(\frac{2}{3}\). On the other hand, suggestions with duplicate tokens (i.e. a local coverage \(>1\)) are considered invalid.

The local coverage differs from the binding energy in that if there are no gaps in a suggestion the local coverage will always be 1, while the binding energy may be \(>0\) if multiple suggestions have been caught at distances greater than zero. This can be seen in the examples of Figure 4.10. Example (1) in the figure demonstrates the binding energy and local coverage agreeing when the a gap in a larger suggestion is filled. In example (2) the local coverage and binding energy disagree since there are two suggestions with binding energy 1 (2.1 and 2.2),
but their overlap results in a catching distance of $-1$. Therefore, although the local coverage indicates no gaps, the binding energy of (2.3) is 1. This cannot be further combined with (2.4), for example, as it would result in a local coverage greater than 1 (see 2.5) due to duplicates. The third example, particularly the combination of (3.3) with (3.5) to produce (3.6), simply demonstrates the same behaviour for suggestions with internal expectations.

Finally, example (4) of Figure 4.10 illustrates a suggestion containing duplicates and a gap, which is not reflected in its binding energy or local coverage [Hol99]. However, such a construction is unlikely to occur in practice. For example, even the ill-formed sentences

*Matt lent quickly to John the car. (using words as tokens)
*Matt lend <+ed> to John the car. (using morphemes as tokens)

would only give rise to the construction in (4) if all of the expectations of lent were optional. While Holmqvist [Hol93, pg. 226] indicates that all of the expectations of verbs are optional to allow sentences such as To lend [something [to someone]] is a nice gesture, this is unlikely to occur in the context of a controlled language. On the other hand, if such a situation were to occur, the semantic analysis should be able to quickly eliminate the incorrect suggestion. Alternatively, an explicit check could be performed to exclude duplicates within a suggestion if such constructions were common. Lastly, we are yet to introduce some additional mechanisms that will help to avoid this situation.

10 Allowing $\bullet$ to $\circ \to \circ^*$ to catch $\bullet$, $\circ \to \circ^*$ creating $\bullet$ lent/t $\circ \to \circ^*$ to form $\bullet$ lent/t to $\circ^*$. This could then catch $\bullet$ Matt lend/t to $\circ^*$ to form $\bullet$ lent/t (lend/t to) John $\circ^*$ to $\circ^*$. 74
Section 4.3. Syntactic Analysis

![Diagram showing token stream order and syntactic analysis steps with CD values for local coverage vs. binding energy suggestions.]

**Figure 4.10:** Local coverage vs. binding energy of suggestions.
Catch and Caught Weights  We extend these heuristics with two lexical heuristics: (1) catch weight, and (2) caught weight. These weights are functions (in the range \{0..1\}) associated with lexical entries that are evaluated when a catch occurs. The catch weight provides a weighting from the perspective of the catching suggestion, i.e., an estimate of how compatible it is to the suggestion that it has caught. On the other hand, the caught weight is from the perspective of the suggestion being caught, representing an estimate of how compatible it is to the catching suggestion. The symmetry of these two weights was introduced for practical reasons; sometimes it makes more conceptual sense to define the weight in terms of what is doing the catching, other times it is clearer to define it based on the suggestion being caught.

Since the expectation-based parsing is purely syntactic, i.e., based entirely on word-order information, it can potentially lead to suggestions that are obviously wrong in the context of the restricted semantics of a controlled language. Therefore, these weights allow the optimisation of the parsing process by indirectly incorporating some semantic information. The weights can be used to prefer one suggestion over another, potentially pruning a suggestion altogether if the weight is below a threshold. This improves the ability for the correct interpretation to be found early.

For example, the catch weight could be defined to explicitly check for duplicates in the suggestion and prevent such suggestions from being created. Alternatively, the catch weight for the prepositional ‘to’ lexical entry (\(*, \text{‘to’}, *\)) could be defined such that its right expectation should only be able to catch location type suggestions\(^{12}\) thereby preventing the suggestion

\[
\text{Matt lend/t (lend/t \ldots \text{to}) John } \quad \Rightarrow \quad 6
\]

(in the example above) from being created.

Finally, in a more general context, the weights could be determined through statistical analysis of a corpus: for example, word collocation information. This would help to perform disambiguation early based on statistical preferences in a similar fashion to statistical semantic preferences that can be used by more traditional parsers [All95].

Each of the metrics discussed so far has a specific role to play in deciding whether or not a suggestion is valid:

\(^{11}\)In addition, it is left open for future work to extend it into a probabilistic model, which will be important for more general language processing.

\(^{12}\)How a location type suggestion is defined or determined would be specific to the semantics of the language being defined.
Section 4.3. Syntactic Analysis

catching distance: if the catching distance is greater than a specified distance then the catch cannot occur;

binding energy: negative binding energies are not allowed;

local coverage: a local coverage greater than 1 is not allowed;

catch/caught weight: weights below a specified threshold exclude the suggestion.

These metrics help to keep the suggestion list small by preventing certain suggestions from being added to the list in the first place. However, there are two more metrics that are used in conjunction with the previous four to rank and prune suggestions throughout the analysis of a token stream. They are the distance to the current token and the global coverage.

Distance to Current Token The distance from the current token to a suggestion differs from the catching distance between two suggestions as it is simply the linear distance between the current token and the suggestion. A distance of zero indicates that the suggestion incorporates the current token. That is, for the current token $T$ from positions $p_{t1}$ to $p_{t2}$ and a suggestion from $p_{s1}$ to $p_{s2}$, the current distance is $p_{t1} - p_{s2} + 1$. Effectively, each time a token is processed the distance to the suggestions currently in the list increases by 1, while the distance to those suggestions incorporating the token is zero.

While closer distances are potentially indicative of being more useful, we cannot simply discard suggestions once they reach a certain distance or greater [Ho93]. However, keeping all of the early suggestions around over long distances is potentially inefficient and unnecessary. For example, in the suggestion list of Figure 4.11, each branch could potentially be pruned off since its correct catch (i.e., each branch) is found early. This would prevent attempts to catch all of the larger suggestions following it and, in particular, would prevent the suggestion each branch is included in exactly one local area from being formed when the current distance to each is four.

In contrast, nested relative clauses potentially require early suggestions to be kept for a long time until the subclause is complete and can be caught by the early suggestion. For example,
The country that is of a branch at a reasonable current distance of three. However, due to the nested relative clauses, the suggestion would not catch the remainder of the sentence (i.e. the operating country that is of the operating company that . . .) until a current distance of 14. Moreover, the suggestions for the other relative clauses—the operating company that . . . would not catch the appropriate suggestion until current distances of 11, 7, and 3, respectively. In this situation, we do not want to eliminate the earlier suggestions as they would no longer be present when they are needed.

As a result there is no straightforward cutoff point for when a suggestion should be pruned based solely on the distance to the current morpheme. We will discuss the strategy we use for this in Section 4.3.1.
Global Coverage  While the local coverage measures the completeness of a suggestion with respect to its own span, the global coverage measures how many tokens are used by the suggestion with respect to the stream of tokens processed up to the current point. Specifically, global coverage is the number of tokens incorporated in the suggestion divided by the number of tokens encountered up to the current point [Hol93]. Therefore, the ideal global coverage for a complete parse of a sentence is one, i.e., all the tokens of the stream are used in the suggestion.

Similarly to the current distance, global coverage cannot be used to prune suggestions outright by itself. Doing so would most likely remove suggestions that are needed to complete the analysis later; however, keeping very small suggestions after they have been incorporated into larger suggestions potentially maintains information that is no longer necessary. For example, in Figure 4.11 once each \( \circ \) combines with branch to form each branch \( \circ \), the individual suggestions could potentially be pruned off after global coverage drops below a certain threshold, say \( \leq 0.5 \). This would mean they would be removed from the suggestion list once the processing reaches the <is included in> token. However, such a high threshold might start to remove expectations before they can catch, or be caught by, the appropriate suggestion if only global coverage were considered. In this case, exactly one \( \circ \) would be removed from the suggestion list immediately before it could catch local area and be incorporated into each branch is included in \( \circ \). In the following we discuss how the heuristics can be used together to prune the suggestion list of suggestions that are most likely no longer needed for processing.

4.3.1  Pruning Strategy

Providing a pruning strategy is important to keep the suggestion list size small and to aid in arriving at the correct interpretation quickly. The pruning strategy can take into account many factors including the different metrics (catching distance, binding energy, local/global coverage, catch/caught weight, and current token distance) as well as other factors such as the size of the suggestion list and whether or not a suggestion has been successfully analysed semantically.
In [Hol93] Holmqvist considers the combinations of salience (i.e. how successful the suggestion was during the semantic analysis), global coverage, and distance (to the current token) and what it means in terms of the quality of the suggestion. For example, high salience, high global coverage, and a distance of zero indicates a good final parse or good mid-parses, while low salience, intermediate global coverage and high distance should probably be pruned off.

However, in the development of controlled languages (or even applications for unrestricted natural language) additional considerations must be taken into account, such as the effect on distance due to the choice of word or morpheme level tokens; morpheme level tokens may mean that acceptable distances are larger.

While the exact pruning strategy is application specific, it is safe to say that if a suggestion fails the semantic analysis then it should be removed from the suggestion list. Moreover, catch and caught weights below an application specific threshold can be safely removed from the list. The other two important factors for pruning are global coverage and distance, which must be considered together. For example, a low global coverage and a long distance (as for each from Figure 4.11) indicates that the a suggestion could be pruned, while low global coverage and a short distance would not be pruned (preventing, for example, exactly one of Figure 4.11 from being removed to soon). Although, exactly what is meant by low global coverage and short or long distance is application specific.

The other metrics can be used to order the suggestions and, along with a limit to the size of the suggestion list, prune the suggestions off the bottom of the list. In general the preferences for the different metrics are as follows:

- **catching distance**: shorter catching distances are preferred (zero is best)
- **distance to current token**: shorter distances are preferred
- **binding energy**: smaller binding energies are preferred (zero is best)
- **local/global coverage**: higher ($\approx 1.0$) is preferable
- **catch/caught weight**: higher ($\approx 1.0$) is preferred
- **salience**: higher ($\approx 1.0$) is preferred

We will show a concrete pruning strategy for the application described in Chapter 5.
4.3.2 Parse Trees

The expectation-driven parsing process described so far leads to parse trees that differ from those of traditional parsing: both constituent parse trees and dependency trees. Figure 4.12 displays parse trees produced by the syntactic analysis of ‘Each branch is included in exactly one local area.’ It compares and contrasts the tree produced by expectation-based parsing (Figure 4.12a) against a constituent tree (Figure 4.12b) and a dependency tree (Figure 4.12c).[13]

Like a constituency tree, an expectation-based tree shows how words and phrases combine to form larger phrases, but like a dependency tree it more directly represents the relationships between words. Unlike a constituency tree, the constituents are the words and phrases themselves,[14] rather than an abstract representation based on parts-of-speech. In contrast to dependency trees, the relationships between words are of a single type (catches) with the actual relationships determined by semantic analysis, which may consist of more than just grammatical relations. While parts-of-speech and grammatical relations can be represented in

---

13 Note that the complexity of the multi-word expressions that would normally be present in the constituency and dependency trees has been simplified.

14 The ellipses in the expectation-based parse tree represent omission of the caught expression in the larger phrase and are used for brevity.
the semantics (either directly or through derivation), this information is not required for the construction of expectation-based parse trees.

Furthermore, the direction of the dependencies of the expectation-based tree (i.e. the lines with arrows) are not always in the same direction as a in a dependency tree (e.g. ‘each’ → ‘branch’ rather than ‘branch’ → ‘each’) as they go from catching expectation to caught word. While the placement and direction of the expectations is relatively arbitrary, there is a cognitive and practical basis for specifying the expectations in the way that we have. For example, a noun could have a leftward expectation for a quantifier (possibly a determiner or adjective), which would make the catch in the same direction as in dependency grammars. However, a noun can more often appear standalone, whereas a quantifier always quantifies something (even if has been left implicit in a sentence through ellipsis, e.g. ‘each’ can be used by itself as a type of anaphora); therefore, it makes more sense for the quantifier (determiner or adjective) to include the expectation while leaving the noun free of a preceding expectation. Moreover, this has the practical benefit of not requiring the evocation of two variants of each noun, one with the expectation and one without, improving the efficiency of the parsing process through a reduction in the number of suggestions while at the same time allowing restrictions such as requiring that expectation of keywords in controlled languages must be filled.

4.3.3 Tokenisation (and Internal Expectations)

So far we have not thoroughly discussed the nature of the token stream in the syntactic analysis: avoiding, in general, terms such as ‘word’, ‘sentence’, or ‘morpheme’ and using the generic ‘token’ instead. However, we have suggested that the structure of the token stream will impact any particular application of the framework presented here.

In the preceding examples we have used word-level tokenisation, including multi-word expressions. This form was chosen as it simplifies the examples somewhat and is in line with our target application (presented in Chapter 5). This approach, however, requires an assumption that the text to be analysed has already been pre-processed such that the individual words and, more importantly, noun phrases and verb phrases of multi-word expressions have already been identified to achieve the correct tokenisation. For example, in
the tokenisation indicates that the terms is included in, exactly one, and local area have already been identified as such. This is not problematic as such, since this approach is knowledge-based and makes extensive use of lexical entries, which could be defined as follows:

\[
\begin{align*}
\text{exactly one} & \mapsto ((\text{`exactly'}, \text{`one'}, \ast), \ldots) \\
\text{local area} & \mapsto ((\text{`local'}, \text{`area'}), \ldots) \\
\text{is included in} & \mapsto ((\ast, \text{`is'}, \text{`included'}, \text{`in'}, \ast), \ldots)
\end{align*}
\]

Utilising this knowledge during tokenisation, therefore, would allow the grouping of the more primitive tokens/words (separated by spaces) into the larger tokens of the example token stream. The “problem” arises when there are additional lexical entries. For example:

\[
\begin{align*}
\text{exactly} & \mapsto ((\text{`exactly'}, \ast, \ast), \ldots) \\
\text{local} & \mapsto ((\text{`local'}, \ast), \ldots) \\
\text{is} & \mapsto ((\ast, \text{`is'}, \ast), \ldots) \\
\text{included} & \mapsto ((\ast, \text{`included'}, \ast), \ldots) \\
\text{in} & \mapsto ((\ast, \text{`in'}, \ast), \ldots)
\end{align*}
\]

With the additional lexical entries, the tokenisation would become ambiguous. However, this is not really a “problem” per se, given that the illustrated tokenisation is for example purposes: that is, unnecessary details have been simplified or left out completely. The tokenisation could indeed be displayed as:

\[
\begin{align*}
1 \text{ `<Each>` } 2 \text{ `<branch>` } 3 \text{ `<is included in>` } 4 \text{ `<exactly one>` } 5 \text{ `<local area>` } 6 .
\end{align*}
\]

which is the required tokenisation to allow each lexical entry to be evoked. It is trivial to extend the algorithm provided in Figures 4.3 and 4.8 to handle the lexical ambiguities involved with multi-word expressions. When a token evokes a lexical entry it includes those entries for which the token is the first word of a multi-word expression. The suggestions created in this way are checked for being extended as each token is processed in the same way that expectations are checked. In fact, extending a multi-word expression can be considered as
a forward expectation that is only valid for a specific token at a catching distance of zero. However, by treating multi-word expressions specially some optimisations can be performed.

Figure 4.13 displays the partial suggestion list for the analysis of *is included in* when the individual words have lexical entries as well as the multi-word expression. After *in* is processed, there are two alternative forms of creating what appear to be the same suggestion: (1) and (2) in the figure. The parse trees associated with these parses are quite different though, as shown in Figure 4.14. In the former, the passive construction of *includes* must be constructed, while in the latter it is given by the lexical entry (assuming appropriate semantics for both). In this case, the resultant analysis is the same, except that the multi-word expression provides a shortcut to the semantics. This represents the embedding of the expression’s conventionality in the language as described by Langacker [Lan08]. This is particularly useful in the context of controlled and domain specific languages as it allows terms that are conventional for the domain to be easily represented in the lexicon. Moreover, it means that the constructed suggestion can be removed from the suggestion list (or at least given much less weight) when the complete multi-word expression is found. In a more unrestricted language, this allows conventional expressions with semantics different to those constructed from the smaller constituents to override the meaning of the constructed suggestion. In this case, maintaining the constructed expression may be important in the event the conventional expression does not lead to a valid interpretation after the semantic analysis as the less conventional constructed meaning was used.

By supporting multi-word expressions through both the lexicon and syntactic/semantic analysis, these different situations can be accounted for. The analysis of *exactly one* and *local area* would be similar. In the case of *local area*, this analysis allows the common domain term *local area* to be simply analysed, while still allowing the adjective *local* (i.e. ⟨‘local’, *⟩) to catch other terms to create, possibly novel, compound terms such as *local branch, local anaesthetic, local beach*, etc.

When suggestions involving multi-word expression lexical entries are processed, some optimisations can be made. Apart from removing the constructed suggestion once the lexical entry’s expression is complete, the previous suggestions for the multi-word lexical entry with a shorter span can be removed. This is demonstrated in Figure 4.13 by the struck out
Section 4.3. Syntactic Analysis

Figure 4.13: Expectation-based analysis of multi-word expressions.

Figure 4.14: Expectation-based parse trees for multi-word expressions: (a) analysis with separate lexical entries, and (b) analysis using the single lexical entry.
suggestions. In addition, if a token does not match the next token of a multi-word expression, the suggestions involving that lexical entry can be removed. This is particularly important as the size of the lexicon grows. For example, if the lexicon included additional entries for other conventional passive constructions, such as *is located in* and *is base for*, once the second word of the expression has been encountered, the non-matching suggestions can be removed from the suggestion list. Alternatively, a particular application may incorporate the analysis of multi-word expressions into the tokenisation using the lexicon. This may be appropriate if the lexical entry always overrides a constructed suggestion for the application. Finally, forward expectations cannot catch anything until the suggestion has been extended to reach the expectation, just as multiple expectations must catch suggestions in order.

A similar analysis occurs for lexical entries with internal expectations. For example, the suggestion \( \text{a * lent} \circ \text{to} \rightarrow \text{b} \) cannot be complete until a *to has been processed and used to extend the suggestion. However, its internal expectation(s) can catch suggestions in the meantime as any suggestions caught up to that point must obey the constraint that they occur within the gap. This is shown in Figure 4.6, although it was not taken as given at the time. Moreover, catching the internal suggestions before the following tokens are used to extend the suggestion is potentially necessary to ensure that they are caught before being pruned. Of course, internal suggestions cannot be caught if multiple tokens are expected before the expectation, as in *is lent to . . . by* (i.e. \( \langle*, 'is', 'lent', 'to', *, 'by', *\rangle \)) which would require * . . . is lent to * by * \( \rightarrow \) before the internal suggestion could catch anything. The difference between lexical entries with internal expectations and other multi-word expressions is that the tokenisation process cannot take them into account as easily. Such a tokenisation would require the entire sentence to be available from the beginning (preventing a purely incremental approach) and it would produce overlapping tokens, for example:

\[
1 \text{<Matt>} \quad 2 \text{<lent>} \quad 3 \text{<the>} \quad 4 \text{<car>} \quad 5 \text{to} \quad 6 \text{<John>} \quad 7 .
\]

While nested tokenisation such as this can be performed, additional care must be taken to preserve the constraints of the approach. For example, the current position of the token stream—is it 6 once *lent . . . to* is evoked since \(2 \star \text{lent} \circ \text{to} \rightarrow 6 \)?—and catching distances need to be considered carefully—are they negative, or should they still be positive? Exactly how situations like this are handled will be shown in Section 5.4.
Section 4.3. Syntactic Analysis

Now that we have considered multi-word expressions in the lexicon for word-level token streams, it should be clear that handling token streams at the morpheme level requires no additional features (apart from presupposing some method of morphological analysis). For example, given the morpheme stream

1 <Each> 2 <branch> 3 <is> 4 <includes> 5 <ed> 6 <in> 7 <exactly> 8 <one> 9
9 <local> 10 <area> 11 .

and the lexicon

```
each        → ((‘each’, *), . . .)
branch      → ((‘branch’), . . .)
exactly one → ((‘exactly’, ‘one’, *), . . .)
local area  → ((‘local’, ‘area’), . . .)
is included in → ((*, ‘is’, ‘includes’, ‘-ed’, ‘in’, *), . . .)
-ed         → ((*, ‘-ed’), . . .)
```

the syntactic analysis of the sentence would proceed exactly as previously described. The evoked suffix -ed may try to catch another suggestion, but will be controlled by the standard pruning mechanisms and can be eliminated by the expansion of the *is includes-ed in* suggestion once it covers the morpheme.

Finally, lexical multi-word expressions that allow other terms to occur within them must be considered. This is important to allow negation, adverbs, and modal verbs in multi-word expressions containing auxiliary verbs. For example, the negation of *is included in* is represented by *is not included in*. Having to include such forms in the lexicon would be tedious. While an application can implement special handling for this situation if it allows it, it can be handled within the framework already provided. By allowing the extension of a suggestion for a multi-word expression to skip tokens (up to a limit similar to the maximum catching distance), such constructs can be supported. Moreover, when this occurs, the suggestion can be given a binding energy and local coverage as if a catch at distance greater than one had been made. For example, [Figure 4.15] displays the analysis of a negated multi-word expression.

15Given our emphasis on creating controlled languages, it is possible that these constructions may not be permitted.
Unfortunately, allowing such situations may lead to excessively large suggestion lists. Therefore, additional features can be added to the lexicon to allow only certain locations between tokens to be skipped. This is similar to an optional expectation except that the term cannot catch the skipped tokens/suggestions. For example, the lexical entry for \textit{is included in} could be

\[\text{is included in} \rightarrow (\langle \ast, \text{'is'}, [], \text{'included'}, \text{'in'}, \ast \rangle, \ldots)\]

permitting negation or adverbs to appear between ‘is’ and ‘included’, but not between ‘included’ and ‘in’. We could also allow restrictions to specific terms, for example:

\[\text{is included in} \rightarrow (\langle \ast, \text{'is'}, [\text{'not'}], \text{'included'}, \text{'in'}, \ast \rangle, \ldots)\]
Section 4.4. Semantic Analysis

would allow only negation, as represented by *not*, to appear inside the multi-word expression. All of this is supported by the basic framework described earlier in this chapter. It is up to the language designer/implementer to decide if, or how, these features are to be utilised.

4.4 Semantic Analysis

The suggestions produced by the syntactic analysis are forwarded to the semantic analysis process to determine whether or not they are admissible in the given semantics. That is, the semantic analysis process determines if there is an interpretation that conforms to the semantic constraints of the language and application. In the model of Cognitive Grammar developed by Holmqvist [Hol93] the semantic analysis process, called *semantic accommodation*, involves numerous complex processes to arrive at an interpretation (or not) of a suggestion. We have simplified the process by making use of knowledge-based configuration as model search, which subsumes most of the complex mental processes described by [Lan08] and modelled in [Hol93]. As a result many of the disambiguation processes required for semantic analysis—such as word-sense disambiguation, co-reference (or anaphora) resolution, and scope resolution—are unified as a single configuration process.

In order to perform the semantic analysis of suggestions through model search, three things need to be defined [STMS98]:

1. *Configuration model knowledge*: a meta-model consisting of the basic entities, relationships, and constraints that determine a valid configuration;

2. *Configuration solution knowledge*: the partial model (configuration) for which a solution needs to be found; and

3. *Configuration requirements knowledge*: any additional constraints on the valid solutions that are not strictly part of the meta-model (often user requirements in the general use of configuration).
Chapter 4. Knowledge-based Framework for CNL Understanding

The meta-model specifies the a priori semantics of the controlled language being defined, while the partial model and additional constraints are derived from the syntactic analysis of the text.

To help explain and demonstrate the general framework, the following presents the formulation of the semantics based on a simple version of Conceptual Graphs [Sow08]. Conceptual Graphs are a graphical, graph-based notation for first-order logic whose primary operators are existential quantification, conjunction, and negation.

4.4.1 Configuration Model Knowledge

For model search, the configuration model knowledge consists of two meta-models [KDA10]: (1) a constrained meta-model, and (2) a relaxed meta-model. The constrained meta-model is the meta-model that serves as the final target of the transformation with all of the constraints that a valid configuration must fulfil. The relaxed meta-model is simply the constrained meta-model without many of the constraints. Specifically, minimum cardinalities are reduced to zero, attributes are optional, and any invariants specified for the meta-model are removed [KDA10]. This relaxation is a straightforward process and can easily be performed using a model transformation.

The constrained and relaxed meta-models for the Conceptual Graph semantics are displayed in Figures 4.16 and 4.17, respectively. Note that the maximum cardinalities are unchanged in the relaxed version and that the restriction placed on the type attribute in Relation has been removed due to the relaxation making it equivalent to the attribute’s original definition in Node.

The form of Conceptual Graphs supported by the constrained meta-model is quite simple. Like all graphs, Conceptual Graphs consist of arcs and nodes represented by Arc and Node, respectively. In general, nodes may be connected to other nodes through any number of inArcs and outArcs. On the other hand, Arcs must have one each of a source node and a destination (abbrev. dest) node. Since Conceptual Graphs represent a typed version of logic, all nodes may have a type label provided by the type attribute on Node.
Section 4.4. Semantic Analysis

Figure 4.16: Simple Conceptual Graphs (constrained) meta-model in UML notation.

Figure 4.17: Simple Conceptual Graphs relaxed meta-model in UML notation.
Nodes can be one of three types: a *conceptual relation* (or simply *relation*), a *concept*, or a *context*. A *Concept* node represents an *existentially quantified variable*, possibly constrained by a type. Moreover, a *Concept* may indicate a specific referent if a value for its *name* attribute is specified. The following illustrates the different combinations of attribute values allowed for a concept alongside their Conceptual Graph notation and their equivalent in (fragments of) typed first-order logic.

<table>
<thead>
<tr>
<th>UML Object Instance</th>
<th>Conceptual Graph notation</th>
<th>First-Order Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>: Concept</td>
<td>Person: Matt</td>
<td>Person(Matt)</td>
</tr>
<tr>
<td>type = 'Person'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>name = 'Matt'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>: Concept</td>
<td>Person</td>
<td>∃x : Person</td>
</tr>
<tr>
<td>type = 'Person'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>name = null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>: Concept</td>
<td>: Matt</td>
<td>Matt</td>
</tr>
<tr>
<td>type = null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>name = 'Matt'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Relation nodes represent *predicates* in first-order logic and, therefore, must indicate a type. Moreover, the connected arcs specify the arguments of the predicate. The single *inArc* of a *Relation* (note the restriction) specifies the first argument, while the *outArcs* indicate the remaining arguments in order. In the simple version of Conceptual Graphs specified here we have limited relations to 3 arguments, i.e., to ternary relations at most. The following examples show unary, binary, and ternary relations as an instance model, Conceptual Graph notation, and first-order logic.
The Context nodes represent sub-graphs of different types. For the example, we support only negated contexts with the Negation sub-type. Contexts may contain nodes, including other contexts. Moreover, context nodes can be involved in relations. Therefore, with the definition of other types of Context, various logical operators and scopes could be represented, such as: if-and-only-if and proposition contexts. In the following we demonstrate the notation for contexts and their first-order logic equivalents.
Another form of connection that can exist in a Conceptual Graph is a coreference link, represented by the Coreference class. A Coreference can only occur between two Concepts and indicates that they refer to the same entity. For example,

Finally, there are a couple of constraints to ensure that the graphs are created correctly: i.e., each Arc must connect a relation to a non-relation node. This is ensured by the pair of invariants attached to the Arc class (written in OCL): nodesNotTheSameType, which ensures that the source and dest nodes of an Arc of different types from the meta-model, and oneNodeMustBeARelation, which ensures that either the source or dest node is a Relation.

When interpreting a Conceptual Graph in first-order logic, there is an implicit conjunction between nodes within a context (and the graph) [Sow08]. For example, Figure 4.18 shows a simple, but complete, Conceptual Graph and its first-order logic equivalent.

The semantics defined by this meta-model are very simple, amounting to specifying syntactically valid Conceptual Graphs. Moreover, as Conceptual Graphs are a representation for
first-order logic, the semantics allow for very general models to be instantiated. The result
is that the semantics defined here allow one or more logical representations of a sentence to
be created, but is limited in determining which form is *correct* in the case of ambiguities.
However, the limitations of this simple example are not limitations of the overall approach.
For example, a domain specific meta-model may be much more restrictive, incorporating
many more constraints that allow increased disambiguation potential. Chapter 5 will further
demonstrate the framework on a more capable set of semantics.

4.4.2 Configuration Solution Knowledge

The configuration solution knowledge consists of the partial model that will be completed by
the configuration/search process to arrive at the interpretation of a sentence. The partial model
primarily contains the elements related to the suggestions for which we are attempting to
determine their semantic viability. In addition, the partial model may include model elements
from the lexicon and the current context. While the former is created during the definition of
the lexicon, the latter is constructed during the processing of a text and its scope may differ
between applications. For example, in a restrictive application the scope of the context may
be limited to the current sentence, while in other applications the full discourse may be within
the scope of the context.

Creating the partial model from the suggestions requires that a set of mapping rules be defined
between them. At their simplest, these rules just reference the semantic element of the lexical
entries evoked by the suggestions. However, depending on the application and the meta-model,
more complex mappings may be required to create additional supporting elements necessary
for the interpretation of a sentence but otherwise do not belong in the lexicon.\footnote{In this simple example the latter is not necessary; however, additional elements are required by the SBVR semantics used in Chapter 5}

Consider the following lexicon, in which we use our simple Conceptual Graphs as the
associated semantic structures:

\[
\begin{align*}
\text{a, some} & \mapsto \left(\{\text{‘a’, *}, \text{‘some’, *}\}, *, \right)
\cup \\
\text{branch} & \mapsto \left(\{\text{‘branch’}, \text{Branch}, \{\}\right)
\end{align*}
\]
local area \mapsto (\{\text{\'local\}', \text{\'area\}', LocalArea}\}, \{\})

is included in \mapsto \begin{pmatrix}
\langle *, \text{\'is\}', \text{\'included\}', \text{\'in\}', *\rangle, \\
\text{Includes} \rightarrow \text{\#1}, \\
\{ \langle 1, \text{backward}, \text{required}, \text{Relation.outArc[0].dest} \rangle, \\
\langle 2, \text{forward}, \text{required}, \text{Relation.inArc.source} \rangle \} 
\end{pmatrix}

This lexicon defines four lexical entries. The first defines a pair of synonyms for existential quantification, namely \textit{a} and \textit{some}. Since existential quantification is represented by a Concept in the Conceptual Graph meta-model, the semantic structure \(\*\) illustrates a Concept that is missing its \textit{type}. The single expectation is then linked to the \textit{type} attribute, i.e. the elaboration site of the semantic structure, to indicate that it should assign the \textit{type} when the expectation is filled.

The next two entries are very simple, indicating that the \textit{branch} is associated with the \textbf{Branch} concept, and \textit{local area} is associated with the \textbf{LocalArea} concept. Although they are displayed within concept boxes, the semantic structures could be considered to really represent the \textit{type} name itself. This could potentially allow higher-order semantics for sentences that talk about the concept itself, e.g. Concept: Branch, in which case it would specify the value of the \textit{name} attribute of a (meta-)Concept instead.

The last lexicon entry represents the passive verb phrase \textit{is included in} and maps it to the \textbf{Relation Includes}. The semantic elaboration site of the backward expectation is actually the second argument of the relation, i.e., the \textit{dest} attribute of the first \textit{outArc} (represented through the array index notation ‘[0]’). Similarly, the forward expectation is associated with the \textit{source} attribute of the \textit{inArc}. The definition could go further to include a constraint that the arguments are Concepts and not Contexts; however, for the simple Conceptual Graph meta-model we have defined, it does not make a difference.

We can now use this lexicon to analyse the sentence \textit{Some branch is included in a local area}. This is similar to the example discussed in Section 4.3 (i.e. \textit{Each branch is included in exactly one local area}), so we will not go into details of its syntactic analysis. The final parse tree and the suggestions that form it are displayed in Figure 4.19.

Several mapping rules are needed to instantiate the partial model from this parse tree and the suggestions that form it. While they are quite straightforward, they do not simply reference
Section 4.4. Semantic Analysis

the semantic structures evoked from the lexicon. Doing so for this meta-model would lead to incorrect interpretations: for example, associating a single concept to all of the relations in different sentences that refer to that concept would result in saying that there exists a single entity that takes part in all of the relations, which may not be the case. Instead the semantic structures of the lexical entries are more like model templates that should be instantiated each time one is required. The basic required mappings are shown in Figure 4.20. Given a suggestion $s$, leaf($s$) indicates a leaf node or simple suggestion (i.e. a suggestion with no catches), and semantic($s$) maps to the semantic structure of the suggestion. The right-hand side of the figure indicates the created model elements using the UML notation.

In each of these cases the rules apply to only the leaves of the tree and new model elements are created in the partial model, as indicated by the right hand side of the implication. Moreover, the Relation rule is generic for any number of arguments (with at least one) and indicates that all of the necessary Arcs are also created in the partial model. The result of applying the mapping rules of Figure 4.20 to the parse tree of Figure 4.19 is displayed in Figure 4.21.

Finally, the bold elements (shown in Figures 4.20 and 4.21) indicate the profiled part of the semantic structure: i.e., the most salient aspect of the semantic structure [Lan08]. This demonstrates a key difference between the semantic structure of some compared to that of branch or local area, which would otherwise seem identical: the existential quantification profiles the Concept as a whole, while the type name is profiled for the nouns. In this approach,
Chapter 4. Knowledge-based Framework for CNL Understanding

\[ \text{leaf}(s) \wedge \text{semantic}(s) = \star \Rightarrow \exists x : \text{Concept} \]
\[ x : \text{Concept} \]
\[ \text{type} = \star \]
\[ \text{name} = \text{null} \]

\[ \text{leaf}(s) \wedge \text{semantic}(s) = \langle \text{type} \rangle \Rightarrow \exists x : \text{Concept} \]
\[ x : \text{Concept} \]
\[ \text{type} = \langle \text{type} \rangle \]
\[ \text{name} = \text{null} \]

\[ \text{leaf}(s) \wedge \text{semantic}(s) = \langle \text{type} \rangle \Rightarrow \exists x : \text{Concept} \exists y, o_1 \ldots o_n : \text{Arc} \]
\[ x : \text{Relation} \]
\[ \text{type} = \langle \text{type} \rangle \]
\[ y : \text{Arc} \]
\[ \text{inArc} \]
\[ \text{dest} \]
\[ \text{source} \]
\[ o_1 : \text{Arc} \]
\[ \text{outArc}_1 \]
\[ o_2 : \text{Arc} \]
\[ \text{outArc}_n \]

Figure 4.20: Basic suggestion mapping rules for the simple Conceptual Graph semantics.

Figure 4.21: Partial Conceptual Graph model for Some branch is included in a local area after applying the mapping rules.

the profile indicates what is passed up the parse tree by a catching suggestion. Therefore, the result of \( \text{some branch} \rightarrow \text{branch} \), in which some catches branch, would be the profiled concept and not simply the type name. Moreover, the profiling information is used to identify which model elements should be connected and associated with constraints from the configuration requirements knowledge.

4.4.3 Configuration Requirements Knowledge

The configuration requirements knowledge is provided by the catch relationships of the suggestions. As the parse tree is processed by the mapping rules, additional constraints are encoded into the partial model as necessary. These constraints primarily consist of catch or word-order constraints, without which the configuration of the partial model may produce
incorrect, but otherwise valid, interpretations of the sentence. For example, the partial model of Figure 4.21 would allow `Branch` or `LocalArea` to be attached to either of the arcs, creating two possible configurations/interpretations\(^\text{[17]}\).

![Diagram](image)

Although both interpretations are valid with respect to the constraints of the meta-model, only the latter configuration is intended. Therefore, we introduce the additional constraints to prevent the latter from being found to be a valid interpretation. A simple and generic way to add these constraints is by applying a mapping rule to non-leaf nodes of the parse tree, for example:

\[
\neg \text{leaf}(s) \land es = \text{eSite}(s) \land p = \text{profile}(s.caught) \implies \text{catchConstraint}(es, p)
\]

where, for simplicity, `eSite(s)` provides the elaboration site of the suggestion’s associated semantic structure, `profile(s)` provides the profiled element of the suggestion’s semantic structure, and `catchConstraint(x, y)` introduces a relationship between two elements of the semantic structure that represents the constraint between the two due to the catch relationship of the suggestions.

Such a rule would lead to the partial model shown in Figure 4.22, in which the dotted lines represent the constraints. The exact form of the constraint for a given application must take into consideration the meta-model and the means with which the configuration will be performed (i.e. how it is encoded as a configuration problem). For example, the constraint between the type of the `Concept` created from the term `some` and the type of `Branch` may not be directly representable in a UML-based partial model since it is a relation between attributes\(^\text{[18]}\). Instead, a relationship may be added to the meta-model, say `typesEqualConstraint`, on `Concept` with an OCL invariant specifying that the type of the `Concept` at each end of the relationship must be the same. Similarly, a relationship for representing constraints between the end of an `Arc` and a `Concept` would be defined between the class `Arc` itself. An example of this is shown in Figure 4.23; it could even include multiple constraint relations on `Arcs`, one for

\(^{17}\)excluding the models formed by connecting to the untyped concepts

\(^{18}\)To do so would require defining the constraint at the level of the UML meta-model, for example, which is a meta-level above that at which we defined the Conceptual Graph meta-model. We will not go into such meta-meta-modelling here.
Figure 4.22: Partial Conceptual Graph model for Some branch is included in a loca area including catch constraints (dotted lines).

Figure 4.23: Conceptual Graph meta-model including constraint relations.

mustConnSource and another for mustConnDest. Furthermore, different constraints may need to be applied by different (mapping) rules\(^9\). For simplicity we will assume for the moment that this simple, generic rule is enough to define the constraints between any element.

4.4.4 Configuration Result

Now that we have a completed partial configuration containing both the solution knowledge (the Conceptual Graph model elements) and the requirements knowledge (the additional constraints imposed by the syntactic analysis) we can perform the configuration process. This process searches for one or more valid models given the partial model. If a solution can be

\(^9\)In fact, with such simple constraints it is even possible that the mapping rules themselves could perform the required operation (e.g., assigning the value of the type attribute) and leave more complex constraints to be resolved by the configuration. For the sake of example, though, all constraints are resolved through configuration.
found, then the sentence to which it corresponds can be considered to have an interpretation in the language defined by the meta-model.

The result of configuring the model of Figure 4.22 is displayed in Figure 4.24. The constraints have ensured that LocalBranch is the first argument of \texttt{Includes}, while Branch is the second. Moreover, the constraints between the concepts evoked by the existential quantification terms (\texttt{a} and \texttt{some}) and those created by the type names (\texttt{branch} and \texttt{local area} has resulted in coreference links between their respective concepts. This leads to unnecessary duplicates in the Conceptual Graph. Therefore, the final model can be obtained by eliminating the identical concepts with the \texttt{simplify} operation of Conceptual Graphs (displayed on the right-hand side of the figure).

### 4.4.5 More Complex Language

The language defined so far can only make very simple statements about the existence of entities and the relationships between them. This is the case even though the semantics we have defined supports more complex statements through the use of negated contexts. We can extend the language by specifying additional lexical entries for \textit{closed class words} to support negation, implication, and universal quantification, as in the following:

\[
\text{not} \quad \mapsto \quad \left( \langle \text{‘not’}, * \rangle, \neg \right) \\
\quad \quad \quad \left\{ (1, \text{forward, required}, \text{Negation.n}	ext{odes}) \right\}
\]

\[
\text{it is not the case that} \quad \mapsto \quad \left( \langle \text{‘it’}, \text{‘is’}, \text{‘not’}, \text{‘the’}, \text{‘case’}, \text{‘that’}, * \rangle, \neg \right) \\
\quad \quad \quad \left\{ (1, \text{forward, required}, \text{Negation.n}	ext{odes}) \right\}
\]

\[
\text{no} \quad \mapsto \quad \left( \langle \text{‘no’}, _1, _2 \rangle, \neg \right) \\
\quad \quad \quad \left\{ (1, \text{forward, required}, \text{Negation.n}	ext{odes}) \right\}
\]
Chapter 4. Knowledge-based Framework for CNL Understanding

if . . . then $\mapsto \left(\langle 'if', *_{1}, 'then', *_{2}\rangle, \neg *_{1} \neg *_{2}\right)$

$\left(\{1, internal, required, Negation\text{.nodes}\},\right.$

$\left.\{2, forward, required, Negation\text{.nodes}[0].nodes\}\right)$

each, every, any $\mapsto \left(\langle 'each', *_{1}\rangle, \langle 'every', *_{1}\rangle, \langle 'any', *_{1}\rangle\right),$$

\left\{\neg *_{1} \neg *_{2}\right\}$

$\left.\{1, forward, required, Negation\text{.nodes}\}\right)$

All of these lexical entries, or rather the semantic structures they refer to, utilise negated contexts (i.e. Negation nodes) and additional constraints. The three different negatives (not, it is not the case that, and no) support different scopes of negation. The first negates the relation that it catches, the second negates an entire caught clause/proposition, and the third negates an entire clause that the caught concept is involved. This allows for a number of additional sentences to be interpreted, such as:

Some branch is not included in some local area.

$\implies \exists x : Branch \exists y : LocalArea \left[\neg \text{Includes}(y, x)\right]$

It is not the case that some branch is included in some local area.

$\implies \neg \left[\exists x : Branch \exists y : LocalArea \left[\text{Includes}(y, x)\right]\right]$

No branch is included in some local area.

$\implies \neg \exists x : Branch \exists y : LocalArea \left[\text{Includes}(y, x)\right]$

The first-order logic statements below each sentence indicate the scoping of the negation. While the last two are effectively the same logically, the difference appears in language use. For example, the word no is used as a determiner, similar to a, while the phrase it is not the case that typically precedes an entire clause and can be used to negate sentences as a whole.

The lexical entry for if . . . then is quite straightforward. It has two grammatical expectations and two semantic elaboration sites associated with nested negative contexts. Remembering that conceptual graphs are interpreted in first-order logic as conjunctions, the nested concepts are equivalent to $\neg(p \wedge \neg q),$ where $p$ is the content of the outer context and $q$ is the content of the inner context, which is equivalent to $(p \rightarrow q)$ following De Morgan’s rules [RN03]. Similar to it is not the case that, if . . . then can catch entire clauses, for example:

If a car is damaged then it is not rentable.

$\implies \forall x : Car \left[\text{Damaged}(x) \rightarrow \neg \text{Rentable}(x)\right]$
Section 4.4. Semantic Analysis

The last lexical entry defines a number of synonyms for universal quantification, namely *each*, *every*, and *any*, used in different contexts in natural language. Similarly to the implication of *if . . . then*, the universal quantifier is created through nested negations; however, they are constrained to catching concepts and there is no second grammatical expectation with which to catch the content of the nested negation. This second semantic elaboration site is realised through constraints to ensure that the other elements of the graph end up at the necessary scope when the model is configured.

Furthermore, since the desired scoping is affected by word-order, we will extend the generic catch constraints created between suggestions with *word-order* constraints—since the catches may be *backward*, the catch constraints alone do not give us word-order and there is nothing in the Conceptual Graph meta-model that identifies word order. These constraints form a transitive relation *precedes* and are applied all the way up the parse tree. The additional constraints are introduced as follows:

\[
\neg \text{leaf}(s) \land p_1 = \text{profile}(s) \land \text{catch\_type}(s) \neq \text{backward} \land p_2 = \text{profile}(s.\text{caught}) \\
\implies \text{precedes}(p_1, p_2)
\]

\[
\neg \text{leaf}(s) \land p_1 = \text{profile}(s) \land \text{catch\_type}(s) = \text{backward} \land p_2 = \text{profile}(s.\text{caught}) \\
\implies \text{precedes}(p_2, p_1)
\]

Note that this does not provide a complete chain of *precedes* relations matching the token stream, rather the precedes relations follow the structure of the parse tree. This may not be suitable for all models.

The more specific constraints to support the scoping are needed on a per lexical entry basis for the closed class words represented here. Figure 4.25 incorporates the additional constraint relations into the meta-model, along with some operations required for specifying the constraints. Figure 4.26 displays the mapping rules for the negation based terms, illustrating the elements and constraints that are generated in the partial model.

\[^{20}\text{Since the term no is effectively a universal quantification with an embedded negation, it displays the same characteristic.}\]
With the constraints to ensure the correct scoping, we can now analyse sentences with mixed quantifiers and arrive at the intended interpretation.\(^{21}\) Consider the sentence and its intended interpretation:

Some branch is included in each local area.

\[ \exists x : \text{Branch} \forall y : \text{LocalArea} [\text{Includes}(y,x)] \]

This is a potentially problematic sentence due to the requirement that the existential quantification of \textit{branch} occur outside the negative context representing the universal quantification of \textit{local area}. The partial model of this sentence is displayed in the top half of Figure 4.27; it shows the elements and the constraints between them. As you can see, nothing is directly

\(^{21}\)In most cases at least. In general natural language processing there is still the ambiguity of distributive vs. collective interpretations of the universal quantification in a sentence \cite{All95}. For example, \textit{Some branch is included in each/every local area, or Every branch is included in some local area} could be interpreted as \[ \exists x : \text{Branch} \forall y : \text{LocalArea} [\text{Includes}(y,x)] \text{ or } \forall y : \text{LocalArea} \exists x : \text{Branch} [\text{Includes}(y,x)]; \] in the latter there is a potentially different branch for each local area, whereas it is the same branch for all local areas in the former. This is particularly a problem in the presence of the indefinite article \textit{a}, as it is also polysemous, i.e., it can be either universal or existential quantification. Due to ambiguities such as this, language appears to have developed ways of clarifying the intention through different terms: e.g. \textit{a particular} instead of \textit{some, all or any} instead of \textit{each}. Therefore, within the context of a controlled language it is reasonable to fix the allowable interpretations to a limited set that will be correct most of the time. Validation by users can then be performed by checking paraphrases, to ensure that it was interpreted as expected by the user. This is a commonly used technique. Alternatively, if we were to relax the constraints, we could find all of the different interpretations during model search and report them to the user and have them select one.
Figure 4.26: Negation term mapping rules for the simple Concept Graph semantics.
connected to the nested context that was created as part of the universal quantification. However, the constraints we have defined push the required nodes into the nested Negation. For example, the catch constraint causes \textit{Includes} to be directly connected to the concept in the outer negation and the constraint for ‘each’ says that nodes that are directly connected to that concept must be included in the inner negation. Therefore, for a configuration to be valid, \textit{Includes} must appear within the nested negation; anywhere else and there would be a constraint violation. This is shown in the bottom half of Figure 4.27 which illustrates the solution of the model search.

In the final model, \textbf{Branch} is kept outside of the scope of the universal quantifier—even though it is connected through the relation \textit{Includes}—as \textit{Branch} \textit{precedes} \textit{Includes} and, hence, \textit{precedes} the universal quantifier. The more complex example of Figure 4.28 illustrates the handling of additional levels of scope/nested Negations. With the defined constraints, the semantics of the language will be consistent across various sentences that could otherwise be interpreted in different ways. For example:

\begin{align*}
\text{Each/every branch is included in some (a particular?) local area.} \\
\implies \forall x : \text{Branch} \exists y : \text{LocalArea} \ [\text{Includes}(y, x)] \\
\not\implies \exists x : \text{LocalArea} \forall y : \text{Branch} \ [\text{Includes}(x, y)]
\end{align*}
Some local area includes every branch.

$$\Rightarrow \exists x : \text{LocalArea} \ \forall y : \text{Branch} \ [\text{Includes}(x, y)]$$

Some (a particular) branch is included in every (all) local area(s).

$$\Rightarrow \exists x : \text{Branch} \ \forall y : \text{LocalArea} \ [\text{Includes}(y, x)]$$

$$\nLeftarrow \forall x : \text{LocalArea} \ \exists y : \text{Branch} \ [\text{Includes}(x, y)]$$

Each local area includes some branch.

$$\Rightarrow \forall x : \text{LocalArea} \ \exists y : \text{Branch} \ [\text{Includes}(x, y)]$$

To reverse the order of the quantifications you must reverse the order of the sentence. This restriction is not necessarily a bad thing in the context of controlled languages; however, modifications to the constraints could allow all of the different interpretations to be found during model search. Other complex language features can be handled by the configuration process in a similar manner, i.e., based on the constraints of the meta-model and the additional requirements introduced by the syntactic analysis. The unification of different semantic analysis tasks into a single process is a key benefit of using configuration/model search for the semantic analysis of text.

4.4.6 Interactive Feedback, Alternatives, and Corrections

Another benefit of using configuration for semantic analysis is the reporting of errors (i.e. constraint violations) and their repair, as well as presenting alternative interpretations. The former occurs when no valid configurations can be found, in which case an explanation of why the sentence was unable to be parsed can be generated. Moreover, potential solutions can be suggested by relaxing the constraints around the error. If, on the other hand, multiple solutions are found due to ambiguities, the alternative interpretations can be presented to the user.

Such flexibility in the process allows an application to actively support users in refining their documents into more precise and less ambiguous forms. However, this is heavily dependent on the meta-model representing the semantics of the controlled language being defined. The

---

22 Generating the alternatives may not be desirable and additional information could be added to the model to handle only the potentially ambiguous combinations of terms, e.g. every and some, in such a manner

23 Others will be discussed in the context of the application in Chapter 5
simple Conceptual Graph meta-model presented in this chapter is too general and lacks a number of constraints that would be necessary to support error correction and refinement: for example, support for sub-types and relation definitions (or signatures) would be necessary to improve the feedback, repair, and alternative suggestions of erroneous statements due to the wrong types being involved (i.e. selectional restrictions [All95]). Therefore, we present further details of feedback in relation to the semantics and application described in Chapter 5.

4.5 Summary

In order to cope with the increasing level of abstraction in the specification and development of software, it is important to provide tools that can be used by both business and domain experts, not only technical experts. In particular, tools should support these non-technical users in formalising and improving their specifications while reducing the time to do so. This means the tools need to be able to express what the users need to represent and do so using a
natural notation or language that improves the user’s understanding of the specification. To achieve this, we believe a flexible NLU approach is required.

To that end, this chapter presented our general knowledge-based framework for constructing flexible CNLs and NLU. Like other NLU approaches it consists of three key aspects: (1) lexicon, (2) syntactic analysis, and (3) semantic analysis. The novelty of our approach lies in the synthesis of Cognitive Grammar and Knowledge-based configuration, which combines a simple, flexible syntactic analysis with a powerful semantic analysis that unifies common processes required for NLU applications—e.g. word-sense disambiguation, co-reference resolution, and scope resolution. Moreover, the flexible syntactic analysis of Cognitive Grammar supports a lexical structure that is more amenable to non-linguists as it does not require complex linguistic information to define its entries. Finally, we discussed the potential of the configuration-based semantic analysis to provide feedback on errors, suggested corrections, and alternative solutions. Such features would support users in reducing ambiguity and formalising their specifications. Although the simple meta-model used for the example limited us from demonstrating a concrete example of the feedback and error handling aspect, we will return to the topic in the following chapter which presents the application of this framework to SBVR.
Chapter 5

Application to Specifications using SBVR

In this chapter we describe CLUE, a prototype implementation of the general knowledge-based framework for understanding described in the previous chapter. In particular, the target application is intended to achieve the goals set out previously: that is, a means for non-technical business or domain experts to formalise their (business or software) requirements specifications using (controlled) natural language. To that end we have identified the Semantics of Business Vocabulary and Business Rules (SBVR) as a suitable target semantics for the application; the resulting tool is dubbed CLUE4SBVR.

In the following we introduce SBVR and why it was selected to form the basis of the application (Section 5.1) followed by a brief overview of the prototype, its architecture and processes (Section 5.2). We then provide a more detailed description of the different components of the implementation that realise different aspects of the framework, namely: the lexicon, Section 5.3; syntactic analysis, Section 5.4; and semantic analysis, Section 5.5 (following structure of the previous chapter describing the general framework). Finally, we describe the processes for lexicon acquisition that assist the users in formalising their specifications by attempting to semi-automate the creation of domain vocabularies/lexicons (Section 5.6).
Chapter 5. Application to Specifications using SBVR

5.1 Semantics as SBVR

The Semantics of Business Vocabulary and Business Rules (SBVR) [OMG08b] is a standard developed by the Object Management Group (OMG) for the documentation of business specifications in natural languages, i.e. sets of business vocabularies, facts, and rules and that describe the operations of a business. SBVR is intended for use by business people to improve communication and assist in the exchange of business knowledge between different organisations and software tools through models. As a result it aims to improve communication between stakeholders and bridge the gap between business people and the IT people (technical experts) who develop information systems for them. The following features of SBVR models make them ideal intermediate models:

1. they are at the CIM LoA of the MDA (fulfilling criterion H1);
2. they support information on both business structure and its operations (specified declaratively rather than the imperative style of an explicit process), which supports criterion H3.1;
3. they can be interpreted in formal logic, i.e. first-order logic, with extensions in higher-order logic and modal logic—both alethic modalities (necessity/possibility/impossibility) and deontic modalities (permission/obligation)—making them expressive enough to fulfil criteria H3.2 and H3.3 (i.e. PEN(N)S expressiveness $E^3_m$);
4. they can be processed by model transformations to create or complement other models (e.g. UML Class Diagrams, BPMN models); and
5. they partly retain the textual structure of the specification, improving traceability to the original text.

Although SBVR is designed for business users, it does not mean that it cannot be used by others, e.g. technical experts, at other levels of software development. Indeed, the features that make it suitable to formalise business specifications are also beneficial in formalising software specifications at lower levels. Furthermore, as improved communication and shared understanding is an important aspect of SBVR, it could be argued that technical experts should
be using SBVR especially when working in large distributed development teams and when requiring feedback and validation from domain experts and other non-technical stakeholders.

The SBVR specification includes two important aspects: (1) a conceptual (meta-)model, and (2) a controlled English representation called SBVR SE. We will briefly introduce the two components in what follows.

5.1.1 SBVR Metamodel

The conceptual model (and associated MOF-based meta-model) of SBVR standardises a set of concepts for the definition of business vocabularies and rules. It constitutes the semantics of a controlled language, allowing the defined business specifications to be interpreted with reduced ambiguity. In particular, SBVR can be interpreted in formal logic: primarily first-order logic with an extension in modal logic (necessity, obligation, permission, possibility), and higher-order logic using Henkin semantics [OMG08b]. This basis in formal logic is highly expressive and supports various forms of reasoning, and consistency and conformance checking.

The meta-model is separated into two core parts. The first, ‘Meanings and Representations’, forms the basis of defining the vocabulary as a set of interrelated object types (e.g. ‘branch’, ‘local area’), individual concepts (e.g. ‘EU-Rent’, ‘Australia’), and fact types (e.g. ‘includes’/‘is included in’). The meanings are separated from representations to allow a single concept to be represented by different words (in the same or different languages), images, or sounds. The core elements for defining meanings and representations are shown in Figure 5.1 using UML notation.

The ‘Logical Formulations’ component of the meta-model forms the semantic structure of business rules as a conceptualisation of formal logic. The meta-model includes concepts for first-order logical operators (e.g. conjunction, disjunction, implication), quantification (e.g. universal quantification, existential quantification), and modal operators (e.g. necessity, possibility, obligation, permission). An extract of the ‘Logical Formulations’ aspect of the metamodel is shown in Figure 5.2 (the dotted lines indicate that elements have been omitted).

---

1 We use sans-serif font to denote concepts from the SBVR metamodel.
5.1.2 SBVR Structured English

SBVR Structured English (SBVR SE) is a non-normative notation for expressing business vocabularies and rules with SBVR semantics. It constitutes the syntax of a controlled language that ‘... maps mechanically to SBVR concepts’ [OMG08b, p. 237]; however, it does not eliminate ambiguity altogether. SBVR SE is mainly described by restrictions to standard English and, from the linguistic point of view, has a number of features, including:

1. verbose expressions with deeply nested clauses and long dependencies between syntactic elements;

2. a high degree of polysemy (e.g. ‘local area includes branch’ and ‘company includes local area’ are considered semantically different relations);
3. negation always has local scope;

4. modalities always have wide scope;

5. quantification scope is restricted, but not deterministic, and must be resolved according to the rules of standard English;

6. anaphoric reference (i.e. forward or backward references to the mention of an object) is limited to within each rule and to definite reference, i.e. typed references such as ‘the branch’ are allowed but not pronominal references (e.g. ‘it’);

7. ellipsis of repeated subjects and verbs (e.g. in ‘a branch has a name and [the branch has] a country’ the expression in square brackets may be omitted); and

8. implicit transformations between some types of terms: e.g. certain unary fact types can be used as adjectives, and some reduced relative clauses are supported (e.g. ‘of’ is interpreted as ‘that is of’).

While many of these features help to reduce ambiguity, some features (or combinations thereof) lead to increased ambiguity and other issues. For example, restrictions to the scope of negations help reduce ambiguity, while the distant dependencies created by deeply nested clauses are problematic in natural language processing. Furthermore, combining a high degree of polysemy with omitted verbs leads to a complicated situation where the verb is considered the same from the point of view of the user, but semantically different from that of the computer.

An [SBVR SE] specification is separated into sections for vocabulary and rules. The vocabulary section takes the form of a detailed glossary with elements such as notes, examples, synonyms, general concept, and concept type; most of which are optional. A vocabulary entry can also include structural rules directly related to the given term. The rules section consists of individual rule statements, which may include additional guidance information, notes, and other fields. These additional elements can help to correctly process the specification; however, they are not strictly necessary as many are intended as additional information for the people reading the specification, not the computer.

When using ‘not . . . ’. There is also a form with wide scope, ‘it is not the case that . . . ’.
SBVR SE uses text styling (bold, italics, colour, etc.) to differentiate elements of a specification that map to different elements of an SBVR model. When performed automatically by a tool, this can aid a user in determining whether or not the vocabulary and rules have been correctly interpreted. The following styles are used in this chapter to demonstrate the intended SBVR interpretation of statements. Although similar to the styles defined in [OMG08b], we more clearly delineate keywords with dotted underlining (to better differentiate adjacent keywords and to help identify keywords from normal text in black and white printing).

- underlined terms represent the designations of object types
- **Names** represent the designations of individual concepts
- italicised verbs represent the designations of fact types
- keywords represent words forming statements when combined with other words and that typically map to logical formulations
- informal text is shown in normal font, which allows for additional notes and examples that are not intended to be interpreted formally

We use SBVR SE as a starting point for processing natural language into SBVR models for several reasons:

- it is intended for use by business users and has a reading and writing naturalness of $N^4 N^4_{uw}$, fulfilling criterion H2
- it has a relatively straightforward mapping to SBVR semantics, described in [OMG08b, Annex C];
- it is not overly restrictive, allowing rules to be expressed quite freely in a number of ways; and
- it is the notation used by the SBVR specification itself, which provides access to a reasonably sized case study in [OMG08b, Annex E].
Section 5.2. Process Overview and Architecture

5.2 Process Overview and Architecture

As the tool is intended to support users in formalising natural language specifications, let us consider our approach in the context of a possible organisational workflow that makes use of it. The two processes, the general workflow and the analysis process performed by CLUE4SBVR, are displayed in Figure 5.7.

The general workflow (Figure 5.7a) represents the process of an organisation developing a formalised business specification (documents detailing the terms and requirements of the business in the form of business rules [HH00]). While we assume that an organisation already has some existing documentation or specification, it is not a requirement. The process is equally applicable to an organisation that wants to formalise their business specification from scratch. Moreover, while we focus on business users here, our approach does not preclude a collaborative or agile approach involving multiple stakeholders. For example, it is likely that the process will still require a business analyst or requirements engineer who can ‘ask the right questions’ to elicit requirements that the business users take for granted (or are otherwise left implicit). Moreover, since our approach works by incrementally building and checking the formal model, it fits naturally into an iterative process (such as the Agile Business Rules Approach [BM11]) where the vocabulary and rules are incrementally defined, refined and extended in scope.

For the sake of example, we consider the EU-Rent case study, developed by the Business Rules Group, of a fictitious car rental company with global presence [HH00, Annex D]. Figure 5.3 provides an informal summary (or problem statement) of EU-Rent’s business structure and locations. In contrast, it is expected that business rules are documented in a more structured fashion: for example, by dividing the document into sections and separating individual rules. Such semi-formalisation can regularly be seen in legal, policy, guideline, and technical documents in which sections, lists, etc. are used to structure the information. This document structure can be used to help process the specification. Figure 5.4 illustrates a semi-formal specification for the EU-Rent case study.

---

3While these could be considered formal documents, they are still only text and, therefore, are semi-formal in contrast to a formal model or notation.

4Often there are best practices that recommend using such constructs, for example those for requirements engineering [SS97; Lau02] and business rules [BM11; HH00].
EU-Rent is a car rental company owned by EU-Corporation. It is one of three organisation units—the other two being hotels and an airline—that each has its own business and IT systems, but with a shared customer base. Many of the car rental customers also fly with EU-Fly and stay at EU-Stay hotels.

EU-Rent has 1000 branches in towns in several countries. At each branch cars are available for rental to customers. Each branch is part of a local area responsible for managing their respective branches. The local area managers form part of a company operating out of a specific country.

The company operating out of Australia is EU-Rent AU.

**Figure 5.3: EU-Rent case study overview adapted from [HH00]**

**EU-Rent Business Rules**

**Organisation Structure**

- A branch is part of a local area.
- A local area is included in an operating company.
- An operating company operates out of a specific country.
- The country of a branch is the same as the country of the operating company that includes the local area of the branch.

**Facts**

- The company operating out of Australia is EU-Rent AU.

**Figure 5.4: Semi-formalised EU-Rent business rules**

**Glossary of Terms**

- **branch**: an organisational unit responsible for renting cars to customers
- **local area**: an organisational unit responsible for managing a group of branches
- **operating company**: an EU-Rent member company that performs EU-Rent business in a specific country

**Figure 5.5: Semi-formalised EU-Rent business vocabulary**

Apart from clear structure, a glossary of terms can also be helpful in processing specifications with a knowledge-based approach. Although [HH00] does not include a glossary for the EU-Rent case study, glossaries are a common element of business and technical documentation, and an important aspect of formalising a specification (as evidenced by its incorporation into the SBVR standard). A glossary to accompany the EU-Rent extract may appear similar to that displayed in Figure 5.5.

The goal, then, is to translate these sets of natural language rules and their accompanying
### EU-Rent Locations

#### Vocabulary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
<th>Concept Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>rental organisation unit</td>
<td>organisation unit that operates part of EU-Rent’s car rental business</td>
<td>role</td>
</tr>
<tr>
<td>rental organisation unit having rental responsibility</td>
<td>the rental organisation unit is responsible for the operation of customer-facing rental business</td>
<td></td>
</tr>
<tr>
<td>rental organisation unit having area responsibility</td>
<td>the rental organisation unit includes organisation units for which it has the responsibility to coordinate operations and ensure resources</td>
<td></td>
</tr>
<tr>
<td>local area</td>
<td>rental organisation unit that has area responsibility</td>
<td></td>
</tr>
<tr>
<td>branch</td>
<td>rental organisation unit that has rental responsibility</td>
<td></td>
</tr>
<tr>
<td>branch is included in local area</td>
<td>Synonymous Form: local area includes branch</td>
<td></td>
</tr>
<tr>
<td>branch has country</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EU-Rent operating company</td>
<td>Synonym: operating company</td>
<td></td>
</tr>
<tr>
<td>EU-Rent operating company operates in country</td>
<td>Synonym: operating company has country</td>
<td></td>
</tr>
<tr>
<td>EU-Rent operating company includes local area</td>
<td></td>
<td></td>
</tr>
<tr>
<td>country</td>
<td>Source: MWU(1, 2b) [&quot;country&quot;]</td>
<td></td>
</tr>
<tr>
<td>Australia</td>
<td>General Concept: country</td>
<td></td>
</tr>
<tr>
<td>EU-Rent AU</td>
<td>Definition: the EU-Rent operating company that is located in Australia</td>
<td></td>
</tr>
</tbody>
</table>

#### Rules

- Each branch is included in exactly one local area.
- Each local area is included in exactly one operating company.
- Each operating company operates in exactly one operating country.
- The country of a branch is the country that is of the operating company that includes the local area that includes the branch.

**Figure 5.6:** EU-Rent case study, Locations extract [OMG08b, Annex E]

Glossary (i.e. the business specification) into formal models. For example, Figure 5.6 (extracted from [OMG08b, Annex E]) represents part of the EU-Rent case study using SBVR SE and, hence, is mapped to the SBVR's formal model. The process of getting from one to the other is what is illustrated in Figure 5.7a.
Chapter 5. Application to Specifications using SBVR

(a) General workflow

The process begins whenever the organisation has some rules they wish to specify; this could be due to changes to policies or the business environment, or due to wanting to formalise their rules for the first time. A business person then goes about defining (or updating) the business vocabulary and rules of the organisation (1). Although, we make use of SBVR sematics, we do not require the use of SBVR SE document structure. Rather, we make allowance for the organisation to use their own preferred style and format in order to support criterion H4. This helps to minimise the manual work required in formalising a business specification. If the document format varies too widely, however, a technical expert may be required to assist in customising the rules that identify the different elements of the document (2).

(b) Details of subprocess (3) ‘Process Vocabulary and Rules’

Figure 5.7: A possible workflow using our approach and details of the parsing process

While we show the technical expert as part of the organisation, this is not necessarily the case.
Once document structure and formatting issues are dealt with, the vocabulary and rules are processed by our system in 3, which is expanded in Figure 5.7b. First, the different parts of the business specification are identified using tokenisation, sentence splitting, and rules for identifying structural elements of the document (e.g. vocabulary and rule entries). The tokenisation and sentence splitting are performed using standard components of GATE (General Architecture for Text Engineering) [CMBT02].

Next, a component of CLUE4SBVR (realised as a GATE plugin) creates a preliminary SBVR model by processing the vocabulary entries of the specification. This process is described in Section 5.6.1. The SBVR model that results forms the basis of the lexicon, allowing the subsequent syntactic and semantic analysis of the rules, and is incrementally revised as rules are parsed. CLUE4SBVR then performs syntactic analysis of the document using an implementation of the expectation-based parsing approach introduced in Section 4.3 (the SBVR-based implementation is described in Section 5.4).

Following the syntactic analysis stage, the semantic analysis sub-process takes the parse tree and the partial SBVR model and calls the configurator to perform model search, described in Section 5.5. The configurator, COCOA (COnstraint-based COnfiguration Architecture), is a Smalltalk implementation of a “generative constraint satisfaction” solver (based on that described in [SFH98]), which is based on a non-deterministic, backtracking, depth-first search constraint solving algorithm. To perform the configuration, COCOA relies on a knowledge-base, which is a version of the SBVR meta-model transformed into the format required by COCOA. If the configuration process is successful, the result is an interpretation of the rule(s) as a valid SBVR model. However, if the configuration fails, the errors and inconsistencies are analysed for any that can be resolved automatically. If possible, the errors are corrected and the specification is reprocessed to ensure no inconsistencies were introduced by the modifications.

The result of parsing a specification is a progressively more detailed SBVR model containing the concepts, their definitions, and associated rules. After processing the specification, the results (including any errors that could not be automatically resolved and any corrections that were made) are presented to the user for review—stage 4 of Figure 5.7a. The user can then make changes if necessary and reprocess the document, returning to 1, which updates the
generated SBVR model and provides additional feedback. This cycle will continue until no more changes need to be made.

Once the business user has resolved any issues with the specification, the SBVR model will be provided to the technical experts to support the development of an information system. Moreover, it can undergo automated model transformations or otherwise be used to develop PIMs in accordance with the Model-Driven Architecture [MM03]. For example, model transformations could be used to create UML models (e.g. Class and Activity Diagrams), rules for execution on a rules engine, etc.

The components used to realise the analysis process are displayed in Figure 5.8. GATE provides a pluggable development environment for the annotation of texts, for which our prototype is implemented as a plugin. We reuse some of the core GATE components, including: JAPE, which provides a regular expression-like language for matching text and annotations in a document; the tokeniser, which annotates individual tokens of a document based on a set of JAPE rules; and the sentence splitter, which identifies the beginning and ending of individual sentences. We also make use of the Eclipse Modelling Framework (EMF) [Ecla], which provides an implementation of the MOF meta-meta-model, and Epsilon [Eclb] for model transformations.

The CLUE4SBVR plugin itself includes components for the different stages of processing. In addition it incorporates a couple of supporting components: the ‘Lexicon’ component, which manages the learnt vocabulary, and the ‘SBVR Meta-model’, on which the semantic structures of this implementation are based.

### 5.3 Lexicon

When using SBVR, the lexicon is partly derivable from the ‘Meanings and Representations’ aspect of the models. This because object types, individual concepts and fact types form the semantic structures of open class terms and the model itself incorporates the textual representations of these concepts. Deriving the lexical entries for object types and individual concepts is straightforward as they do not require any expectations. Basically, a new lexical
Section 5.3. Lexicon

**Figure 5.8:** Component diagram of the CLUE4SBVR prototype implementation of the framework.

**Figure 5.9:** Derivation of lexical entries from object types and individual concepts.

An entry is created for each synonymous representation (expression) of an object type or individual concept, all linked to the same concept, and mappings are introduced for each expression. The generic derivation is displayed in **Figure 5.9**, in which the doubled-up objects indicate collections.

For fact types, lexical entries are created in a similar manner with the addition of grammatical expectations being derived from the placeholders associated to the representations. The
grammatical expectations created in this way are linked to the fact type roles of the fact type. The general structure of the fact type lexical entries is created as shown in Figure 5.10. The functions \(idx\) and \(type\) (used in the figure) determine the index and type (backward, forward, or internal), respectively, of the grammatical expectation based on the position information (the \(is\ at\) attribute) of the placeholder.

An example SBVR model containing the ‘Meanings and Representations’ elements and its derived lexicon is displayed in Figure 5.11.

In contrast, the lexical entries for the closed class terms map to the ‘Logical Formulations’ aspect of the SBVR meta-model and are manually created to form the initial lexicon. These lexical entries are associated with elements of the meta-model itself, rather than instances of an SBVR model, that must be instantiated anew each time the term is used. Each of the main sub-type of logical formulation have similar lexical entries. Some examples are illustrated in Figure 5.12. These lexical entries define the keywords of the controlled language, giving them a specific meaning with respect to the SBVR semantics.

The alternative lexical entries for modalities allow rules two be specified in different ways. For example, the following two sentences represent the same semantics:

- It is necessary that each branch is included in exactly one local area.
- Each branch is always included in exactly one local area.

Finally, the lexicon is compiled into a JAPE grammar, allowing the simple annotation of a document with known terms. An example JAPE grammar\(^6\) is shown in Listing 5.1, which

\[^6\]A JAPE grammar is similar to a regular expression except that it matches annotations (delimited by curly braces), rather than text.
Section 5.3. Lexicon

Figure 5.11: SBVR model of the example vocabulary and the lexicon derived from it. Grey dotted arrows indicate relations between different vocabulary entries.

Illustrates simple and multi-word object types, a fact type, and a keyword with an internal expectation. Each rule of the grammar creates a “word” annotation if it finds a matching (sequence of) token(s) based on the text or the root form of the word (when taking morphological analysis into account). The rules for object types, e.g. the ‘Branch’ rule on Line 1, are quite straightforward, while those for fact types and keywords are more complex as they must take into consideration such things as gaps for internal expectations (see Line 38). An example of the generated annotations is displayed in Figure 5.13; these annotations are later used in the syntactic analysis process.
∀ex ∈ \{\langle each \rangle, \langle every \rangle, \langle a \rangle \}.

∀ex ∈ \{\langle some \rangle, \langle a \rangle, \langle at, least, 1 \rangle \}.

(a) Quantifications

⟨always⟩ ⟷

\begin{align*}
\langle always \rangle & \rightarrow \\
\text{⟨prototype⟩} & : \text{universal quantification} \\
\text{v: variable} & \text{ introduces} \\
\{(1, forward, required, v)\}
\end{align*}

⟨it, is, necessary, that⟩ ⟷

\begin{align*}
\langle it, is, necessary, that \rangle & \rightarrow \\
\text{⟨prototype⟩} & : \text{existence quantification} \\
\text{min. cardinality = 1} & \text{ introduces} \\
\text{v: variable} & \text{ introduces} \\
\{(1, forward, required, v)\}
\end{align*}

(b) Modalities

⟨and⟩ ⟷

\begin{align*}
\langle and \rangle & \rightarrow \\
\text{⟨prototype⟩} & : \text{conjunction} \\
x & \text{ scopes over x} \\
y & \text{ scopes over y} \\
\{(1, backward, required, x), \ (2, forward, required, y)\}
\end{align*}

⟨if, then⟩ ⟷

\begin{align*}
\langle if, then \rangle & \rightarrow \\
\text{⟨prototype⟩} & : \text{implication} \\
x & \text{ antecedent} \\
y & \text{ consequent} \\
\{(1, internal, required, x), \ (2, forward, required, y)\}
\end{align*}

⟨not⟩ ⟷

\begin{align*}
\langle not \rangle & \rightarrow \\
\text{⟨prototype⟩} & : \text{negation} \\
x & \text{ operand1} \\
\{(1, forward, required, x)\}
\end{align*}

⟨if⟩ ⟷

\begin{align*}
\langle if \rangle & \rightarrow \\
\text{⟨prototype⟩} & : \text{implication} \\
x & \text{ antecedent} \\
y & \text{ consequent} \\
\{(1, backward, required, y), \ (2, forward, required, x)\}
\end{align*}

(c) Logical Operations

Figure 5.12: Example lexical entries for the main categories of logical formulation: Quantifications [a], Modalities [b], and Logical Operations [c]. The dashed line in the lexical entry for ⟨always⟩ indicates a constraint between the ends of the two relations.
In order to implement the expectation-based syntactic analysis in CLUE4SBVR, certain design decisions had to be made with respect to the aspects of the general framework that were left undetermined in Chapter 4. This includes choosing the form of tokenisation (i.e. morpheme or word-level, see discussion in Section 4.3.3), specifying a concrete pruning strategy (see Section 4.3.1), defining a maximum catching distance, specifying catch/caught weights, and implementing the appropriate form of the algorithm (see Section 4.3 for the general form of the algorithm).
Each branch is always included in exactly one local area.

**Figure 5.13:** Example annotations created from lexicon JAPE grammar.

### 5.4.1 Tokenisation Strategy

While we perform morphological analysis, the token stream for the SBVR-based application is at the word level. This can be seen in the JAPE grammars created from the lexicon discussed in Section 5.3. In these grammars, ‘Word’ annotations are created by matching text, or possibly the root word, of the initial ‘Token’ annotations, which are created by the GATE tokeniser based on spaces and punctuation. The ‘Word’ annotations then form the token stream that is processed by the syntactic analysis component.

The word-level approach was chosen for several reasons. Firstly, it is somewhat simpler than a morpheme level approach and, therefore, simplifies the initial prototype implementation, including the vocabulary acquisition from glossaries. The downside is that it is less general, with more entries needing to be specified in the lexicon to handle the different morphological variants of a word. However, in the context of an initial prototype for a controlled language, this was considered an acceptable trade-off.

In addition, the word-level approach corresponds well with SBVR as it focuses on complete words, including compound words (multi-word expressions). Moreover, the controlled nature of SBVR limits the number of morphological variants used for most terms.
Finally, in the initial implementation the morphological analysis was not used as it required POS tagging to be performed first for best results, which we were trying to not rely on. The addition of morphological analysis was added later to better support the acquisition of lexical entries from unrestricted text (see Section 5.6.2). Future versions of the prototype will look at removing the requirement of the POS tagger for morphological analysis and make better use of the information during the syntactic analysis. This will be important for generalising the implementation to other controlled languages (and eventually general natural language).

5.4.2 Pruning Strategy

In Section 4.3 and Section 4.3.1 we discussed the general aspects of a pruning strategy, including the different factors that may be taken into account: e.g., catching distance, binding energy, local and global coverage, catch/caught weights, and current token distance.

The pruning strategy implemented in the prototype is based on the suggestions made by Holmqvist [Hol93] with extensions to our catch and caught weight functions and to the concept of important suggestions. A suggestion is considered important if it is large (which we define as consisting of six or more elements) and has unfilled forward expectations. This is necessary to cope with rules consisting of a large number of nested restrictions (definite descriptions/relative clauses) since, without it, suggestions early on may be pruned before they can catch the complete nested structure. This is somewhat an SBVR specific issue due to the nature of rules specified using SBVR SE. For example, in

The country of a branch is the country that is of the operating company that includes the local area that includes the branch.

all of the relationships must be specified explicitly in a single sentence resulting in a deeply nested structure. The parse tree is shown in Figure 5.14. This is somewhat artificial and not a very nice way of saying that the ‘country of a branch is the same as that of the operating company it is a part of.’ However, such nested structures can, and do, occur in unrestricted natural language for which the strategy advised by Holmqvist [Hol93] is to reprocess the

7The pipeline in GATE is: tokenisation → POS tagging → morphological analysis. Each stage updates the initial word-based ‘Token’ annotations with additional attributes.
sentence allowing greater distances if it reaches the point where it appears a suggestion has been pruned that is necessary for the entire parse. This is due to his motivation for pruning being, not simply for efficiency, but to simulate the limited working memory of people. Moreover, the overarching consideration is that the parameters should allow efficient parsing in most cases; therefore, since deeply nested structures are likely to be common in SBVR SE-based texts, we ensure they can be processed the first time, rather than needing to constantly perform reprocessing.

The requirement for important suggestions is also somewhat an artefact of the lack of a completely specified grammar and the entirely bottom-up approach we have taken, in which unfilled expectations are hidden when a suggestion is caught. For example, when

```
7  the country that ◦  → 10
   1  the country of a branch is ◦  → 7
```

it produces the suggestion

```
1  the country of a branch is the country that ◦  → 10
   7  the country that ◦  → 10
```

The only way for larger suggestions to be created is for

```
7  the country that ◦  → 10
   1  the country of a branch is ◦  → 7
```

The waiting allows the previous suggestions to be pruned before the end of the sentence is reached. While this can be problematic, we believe the overall flexibility of the approach, compared to a completely specified grammar, outweighs the slight penalty of having to maintain certain suggestions for longer. A future enhancement would be to combine a kind of top-down approach with the bottom-up approach by allowing the caught suggestions to catch further suggestions instead. For example, the suggestion

```
1  the country of a branch is the country that ◦  → 10
```

would be created instead that could continue to be extended. However, this approach was not taken since it may lead to duplicate suggestions created via alternative paths and constantly comparing suggestions to eliminate duplicates would be inefficient.

---

8In fact it would not actually be allowed in the presence of required expectations.
The country of a branch is the country that is of the operating company that includes the local area that includes the branch.

**FIGURE 5.14:** Parse tree with deeply nested relative clauses.
Chapter 5. Application to Specifications using SBVR

The pruning strategy is as follows:

- Suggestions that fail semantic analysis (i.e., no configuration can be found) are pruned.
- Suggestions are deemed unimportant and pruned if:
  1. they have global coverage < 0.5, and
  2. their distance to the current word is > 3–9 (scales up as the length of the sentence increases).
- Suggestions whose catch or caught weight function results in a value < 0.1 are pruned.

5.4.3 Maximum Catching Distance

Since we are focusing on the precise interpretation of a CNL, we are very restrictive for the maximum catching distance. Therefore, the maximum allowable catching distance for this application is zero, with one exception: modal and negation keywords may have a catching distance up to two if they occur within a fact type annotation. This is to allow for modalities and negations to appear after the auxiliary verb of the passive form of a fact type, e.g. ‘...is always included in ...’. Figure 5.15 illustrates the syntactic analysis of a sentence including such a construction. Since we identify the full span of a fact type with the JAPE grammar, fact types are allowed to have a gap after the auxiliary verb, also of up to a distance of two. The distance of two allows possible combinations of modality and negation: e.g., ‘...is not always included in ...’, or ‘...is always not included in ...’, while excluding complex and difficult to interpret forms such as ‘...is not always not included in ...’. For this the slightly more understandable form ‘It is not the case that ...is always not included in ...’, or better yet the equivalent ‘...is possibly included in ...’.

5.4.4 Catch/Caught Weights

Due to the restrictions of the SBVR meta-model, we manually defined some catch and caught weights to have values below the pruning threshold to exclude obviously incorrect suggestions from being created in the first place. This prevents the suggestions from needing to be
configured by the semantic analysis to determine that it is invalid, thereby improving the performance of the prototype. For the most part, these weights are associated with the lexical entries of the different types of keywords; however, there is a general catch weight function for lexical entries of fact types as well. The weight functions for the different type are as follows:

**Quantifications:** prefer to catch suggestions that profile object types. In addition, quantifiers with a cardinality, e.g. ‘at least $n$’, can only catch numeric individual concepts for the expectation linked to the cardinality.

**Definite References:** (e.g. ‘the’, are basically quantifiers with a single referent) prefer to catch suggestions that profile object types, but can also catch individual concepts.
Anaphoric Reference: (basically just a binding relationship with an unconnected end) can only catch object types and cannot be caught by restrictions.

Restrictions/Relative Clauses: (introduced by ‘that’, ‘who’ or the allowed reduced relative clauses) prefer to catch quantifications (including definite references) with its backward expectation.

Modalities: prefer to catch non-keywords and cannot be caught by anything except negation as they have the widest scope.

Logical Operations: prefer to be caught by other logical operations.

Conjunctions and Disjunctions: prefer to catch smaller suggestions, i.e., they prefer more local scope.

Equivalences: (if and only if) prefer to catch full logical formulations, i.e., not object type or quantification suggestions.

Fact Types: prefer to catch non-fact types.

5.4.5 Implemented Algorithm

While we described the expectation-based parsing algorithm in Section 4.3, it was only a general algorithm that did not take into account any application specific considerations. In particular, the form of the token stream and the suggestions can affect the specific implementation of the algorithm. Before describing the implemented algorithm, we will consider these aspects.

As shown in Section 5.4.1, the token stream consists of word-level annotations, including multi-word expressions with gaps and internal expectations, created by applying the JAPE grammars of the lexicon. Although the difference between word-level and morpheme-level token streams is quite minor, there are potential issues with overlapping annotations due to the inclusion of multi-word expressions containing gaps. To avoid these issues, and reduce the number of unnecessary suggestions that are created, we preprocess the word annotations for
Section 5.4. Syntactic Analysis

a sentence before analysing the resultant token stream. The preprocessing is quite intuitive. Consider the annotated sentence:

Each city branch is always included in exactly one local area.

Assuming that $\square$, $\bigcirc$, and $\triangle$ represent ‘Word’ annotations for object types, we can simplify the token stream rather than trying to combine ‘city’ and ‘branch’ through the expectation-based parsing algorithm. To do this we simply remove annotations $\square$ and $\bigcirc$ from the token stream, leaving only the most specific object type annotation, ‘city branch.’ This is not just a straightforward “longest match wins” situation though. For example, if annotation $\square$ were for a fact type, the annotations would not be removed. In such a case there is an ambiguity that must be resolved (do ‘cities branch’ or is it a ‘city branch’?) by the parsing process. In contrast, the overlap of annotations $\bigcirc$ and $\triangle$ is a simple “longest match” situation.

Annotations $\square$, $\bigcirc$, and $\triangle$ overlap as well. Since there would be an incomplete analysis without the fact type annotation $\triangle$, we can safely remove annotation $\square$ (which is also for a fact type) but the annotation for the ‘always’ keyword must be kept; it would be the same for any keyword annotation within the span. A similar situation occurs with annotations for fact types with internal expectations. In such situations, the ‘Boundary’ annotations (seen in Listing 5.1 on page 127) are used to determine if the overlap is with the terms of the fact type or with the gap for the internal expectation; those annotations in the gap must be kept, unless they have other overlaps that cause them to be removed.

Finally, a simplification similar to the first can be performed for annotations $\square$, $\bigcirc$, and $\bigcirc$. While not all three are keyword annotations, $\square$ is for an individual concept, the nature of the quantification keywords allows $\square$ and $\bigcirc$ to be removed. After this preprocessing, the token stream for the sentence would appear with almost no overlaps as follows:

Each city branch is always included in exactly one local area.

The other aspect that needs to be considered is the structure of the suggestions. The general algorithm assumed simply structured suggestions; however, for practical purposes we implemented a more complex suggestion structure similar to a packed representation used in chart
Chapter 5. Application to Specifications using SBVR

parsing for the same purpose [All95]. This is particularly important to more efficiently handle
the high level of polysemous fact types in SBVR-based texts. The definition of suggestions for
the implementation is given in the following.

Definition 5.1 (Suggestion). A suggestion \( s \) is a frame:

\[
s = \begin{cases}
\text{SUGGESTION} & LE \mid LE \in \mathbb{LE}^2 \land \forall \le_1, \le_2 \in LE \\
& \text{type}(\le_1) = \text{type}(\le_2) \land \\
& \left\{ \begin{array}{l}
\text{pge}_1 \mid \forall \ge_1 \in \le_1.\text{expectations} \\
\text{pge}_1 = \text{proto}(\ge_1)
\end{array} \right. \\
& \left\{ \begin{array}{l}
\text{pge}_2 \mid \forall \ge_2 \in \le_2.\text{expectations} \\
\text{pge}_2 = \text{proto}(\ge_2)
\end{array} \right.
\end{cases}
\]

\[
\begin{array}{l}
\text{catch} \mid \text{expectation} \quad \text{base.expectations} \\
\text{suggestion} \quad \mathbb{S} \setminus \{s, s.\text{base}^+\} \\
\text{catchingDistance} \quad \mathbb{Z}
\end{array}
\]

\[
\begin{array}{l}
\text{base} \quad \mathbb{S} \setminus \{s, s.\text{catch.suggestion}\} \\
\text{start} \quad \mathbb{Z}^+ \\
\text{end} \quad \mathbb{Z}^+ \\
\text{bindingEnergy} \quad \mathbb{Z} \\
\text{localCoverage} \quad \{0, \ldots, 1\}
\end{array}
\]

where \( \mathbb{LE} \) is the set of all lexical entries, \( \mathbb{S} \) is the set of all suggestions, and the function
\( \text{proto} : GE \to GE \) creates a prototype expectation by excluding the elaboration site, i.e.,
\( \text{proto}((\text{index}, \text{type}, \text{required}, \text{es})) = (\text{index}, \text{type}, \text{required}, \emptyset) \).

The basic idea is that two suggestions can be collapsed if and only if their lexical entries
are of the same type (e.g. object type, fact type, keyword) and they have the same set of
grammatical expectations, ignoring their elaboration sites. For example, the initial suggestion
\( 3 \bullet \) is included in \( 4 \bullet \), given two lexical entries \( \le_1 = \text{'local area is included in operating company'} \) and \( \le_2 = \text{'branch is included in local area'} \), would appear as in Figure 5.16.

During processing, each time a catch is performed a new suggestion is created that points to
both the suggestion it was created from and the suggestion that was caught through the base
and catch attributes, respectively. In addition, the expectations and catches are updated by
excluding the catching expectation from the set of expectations and assigning it to the catch
of the new suggestion. Figures 5.17 and 5.18 illustrate the composite suggestions for the first
Section 5.4. Syntactic Analysis

part of the sentence shown in Figure 5.15 on page 133. This composition of suggestions gives rise to the parse tree, in which the catch links provide the dependencies (i.e. the lines with arrows) and the base links provide the compositional structure as illustrated in Figure 5.19.

Now that we have considered the structure of the suggestions and how they are composed to form the parse tree we can explain the full expectation-based parsing algorithm. The general algorithm and its supporting functions are shown in Algorithms 1 to 3. Note that functions in italics have been described or illustrated elsewhere, while functions in typewriter font are specified in Algorithms 2 and 3. While the implementation of expectation-based parsing follows the general algorithm for the most part, there are some minor differences. For example, the agenda is not utilised for the backwards expectations. Instead, the backward catches are searched recursively and all of the new suggestions created by this search are added to the agenda for checking against internal and forward expectations. This is straightforward as backward expectations can only occur when a new token is evoked. Moreover, it allows some slight efficiencies to be gained as the suggestion list and agenda are indexed by position. Therefore, the recursive backward search is able to begin where the previous suggestion left off due to the way the suggestions are extended. Similarly, when checking forward expectations all of the suggestions ending at the certain position can be checked against the subset of suggestions in the agenda starting from that position forward.
Chapter 5. Application to Specifications using SBVR

F I G U R E 5.17: Composite suggestions for ‘each branch’.

ALGORITHM 1: Implementation of the expectation-based parsing algorithm.

INPUT: A token stream $TS = \langle t_1, \ldots, t_n \rangle$

OUTPUT: The suggestion list ($SL$) resulting from the parse.

1. $SL \leftarrow \emptyset$ \hspace{1cm} // Initialise the suggestion list
2. FOR $t_i \in TS$ DO \hspace{1cm} // Main loop of Figure 4.3
3. \hspace{1cm} $LE \leftarrow \text{evoke}(t_i)$ \hspace{1cm} // Step 1 of Figure 4.3, empty agenda
4. \hspace{1cm} $s \leftarrow \text{initSuggestion}(LE, t_i)$ \hspace{1cm} // Step 1 of Figure 4.3 con., empty agenda
5. \hspace{1cm} $SL \leftarrow SL \cup \{s\}$ \hspace{1cm} // Step 2 of Figure 4.3
6. \hspace{1cm} $\text{Agenda} \leftarrow \{s\} \cup \text{catchBackwards}(s, SL)$ \hspace{1cm} // Step 3 of Figure 4.3
7. $SL \leftarrow SL \cup \text{Agenda} \cup \text{catchInternalAndForwards}(SL, \text{Agenda})$ \hspace{1cm} // Step 4 of Figure 4.3
8. $SL \leftarrow SL \setminus \{s_i \in SL \mid \text{shouldPrune}(s_i)\}$ \hspace{1cm} // Step 5 of Figure 4.3
9. END-FOR
Section 5.5. Semantic Analysis

Similar to the syntactic analysis, there are some aspects of the semantic analysis that must be specified for the SBVR-based implementation of the framework. In the following we will introduce the specifics of the three types of configuration knowledge (configuration model, solution knowledge, and requirements knowledge) that are needed to perform semantic analysis for SBVR. In addition, we provide some concrete examples of the configuration with SBVR and, finally, the handling of configuration failures and corrections.
Chapter 5. Application to Specifications using SBVR

ALGORITHM 2: Function for backward expectations

INPUT : The catching suggestion, s, and the suggestion list, SL.
OUTPUT : A set of new suggestions, SN.

1 FUNCTION catchBackwards (s, SL) IS
   2 SN ← ∅
   3 IF ∃ ge . ge ∈ s.expectations ∧ ge.type = backward THEN // Step 3 of Figure 4.3
      4 SS ← {si ∈ SL | ∀ ge ∈ si.expectations . ¬ge.required ∧ // Step 3 of Figure 4.3
            5 ssi . end ≤ s.start ∧ ssi . end ≥ s.start − catch_max(s)}
   6 FOR si ∈ SS DO // Step 1a of Figure 4.8
      7 sn ← combine(s, si) // Step 1a of Figure 4.8, create composite
         suggestion
      8 IF valid(sn) THEN // Step 1b of Figure 4.3
      9      SN ← SN ∪ {sn}
     10 END-IF
    11 END-FOR
   12 END-IF
   13 RETURN SN

ALGORITHM 3: Function for internal and forward expectations

INPUT : The suggestion list SL and the Agenda.
OUTPUT : A set of new suggestions, SN.

1 FUNCTION catchInternalAndForwards (SL, Agenda) IS
   2 SN ← ∅
   3 FOR si ∈ Agenda | ∀ ge ∈ si.expectations . ¬ge.required DO // Step 2 of Figure 4.8
      4 SS ← {sk ∈ SL | ∃ ge ∈ sk.expectations . // Step 4 of Figure 4.3
            5 (ge.type = forward ∧ sk . end ≤ si . start ∧ sk . end ≥ si . start − catch_max(s)) ∨
               (ge.type = internal ∧ sk . currentEnd ≤ si . start ∧
                6 sk . currentEnd ≥ si . start − catch_max(s) ∧ sk . end > si . end)}
   7 FOR sk ∈ SS DO // Step 2a/b of Figure 4.9
      8 sn ← combine(sk, si) // Step 2a/b of Figure 4.8, create composite
         suggestion
      9 IF valid(sn) THEN SN ← SN ∪ {sn} // Step 2c of Figure 4.3
     10 END-IF
    11 END-FOR
   12 END-FOR
   13 SN ← SN ∪ catchInternalAndForwards (SL, SN)
   14 RETURN SN

5.5.1 Configuration Model Knowledge

The configuration model knowledge for this application comes from the SBVR meta-model introduced in Section 5.1.1. The entities, relationships, and constraints specified in the meta-model allow for the configuration of rich representations of rules. However, the original meta-model is defined in MOF and first had to be converted to the knowledge-base format of the COCOA configurator.

9Specifically, we use an ECore version of the meta-model, i.e., the EMF implementation of EMOF
The knowledge-representation of COCOA is component-oriented. As such, it supports the definition of (a taxonomy of) component types that a system (i.e. a model) can be built from, the ports with which components can be connected by, attributes which specify variable properties of the components, and various types of constraints on the configurations of components, port connections, and attribute values [SFH98].

This component-oriented representation is roughly compatible with the object-oriented MOF meta-meta-model, which is used to define the SBVR meta-model. In general, the following transformations from MOF to the COCOA knowledge-base are performed:

- **Class hierarchy** $\Rightarrow$ Component Type taxonomy
- **Objects** $\Rightarrow$ Components
- **Reference definitions** $\Rightarrow$ Port Types
- **Reference (instances)** $\Rightarrow$ Ports
- **Class Attributes** $\Rightarrow$ Component Type attributes
- **Object Attributes** $\Rightarrow$ Component attributes
- **Methods** $\Rightarrow$ Methods or predicates (for boolean methods)

For example, the class **meaning** from the SBVR meta-model is transformed into a component type named ‘meaning’ in the COCOA knowledge-base, while the **represents** relationship (i.e. reference) becomes a port type named ‘represents’.\(^{10}\) This basic transformation was performed through an automated model transformation using Epsilon \([Eclb]\).

Features of MOF that are supported through the OCL are also supported by COCOA; most important is the support for constraints (invariants). In addition, derived attributes and method definitions can be transformed. However, these transformations had to be performed manually as the OCL code had to be rewritten in Smalltalk for COCOA while making use of the specific methods that support the constraint propagation in the configurator.

\(^{10}\)It should be noted that the MOF version of the SBVR meta-model makes use of reified relationships, i.e., **represents** is actually a class in the meta-model, named **RepresentationRepresentsMeaning**, with two references: one from the relationship class to **representation** and the other from the relationship class to **meaning**. This does not adversely effect the transformation because it is defined in terms of MOF, not how it is used for a specific meta-model.
Moreover, although MOF and the COCOA knowledge representation language are largely semantically compatible, there are several differences that had to be manually adjusted for, including:

1. the SBVR meta-model makes use of multiple inheritance in a few places, while COCOA allows only single inheritance of Component Types;

2. only leaf Component Types can be instantiated in COCOA, i.e., non-leaf Component Types are implicitly abstract, while any class not explicitly specified as abstract can be instantiated from the SBVR meta-model; and

3. the configurator makes use of three-valued Boolean logic (true, false, undefined) for constraint evaluation during configurations, while the constraints defined in OCL are two-valued.

The first issue is handled in different ways depending on the situation. Where necessary, a class with multiple inheritance was split into multiple Component Types, one for each super class, that are connected by a special equivalence Port (Type). As a result, wherever one facet of the Component Type is required, the configurator would generate and enforce the constraints of the other(s). This method ensures that all of the attributes, references, and constraints from the super types are inherited. However, it adds the complexity of ensuring that the constraints navigate to the correct facet of a component. This option was taken for the concept situational role as it needed to inherit all of the elements of object type and role.

If the class with multiple inheritance only inherited a small number of references and/or constraints, but no attributes, from one of the super types then an alternative approach was used. Since the domain of a Port Type could be a set of (possibly disjoint) Component Types, rather than a single Class, the Port Types and constraints were modified to explicitly allow the Component Type that is reduced to single inheritance. This was done for individual concept with respect to its inheritance of bindable target since there was only one reference to bindable target: i.e., the reference that lets it be bound to a role of a fact type in a rule.

The last method of handling multiple-inheritance was used in cases where the super class is defined based on the state of the object. For example, a binary fact type is a fact type with
only two roles. This type of multiple inheritance situation is modelled with derived Boolean attributes combined with constraints. Since the derived attributes can be overridden with assigned values (either on a subtype or on a specific Component instance) this allows for consistency checking. For example, partitive fact type overrides the isBinaryFactType attribute with the value is true; therefore, if a partitive fact type component is connected to more than two roles a constraint violation is detected and assignment fails.

The issue of implicit abstract component types was resolved by automatically creating a sub-Component Type for each non-leaf class not specified as abstract. This allows the class to be instantiated as itself, without having to be one of the other subclasses that are leaves. For example, for the concept object type, a new sub-Component Type ObjectTypeImpl is created.

Finally, the difference between Boolean logic required all of the constraints to be appropriately formalised to ensure that the correct results were obtained. That is, the constraints are specified such that they return true only when they can no longer be made false (excluding through backtracking), they return false when needed (which causes backtracking), and they return undefined when there is not yet enough information to determine whether they should return true or false (e.g. a port used in the constraint may not yet be connected to a component). This was a manual process as it was infeasible to attempt this through an automated model transformation.

Figure 5.20 displays the Component Type taxonomy, including attributes, of the SBVR meta-model as a COCOA knowledge-base; the (reified) relationships have been excluded for brevity. There are a couple of small additional changes to the original SBVR meta-model to make things easier to handle. For example, the SbvrRelation type has been added as a supertype of all of the reified relations, and SbvrObject has been added as a supertype of everything. This allows for constraints that apply to everything (e.g. those related to catch relationships) to be applied more easily. In addition, some names have been prefixed with ‘Sbvr’ to prevent name clashes with the underlying Smalltalk environment.
Figure 5.20: SBVR meta-model as a COCOA knowledge-base. Component type taxonomy and attribute definitions only.
5.5.2 Configuration Solution Knowledge

As in the general framework, the configuration solution knowledge consists of a partial model, in this case a partial SBVR model, generated by mapping rules from the syntactic analysis. For this application, only the model elements of the rule currently being processed are considered as the context. Figure 5.21 illustrates some of the mapping rules used to generate partial models in the SBVR-based application. The mapping rule for object types and individuals simply profiles the concept itself, while the mapping rule for fact types must instantiate new elements of the SBVR meta-model to allow the fact type to be used in the logical formulation of a rule. In general, the mapping rules for closed class terms (keywords) simply instantiate a new copy of the SBVR meta-model elements represented by the evoked semantic structure.

A partial SBVR model created by the evocation of the vocabulary for the rule ‘Each branch is included in exactly one local area,’ is displayed in Figure 5.22. For brevity, only the relevant elements of the vocabulary are included. As before, the partial model is valid with respect to the relaxed SBVR meta-model, but additional components and relationships are required to find the complete interpretation of the rule.

While these rules are quite simple, some more complex rules are required to support ellipsis in coordination. Since SBVR SE allows ellipsis in coordination, there is a mismatch between the structures evoked by the syntactic analysis and those required for the complete configuration. That is, a natural language rule such as ‘EU-Rent operates out of Australia and New Zealand and . . . ’ must be represented in the model with distinct elements as ‘EU-Rent operates out of Australia’, ‘EU-Rent operates out of New Zealand’, etc. For this we specify mapping rules on non-leaf nodes with a conjunction (or disjunction) semantic structure that duplicate the necessary elements of the atomic formulation and required bindings.

5.5.3 Configuration Requirements Knowledge

To represent the configuration requirements knowledge, we extended the COCOA knowledge-base for SBVR with additional reified relationships and associated constraints. As for the general framework, we added a relation to represent the catch constraints, called catches, and a word order relation, called precedes. These relations are shown for the example rule in
Figure 5.21: Suggestion mapping rules for SBVR for a given suggestion, s.

Figure 5.23 The general constraint for the catches relationship is that the two elements must be connected through an SbvrRelation; it is then left to the configurator to determine which relation should connect the two.

There are some restrictions to which SbvrRelations can be used to connect two components during configuration. This is because there are some very generic relations between Things (i.e. the top-most SBVR concept), which would be able to be generated ad infinitum if left unchecked. One such relation is the ‘thing is thing’ relation, which represents that two things the same. Any thing can have any number of these relations between one another; therefore, we restrict these relations to only be added to the model if they are explicitly in the text (e.g. ‘the operating country ... is the country that ...’) or reasoning performed outside of the configuration.
Section 5.5. Semantic Analysis

Figure 5.22: Partial SBVR model with evoked structures. Black bordered elements are the evoked structures, empty boxes are elements from the vocabulary, filled boxes were created by the evocation of the lexical entries. Dashed arrows are relations between different vocabulary entries.

Figure 5.23: Partial SBVR model with evoked structures and *catch* and *precedes* constraints. Black bordered elements are the evoked structures, empty boxes are elements from the vocabulary, filled boxes were created by the evocation of the lexical entries. Dashed arrows are relations between different vocabulary entries.
The precedes relation affects how some of the elements can be connected, but is not associated with a general constraint like the catches relation. For example, a constraint on quantifications utilises the precedes relation to enforce a more strict interpretation of the scoping (through the SBVR scopes over relation), similarly to how it was used to affect the scope of the contexts in the Conceptual Graphs example.

### 5.5.4 Configuration Result

Now that the required knowledge has been specified for the SBVR meta-model, the configurator can be used to find interpretations of the controlled natural language rules. The complete configuration for the example is shown in Figure 5.24.

### 5.5.5 Interactive Feedback, Alternatives, and Corrections

The SBVR meta-model is much more expressive and has many more constraints than the simple Conceptual Graphs model we presented in Chapter 4, which allows more useful feedback to be presented when processing SBVR SE texts. In particular, one useful extension is the support for selectional restrictions in the semantics. For example, the configuration of rules using the fact type ‘branch is included in local area’ is only successful if branches...
Section 5.5. Semantic Analysis

(including more specific types of branches, e.g. receiving branches) and local areas (including more specific types of local area, if they exist) are referred to in the appropriate roles. This allows feedback on whether or not rules and facts have been supplied that are consistent with the definitions of the vocabulary.

Here we will consider three types of errors that may occur and the feedback and/or corrections that can be made. The three cases include: (1) there is an error in the rule or definition, for example, an incorrect term has been used in the role of a fact type; (2) there is an error in the lexicon created from the vocabulary, for example, a fact type learnt as an object type; or (3) a definition or rule has been specified informally, i.e. using terms not formally included in the vocabulary.

In the case that there is an error in the rule or definition (assuming correct vocabulary), alternative lexical entries can be tested to see if they fit “better”. This can be done by relaxing the appropriate parse tree constraint and using the configurator to search for the possible compatible lexical entries. For example, for the erroneous rule ‘Each rental organisation unit is included in exactly one local area’, the configuration of the model with relaxed constraints would suggest that ‘rental organisation unit’ be changed to ‘branch’ instead (due to the fact type ‘branch is included in local area’). Since not all of the catch constraints would be relaxed ‘local area’ would not be suggested because there is no fact type ‘local area is included in local area’.

There are a number of errors that may occur in the learnt lexicon, e.g. a binary fact type learnt as a unary fact type, many of which may be identifiable and correctable by applying the configurator in conjunction with analysis of the parse tree and suggestion list. For example, by relaxing the catch constraints and adding constraints to prevent connections to existing vocabulary elements, a semantic structure can be generated that will suggest what type of lexical entry should occur as part of the rule. For example, if the object type ‘local area’ were incorrectly learnt as an individual concept, the rule ‘Each local area is included in exactly one operating company’ would not be configured correctly. However, by removing the constraints associated with ‘local area’ and preventing connections to existing vocabulary, we can use the configurator to generate a possible solution including an object type appropriately connected to
other elements of the SBVR model. This “new” vocabulary element could then be suggested to the user as a correction to ‘local area’.

In addition, analysis of the parse trees and suggestion list itself may be necessary to identify and correct errors. If, for example, the fact type ‘is included in’ has been learnt with the incorrect number of roles and placeholders, then the suggestion list for the rule ‘Each local area is included in exactly one operating company’ could be analysed to help determine where the placeholders should be. In this rule, the keywords quantify the object types that are the placeholders for the fact type and clearly delineate the second placeholder, even if the term has not been learnt at all. Therefore, if the second role were not present then the suggestion list would appear as in Figure 5.25, where there is no suggestion that covers the entire sentence. Instead there are two “large”, disjoint suggestions. This indicates that there should be a second fact type role and associated object type for ‘operating company’. If, on the other hand, ‘local area’ were omitted from the vocabulary then it would be uncertain as to whether the placeholder between ‘Each’ and ‘exactly one operating company’ is ‘local’, ‘local area’, ‘local area is’, or ‘local area is included’. In that case, automatic correction is unlikely; however, the user can still be given advice that a role or object type may be missing.

Finally, a definition may be specified informally, while rules should always be specified formally. Therefore, an informal rule indicates missing vocabulary and we can automatically attempt to create lexical entries for unknown words using the configurator as discussed above. However, since definitions can be intentionally informal we do not automatically generate lexical entries. Instead, informal definitions can be presented to the user for review, with the

\[\text{Figure 5.25: Example suggestion list with disjoint suggestions}\]

\[\text{SBVR}\]
option to mark them as intentionally informal or to create the lexical entry for them (if the suggested lexical entry is correct).

This capacity to provide detailed feedback to the user on errors and inconsistencies is an important feature of our approach, which is not present in many others. It is critical for the formalisation of specifications by business users as technical modelling knowledge is not required. Moreover, it supports the revision of less restricted language into a more restricted, less ambiguous, controlled form: [SBVR SE] in this case.

In most cases the effort required by the user to revise an erroneous sentence will be to select the intended meaning from a set of solutions suggested by the configurator. In other situations, new vocabulary may need to be added by the user or manual revisions made taking the advice of the produced error(s) into account. However, due to the presence of keywords that are part of the underlying semantic model, we posit there will often be enough of a partial parse to enable (partial) inference of the missing information, which can be further elaborated by the user.

### 5.6 Lexicon Acquisition

With this approach to controlled natural language processing, its success is dependent on providing the system with knowledge. This is due to the emphasis on semantics, with the approach being driven by both the meta-model and the instance models providing the content to reason with. Therefore, it is important to provide users, who are intended to be non-technical, with the ability to provide the vocabulary and knowledge necessary to perform the analysis of rules and specifications.

To achieve this, we propose two complementary methods for the acquisition of business/domain vocabularies. The first is based on the processing of explicitly defined glossaries that form part of a specification being analysed. This approach is supported by the simplified lexicon and semantics driven approach of our framework. The second method uses an [IE]-based approach to identify candidate vocabularies from unrestricted text. This is designed for situations where a glossary is unavailable or it is infeasible for one to be produced from scratch.
These approaches require little or no technical knowledge for acquiring the vocabulary and, hence, support criterion H5 of our goals.

5.6.1 From Glossary

In this process, a preliminary SBVR model and lexicon are created from the glossary. Like a standard glossary, a basic vocabulary entry includes the word (or phrase) and its definition. Since the definition can only be completely parsed once we have a lexicon, we initially use heuristics and other structural elements of the glossary, if present, to determine the most likely lexicon entry for a word. The initial entries are then refined further as definitions, and rules are processed. This allows our approach to maintain its flexibility with respect to document format, where the different SBVR elements for a vocabulary entry may not be included. That is, this method is equally applicable to a simple list of terms as it is an SBVR SE-based vocabulary specifying a definition, concept type, etc. The initial lexicon is acquired as follows:

1. **Add temporary lexical entries**: For each vocabulary entry identified in the document, create a temporary lexical entry without any associated semantics.

2. **Disambiguate concept types**: For each temporary lexical entry, determine which main category of SBVR concept type it most likely belongs to, i.e. object type, individual concept, or fact type (or one of their subtypes). The heuristics used to make these determinations and the potential issues with them are discussed below.

3. **Create the semantic representations**: Create the model elements for each concept using templates for the different SBVR concept types. The separate vocabulary entries are also linked appropriately, for example, the roles of fact types are connected to the object types that they range over.

4. **Add mappings to the lexicon**: Link each temporary lexical entry to its semantic representation. Synonyms are mapped to the same semantic representation rather than instantiating duplicates.

5. **Generate JAPE grammar**: Create the JAPE grammar from the newly acquired lexicon.
Steps 1, 3 and 4 are quite straightforward; therefore, the following will only describe steps 2 and 5 in details. When disambiguating which concept type(s) apply to a lexical entry, a number of features are taken into account, including:

- capitalisation, e.g. ‘EU-Rent’ vs. ‘EU-Rent operating company’ vs. ‘branch’
- the number of placeholders/roles identified
- the position in the expression of each identified placeholder
- any explicitly specified concept types
- any general concept (i.e. super concept); either specified explicitly as a separate field or identified as part of a definition
- the beginning of a definition, if provided
- whether or not the entry has been specified as the concept type for another entry (this occurs when categorising vocabulary entries using the SBVR SE style)

Capitalisation is used primarily to differentiate between object types and individual concepts, consistent with normal English. However, ambiguity may arise in the context of compound terms if the name of an individual concept is included in the term of an object type; for example, ‘EU-Rent operating company’ may be considered an individual concept, as it begins with a capital, when it is actually an object type representing an ‘EU-Rent owned company operating in a particular country.’ Therefore, the heuristic analyses how many words of a multi-word term are capitalised. Other information may also resolve this ambiguity. For example, the type could be disambiguated through a synonym that is clearly an object type, such as ‘operating company’ for ‘EU-Rent operating company.’ Finally, subsequent parsing of the definitions and rules may identify an inconsistency in the term’s usage and update the lexical entry accordingly.

For example, the definition of ‘EU-Rent AU’ is ‘the EU-Rent operating company that is located in Australia’, which only makes sense if ‘EU-Rent operating company’ is interpreted as an object type.

To distinguish fact types, the placeholders (i.e. terms for object types) included in them are identified. This is done by matching the text of the temporary lexical entries against one
another. For example, given the terms ‘branch’, ‘local area’, and ‘branch is included in local area’, the two placeholders ‘branch’ and ‘local area’ can be identified ‘branch is included in local area.’ If placeholders are identified it affects both the term that was matched and that term that the placeholder was found within. The former is more likely to be an object type, while the latter is more likely to be a fact type. A fact type can include one, two, or more placeholders for characteristics (i.e. unary fact types), binary fact types, and other fact types, respectively.

The position of the placeholder is taken into consideration when determining a fact type based on placeholder identification. For example, if a placeholder is found only in the middle or at the end of a term, it is unlikely to actually be a fact type (as in the case for ‘EU-Rent operating company’ and ‘operating company’), while if at least one placeholder occurs at the very beginning of the term then it is more likely to be a fact type. This is intuitive due to the way verbs are used in English.

Identifying placeholders using this method may result in ambiguities. In particular it can be difficult to differentiate between characteristics and object types that are prefixed by a matching term. For example, the object type ‘EU-Rent operating company’ could be identified as the characteristic ‘EU-Rent operating company’. Conversely, if a term has been omitted from the vocabulary, a fact type could be identified as an object type or as a fact type with one less placeholder. In these situations, we rely on other information to help disambiguate the two. Otherwise, the processing of the definition and/or rules may identify inconsistencies in its usage and be able to update the lexical entry accordingly. For example, if ‘local area’ were not included in the vocabulary, ‘branch is included in local area’ would be identified as a characteristic. This would lead to an inconsistency with the rule ‘Each branch is included in exactly one local area’, where the ‘exactly one’ causes a separation in the characteristic and indicates that ‘local area’ should be defined as an object type. Such situations were discussed in [Section 5.5.5]

Another source of information is the definition, even if it cannot initially be parsed completely. This is because there are some common ways of beginning definitions that can indicate the type of a lexical entry, along with its general concept, particularly for object types and individual concepts. The definitions are checked to see if they begin with a variant of ‘[a/the (term)] [is]
a/the (something) that . . . ', if they do then it indicates that the lexical entry is an object type (if ‘a’ is used) or an individual concept (if ‘the’ is used).

The benefit of this approach is that it does not depend on any external lexical resources, which means that: (1) the process focuses on the intended, domain specific meaning of terms rather than the potentially dozens of meanings in a general purpose resource, and (2) domain specific terms that do not exist in the general resources can be handled. Moreover, the ability to acquire lexicon from a glossary using virtually no linguistic or technical information (since most SBVR SE fields are optional) enables non-technical experts to easily define their domain/business vocabulary.

The limitation, though, is that fact types must be expressed in the vocabulary using the fact type form (e.g. ‘term verb term’) as opposed to only the verb itself. However, this is considered beneficial in the development of a formal vocabulary and the usage-based approach is still simpler than other lexical approaches, which require complex linguistic information to be specified.

### 5.6.2 From Unrestricted Text

Manually defining formal vocabularies is a beneficial, but time consuming task. Since one of our goals is to reduce the amount of time it takes to formalise a specification, it is appropriate to incorporate a method for suggesting candidate vocabulary entries from existing documentation. Moreover, doing so supports criterion H4: the analysis of existing documentation without rewriting it manually first. Therefore, in this section we present an IE-based approach to suggesting candidate vocabulary/lexical entries.

Since the candidate vocabulary entries are provided with SBVR semantics, it is very similar to the IE approaches that attempt to create UML class models or ER models from unrestricted texts. Therefore, we analysed the IE approaches surveyed in Chapter 3, identified common extraction rules, and synthesised an approach suited to SBVR vocabulary models. The resulting method extracts less information than some of the surveyed approaches, for example we do not extract cardinalities; however, its focus is on identifying the core vocabulary elements that will allow CLUE4SBVR to process the text. Moreover, the limitations of the
IE component are overcome by combining it with the primary parsing component, which handles cardinalities for example, resulting in an overall very capable approach for formalising specifications.

5.6.2.1 General Process

Like many approaches (e.g. [FMP11] [FKMS+07] [SDGG13] [LZC13] [SA14]) we utilise a dependency parser and a set of extraction rules over the dependencies to identify candidate vocabulary. The dependency analysis provides the grammatical relations between words (e.g. subject, object, prepositional object), which directly link words thereby simplifying the extraction of relationships compared to traversing the abstract categories of a constituency tree. Specifically, we use the Stanford Parser with the Recurrent Neural Network (RNN) model [CM14]—their most accurate parsing model—to perform the dependency analysis.\(^{12}\)

The Stanford Dependencies\(^{13}\) are described in [MM08]. An outline of the process is given below:

1. Tokenisation and sentence splitting is performed on the document(s).
2. Run the Stanford Parser on each sentence.
3. Execute pre-processing rules to normalise the dependencies produced by the Stanford Parser.
4. Execute extraction rules to identify candidate terms as either candidate noun concepts or candidate verb concepts.
5. Filter the candidates based on core SBVR vocabulary and other linguistic exclusions (rules and predefined lists that remove some erroneous candidate terms).
6. Disambiguate and resolve candidate compound noun and adjective-noun concepts.
7. Resolve candidate verb concepts against noun concepts.
8. Determine object type/individual concept status of candidate nouns concepts.

\(^{12}\)The Stanford Parser also produces a constituency parse tree, so we can use a combination of both in the extraction rules if desired.

\(^{13}\)This work was performed before the update to the new Universal Dependencies.
Section 5.6. Lexicon Acquisition

9. Final filter of object types, individual concepts, and fact types.

10. Present candidate vocabulary to the user for review.

5.6.2.2 Initial Parsing

As we did in CLUE4SBVR, we perform tokenisation and sentence splitting using GATE. Furthermore, the Stanford Parser has been integrated into GATE as a plugin. Therefore, after the parsing is performed, all of the POS, constituency, and dependency information is attached to the document as annotations. Finally, the information stored in the annotations is used to generate a Flora2 file, since we use Flora2 to perform the extraction as the code is more maintainable than writing the extraction rules in Java.

5.6.2.3 Pre-processing

To account for some inconsistencies in the generated dependencies and to simplify some of the extraction rules, we rewrite some dependencies to normalise them in a pre-processing step. This is necessary as the dependencies created by the Stanford Parser can vary greatly for similar constructs occurring in different sentences, making it more difficult to write the extraction rules as they must be filled with lots of special cases. These inconsistencies could result in missing terms that should be extracted. Some of the pre-processing rules rewrite dependencies that are not necessarily incorrect, but change them into a form more suitable to candidate suggestion for SBVR vocabularies: for example, converting ‘exactly ...’ phrases from adverbial modifiers to quantification modifiers.

The pre-processing rules fall into several broad categories:

- Corrections for issues with punctuation (e.g. percentages and slashes);
- Corrections for preposition attachment issues;
- Corrections to misidentified relative clauses; and,

\footnote{Flora2 is an object-oriented logic programming language/engine based on F-logic and implemented on top of XSB Prolog.}
• Corrections that support extraction of SBVR concepts/vocabulary.

Punctuation

The following rules are used to address various punctuation issues:

Pre-processing Rule 1 (Percentages to Quantifier Modifier).

\[ y/CD \times/\% \text{of} \ z \quad \Rightarrow \quad y/CD \times/\% \text{of} \ z \]

In addition, all of the dependencies pointing to the percent symbol are redirected to the quantified term (z).

Pre-processing Rule 2 (Fix compound terms surrounding slash ('/')).

\[ x/JJ \ y/NN \ '/' \ z/NN \quad \Rightarrow \quad x/JJ \ y/NN \ '/' \ z/NN \]

In addition, all of the dependencies to and from y are moved to the new head noun z.

The result of this rule is a compound noun that incorporates the slash. There are several more rules similar to this one to fix different cases where the dependencies are inconsistent. We illustrate only one for brevity.

Preposition Rules

These rules fix various prepositional attachment issues.

Pre-processing Rule 3 ('Up to' Quantifier Modifier).

This rule converts 'up to' phrases that have been incorrectly labelled as prepositions (due to the ‘to’) to be a quantifier modifier instead.
Pre-processing Rule 4 (‘of’ Prepositions spanning commas).

This rule moves the ‘of’ prepositional attachment to the nearest noun, rather than it spanning the entire, comma separated, coordinating conjunction.

Pre-processing Rule 5 (Quantity terms with multiple prepositions).

This rule fixes the situation where a “quantity” term (e.g. ‘maximum’, ‘number’, ‘minimum’) is associated with additional prepositions instead of only the ‘of’ linking it to the thing it is an amount of. For example, in ‘number of users with cash cards’ the word ‘number’ is associated with ‘cash cards’ instead of ‘users’.

Relative Clauses

There are some cases of complex sentences where relative clauses are not identified correctly, instead they are identified as an adverbial clause. These rules correct such issues.

Pre-processing Rule 6 (Correct adverbial clause as relative clause).
These rules resolve the situation where a relative clause has not been properly identified, but rather a combination of adverbial clauses have been instead.

**SBVR Related Pre-processing Rules**

These rules modify the dependencies to make it easier to extract relevant information in the creation of an [SBVR] vocabulary model.

**Pre-processing Rule 7** (Exactly quantifiers).

This rule converts ‘exactly . . . ’ phrases into quantifier phrases instead of adjective or adverbial modifier phrases. This removes incorrect or unnecessary terms (e.g. ‘exactly branch’, if it were labelled as an adjective modifier) and helps in the identification of an object type.

**Pre-processing Rule 8** (Fix preposition attachment for Categories).
Section 5.6. Lexicon Acquisition

This rule attempts to correct prepositional phrase attachments for words that form part of a category name, such as ‘Branches by Type’, as the preposition ‘by’ appears to prefer being attached to the verb (often as an ‘agent’ dependency) when another preposition is present.

These rules were created during the testing of the vocabulary suggestion in response to investigations into why terms that should have been extracted were not being identified. Therefore, they are unlikely to cover all the cases where such pre-processing is required. Furthermore, they may not correct in all cases, but were found to be correct more often than not during testing.

5.6.2.4 Candidate Extraction

Following the pre-processing step, we perform the core extraction of candidate vocabulary entries. This is done with rules that identify patterns of dependencies. These rules were developed by analysing the IE-based approaches included in the survey (see Chapter 3) that used a rule-based approach on dependency parses. Not all of the related work provided enough detail those that did include the works of: Anandha Mala and Uma [AU06]; Bajwa, Bordbar, and Lee [BBL10]; Sagar and Abirami [SA14]; Sarkar, Sharma, and Agarwal [SSA12]; Elbendak, Vickers, and Rossiter [EVR11]; and Harmain and Gaizauskas [HG03]. Another valuable reference that described their extraction rules in detail is the work of Zouaq, Gasevic, and Hatala [ZGH12]. A key difference between our approach and that of the last is that we aim to break down long dependencies into the core vocabulary, while [ZGH12] attempt to find as many (binary) relations between concepts as possible for their OWL ontology and, therefore, create large compound relationships.

Our analysis of these works showed that there were many common rules used for the identification of object-oriented models from text. The core rules that were common across the approaches include:
• Nouns (possibly compound nouns) become classes (object types) if they have attributes and/or are used as the subject of a verb.

• Adjectives indicate attributes.

• Possessives become attributes.

• Verbs become relationships or methods, where the subject of the verb denotes the class that owns it; the object of a transitive may become an argument or the other end of the relation depending on the approach.

These rules form the basis of our vocabulary suggestion mechanism. However, since SBVR is an attributeless model, attributes and relations are both represented as fact types. Applying the extraction rules to a document results in a set of annotations indicating the matches and the candidate terms that have been discovered. In the following we present the specific rules used to extract the candidate terms.

Rules for extracting Candidate Noun Concepts

The first group of rules attempt to extract candidate noun concepts, i.e., object types and individual concepts.

Candidate Noun Rule 1 (Simple Noun). Nouns that are subjects or objects of a verb are candidate noun concepts. This includes a variety of subject and object dependencies such as:

• nsubj and nsubjpass

• dobj, iobj, and pobj

• root, and cop (for sentences containing only a copular verb)

• rcmod

Nouns that are only ever the dependant of an object dependency are more likely to be situational roles rather than object types. This is similar to the class/attribute distinction made by other approaches.
Candidate Noun Rule 2 (Simple Compound Nouns). Compound nouns, in which the head noun is the subject or object of a verb as in Noun Rule 1, are candidate noun concepts. This rule excludes compound nouns containing coordinating conjunctions (e.g. ‘and’).

Since the structure of a compound noun is not analysed by the Stanford Parser, the compound noun dependencies (nn) are always associated with the rightmost noun of the noun phrase [MM08]. However, noun compounds can have complex internal structure and, therefore, we extract a set of possible interpretations that are disambiguated across the document after the extraction is complete. This will be discussed in Section 5.6.2.5.

Due to some common incorrect POS assignments, we extract a compound noun even if the head word is tagged as an adjective (JJ), instead of a noun.

Candidate Noun Rule 3 (Coordinated Compound Nouns). Compound nouns that are involved in coordinating conjunctions, and in which the head noun is the subject or object of a verb as in Noun Rule 1, are candidate noun concepts.

The addition of the coordinating conjunction increases the ambiguity of the compound over those terms extracted by Noun Rule 2, therefore, the set of interpretations created for coordinated noun compounds is different.

Candidate Noun Rule 4 (Adjective-Noun Compounds). Nouns with adjectival modifiers (amod) are extracted as candidate noun concepts. Similar to noun-noun compounds, there can be some ambiguity with adjective-noun compounds, hence, we create a set of possible interpretations for these as well.

In the analysis produced by the Stanford Parser, it is not uncommon for words considered adjectival modifiers to be tagged with a verb form POS. To handle this we allow the adjectival modifier to be a verb gerund (VBG) or a past participle verb (VBN) in addition to the adjective parts-of-speech.

Candidate Noun Rule 5 (Nouns with prepositions). Nouns with a prepositional phrase attached are considered candidate noun concepts. This allows the extraction of terms such as ‘Branches by Type’ and ‘request for pick-up’ as concepts.

There are a few restrictions on what forms a valid candidate in this case. Primarily, the only dependencies that the prepositional object can be the head of are noun compound.
Chapter 5. Application to Specifications using SBVR

and adjectival modifier dependencies. This is based on the observation that if determiners, possessives, etc. exist on the prepositional object then it is unlikely to be a correct compound. For example, ‘request for a/the pick-up’ is unusual for a concept. Furthermore, quantity type terms are excluded, e.g., ‘number of’. Lastly, the prepositional object cannot be a pronoun.

Rules for extracting Candidate Verb Concepts

These rules extract candidate verb concepts from a document by looking for different kinds of verbs such as: transitive and intransitive verbs, verbs with prepositional phrases, copular verbs, and specific verbs and prepositions—e.g. ‘identifies’, ‘…of…’. The candidate verb concepts are maintained in active form where possible. We do so for two reasons: first, active form is preferred for SBVR vocabularies and, second, it helps to reduce the number of candidates by merging equivalent active/passive forms.

In many cases, the following rules make their best effort to find an associated subject and/or object by searching through parent or child clauses, such as adverbial clauses (‘advcl’ dependencies), clausal complements (‘ccomp’ and ‘xcomp’ dependencies), and relative clause modifiers (‘rcmod’ dependencies). For example, a verb without a direct subject dependency may find a subject by discovering the direct object of its parent external clausal complement (i.e. xcomp dependency). This is somewhat inspired by the extended relationship rules of [ZGH12]; however, the idea in this case is to find the fundamental fact types of the vocabulary rather than combining them all into a single binary relation. As a result, a verb can search through its child clauses and complements to find an object, while a child clause can search up through its parents to find a subject with each resulting in a new candidate verb concept.

The following rules are described quite abstractly, rather than specifying the exact dependencies that are identified in each case. This is because there are often several variations of a rule that have been implemented to cope with the different, but quite consistent, outputs of the Stanford Parser.

Candidate Verb Rule 1 (Intransitive Verbs). Intransitive verbs, i.e. verbs without a direct object, are extracted as candidate verb concepts. Specifically, they are candidate unary fact types. Moreover, intransitive verbs in this case cannot be attached to prepositions or clausal complements as they are handled by other rules.
**Candidate Verb Rule 2** (Copular Verb with Adjective). This rule creates candidate verb concepts (unary fact types) from the copular verb combined with an adjective. For example:

```
the/DT car/NN is/VBZ available/JJ \implies car is available
```

**Candidate Verb Rule 3** (Intransitive Verb with Preposition). Verbs with no direct object, but a prepositional object are identified to create candidate verb concepts that combine the intransitive verb with the preposition. For example:

```
the/DT car/NN transfers/VBZ to/IN a/DT branch/NN \implies car transfers to branch
```

**Candidate Verb Rule 4** (Intransitive Verb with Clausal Complement). In this rule a candidate verb concept (a binary associative fact type) is created from an intransitive verb with a clausal complement. Similar to Verb Rule 3, the clausal complement is combined with the intransitive verb to form the candidate verb concept. For example:

```
a/DT user/NN authenticates/VBN to/IN reach/VB a/DT web-page/NN \implies user authenticates to reach web-page
```

**Candidate Verb Rule 5** (Transitive Verbs). This is the basic rule to create a candidate verb concept from transitive verbs (i.e. verbs with a subject and a direct object), without incorporating prepositions or clausal complements. The candidates it creates are binary fact types. For example:

```
a/DT local area/NN includes/VBZ a/DT branch/NN \implies local area includes branch
```

**Candidate Verb Rule 6** (Transitive Verb with Preposition). Similar to Verb Rule 3, we extract candidate verb concepts based on the combination of a transitive verb with a preposition. The result is a ternary associative fact type, as in the following:
Candidate Verb Rule 7 (Transitive Verb with Clausal Complement). This rule is similar to Verb Rule 4, except that the head word is a transitive verb. The result is that the candidate verb concepts are ternary associative fact types. For example:

\[
\text{application asks user to confirm action}
\]

Candidate Verb Rule 8 (Passive Verbs). This is a general rule for extracting passive verbs, which are binary fact types and are associated with the active form of the vocabulary entry. It excludes prepositions (other than ‘by’). For example:

\[
\text{branch is frequented by customers and customers frequent branch}
\]

Candidate Verb Rule 9 (Passive Verbs with Prepositions). This rule creates candidate verb concepts from passive verbs that use prepositions other than ‘by’ (and possibly in addition to it). The result is a binary or ternary fact type and is in both active and passive form. For example:

\[
\text{branch is included in local area and local area includes branch}
\]
Candidate Verb Rule 10 (Verb Gerunds). Verb gerunds are analysed to extract candidate verb concepts. This differs from other approaches which typically identify them as candidate classes or concepts. The idea is to identify the underlying fact type, which can be nominalised in the text, if need be. For example,

\[ \text{a/DT user/NN } \ldots \text{ dropping/VBG a/DT player/NN } \implies \text{user drops player} \]

Candidate Verb Rule 11 (Possessives Rule). Similar to other approaches, we identify possessives to create candidate verb concepts, specifically an is-property-of fact type. For example,

\[ \text{the/DT car group/NN 's/POS pay rate/NN } \implies \text{car group has pay rate} \]

Candidate Verb Rule 12 (Explicit Properties). The rule for explicit properties extracts candidate verb concepts, i.e., is-property-of fact types\(^{15}\) from nouns with an ‘of’ prepositional phrase. This excludes quantity term such as ‘maximum of’, ‘number of’, etc.

\[ \text{the/DT pay rate/NN of/IN the/DT car group/NN } \implies \text{car group has pay rate} \]

\(^{15}\) Although, ‘of’ (and the inverse ‘has’) do not always bear the meaning of an is-property-of fact type, in the context of specifications it appears common enough to make that assumption—another common relation would be the is-part-of fact type. If it is incorrect it can be corrected by the user when they review the candidate vocabulary. Furthermore, future work can investigate more complex reasoning about the type of the suggested vocabulary entry.
Candidate Verb Rule 13 (Identification Properties). This rule is similar to Verb Rule 6, except that it identifies specific verb phrases that indicate an identifier of individuals: e.g., ‘identifies’/‘identified by’, ‘denotes’/‘denoted by’, and ‘recognised by’. Candidate verb concepts extracted by this rule are is-property-of fact types and are given the additional information of being a possible reference scheme for the identified type, which is how SBVR specifies the identifiers of an object type.

5.6.2.5 Disambiguation of Compound Nouns

Following the initial extraction of candidates, the candidate noun concepts created from compounds (i.e. noun compounds, adjective compounds, and coordinating conjunctions) are analysed to determine which interpretation should be accepted. This is due to the ambiguity surrounding the internal structure of such compounds, which is lacking in the dependency structure, giving rise to sets of possible interpretations. To resolve these ambiguities, we use an approach similar to that of [DRST13]; the difference being that we resolve the ambiguities across the entire document together, rather than on a case by case basis, and the sets of interpretations in our case are not necessarily mutually exclusive.

By selecting terms in this fashion, groups of related terms (even if they have been extracted from different locations in the document) are maintained. These groups can then be displayed to the user, allowing them to be refined into a more consistent form. Moreover, it allows for the system to provide a minimal set of terms or a maximal set of terms, whichever is desired by the user, and it allows certain terms/concepts to be flagged as needing further definitions. For example, in the case of ‘draft statement’, draft statement could be a specialisation of statement, with a flag indicating that further definition (formal or informal) may be required (i.e. ‘what makes a statement draft?’). If the characteristic ‘statement is draft’ were also provided then draft statement would be defined in terms of the (possibly primitive) characteristic, which would then be flagged as needing a definition.

The selection of candidate noun concepts as final candidates is determined predominantly by frequency of actual occurrence in the text. Actual occurrence is determined as the longest consecutive compound candidate that occurs for each particular annotation of the candidate noun concept—ignoring duplicates in cases where multiple rules have proposed the same
candidate. For example, consider the candidate noun concept annotated in the following sentence fragment:

```
Each draft statement is a statement that . . .
```

In this case there are two different terms with an actual occurrence: ‘draft statement’ and ‘statement’. Moreover, the set of possible interpretations for ‘draft statement’ is as follows.

\[
\begin{align*}
\text{draft} & \quad \text{statement} \\
\quad \text{amod} & \quad \{ \text{statement} \} \quad (1) \\
\{ \text{statement} \} & \quad \{ \text{statement, statement is draft} \} \quad (2) \\
\{ \text{draft statement} \} & \quad (3)
\end{align*}
\]

Since both ‘statement’ and ‘draft statement’ have actual occurrences, it is likely that both rows (1) and (3) will be selected as final candidates. If, on the other hand, the following sentence were used instead:

```
Each draft statement is a formal statement that . . .
```

then there would be no actual occurrences of ‘statement’, so only row (3) would be selected; ‘formal statement’ would most likely be selected as well, but independently.

The idea behind this approach is that we want to bias towards the terminology utilised by the domain experts, while supporting flexible NL text and creating a well defined SBVR-based vocabulary. If a domain expert consistently uses a specific compound noun, then it should be in the vocabulary. If they use it interchangeably with a characteristic (copular adjective form), then they should both appear in the vocabulary (linked appropriately).

In the following we consider the sets of possible interpretations for the three different noun compounds: noun-noun, coordinating conjunction-based, and adjective-noun.

**Noun-Noun Disambiguation** The resolution of noun-noun compounds is based on two interpretations of the compound: one where the entire compound is considered the term,
and one where the individual terms are considered separately with an is-property-of fact type between them. If the compound comprises more than two nouns, the different combinations are considered. For example,

\[
\begin{align*}
&\{\text{statement, account, system, account has statement, system has account}\} & (1) \\
&\{\text{account statement, system, system has account statement}\} & (2) \\
&\{\text{system account, statement, system account has statement}\} & (3) \\
&\{\text{system account statement}\} & (4)
\end{align*}
\]

In which each row is a group of terms that must be selected together, i.e., if row (1) were chosen as final candidates then all of the candidates—\text{statement, account, system, account has statement, system has account}—are chosen. However, multiple rows can be selected: for example, rows (2) and (4) could be selected, in which case \text{account statement, system, system has account statement, system account statement} are all considered final candidates and the remainder (i.e. \text{statement, account, account has statement}) are removed.

Since the noun-noun compounds are resolved shortest first, it may affect the selection of longer compounds. For example, if it has already been determined that \text{statement} and \text{account} are final candidates, then it may lead to row (1) being selected—most likely in addition to row (4). The result is that \text{system} would also be a final candidate, even if it does not actually appear in the document with no actual occurrences. While this seems to conflict with the aim of extracting the terms actually used by domain experts, it is potentially it serves to suggest potentially important terms that should be included in the formal vocabulary.

**Coordinating Conjunction Disambiguation** There are two different sets of interpretations for coordinated compounds, based on the pattern of dependencies the compound is extracted from. The difference arises from where the internal noun compound appears, i.e. before

---

\textsuperscript{16} It is not always the case that this is the correct relation between the terms. See \textsuperscript{Footnote 15} for a similar consideration.
the conjunction, or after. For example, the compound appears after the conjunction in the following:

\[
\begin{align*}
\text{location} \text{and} \text{navigation buttons} & \quad \text{nn} \quad \text{conj_and} \quad \text{nn} \\
& \quad \{\text{location, navigation, button,} \\
& \quad \text{location has button,} \\
& \quad \text{navigation has button}\} \quad (1) \\
& \quad \{\text{location button, navigation button}\} \quad (2) \\
& \quad \{\text{location and navigation, button,} \\
& \quad \text{location and navigation has button}\} \quad (3) \\
& \quad \{\text{location and navigation button}\} \quad (4)
\end{align*}
\]

In this case the noun-noun compound appears after the conjunction, creating four alternatives. Moreover, two patterns of dependencies are identified for this case due to inconsistencies in the output of the parser depending on the complexity of the sentence. For the second case, the noun-noun compound occurs before conjunction, as follows:

\[
\begin{align*}
\text{player statistics and points} & \quad \text{nn} \quad \text{conj_and} \\
& \quad \{\text{player, statistics, points,} \\
& \quad \text{player has statistics, player has points}\} \quad (1) \\
& \quad \{\text{player statistics, player points}\} \quad (2) \\
& \quad \{\text{player statistics and points}\} \quad (3)
\end{align*}
\]

For this case, only three alternatives are produced. While the last possibility, row (3) may seem unnecessary, it may occur if it is used in the sense of a proper noun, e.g. as the name of category, or a user interface element (when processing lower-level descriptions of requirements).

**Adjective-Noun Disambiguation**  Adjective-noun compounds produce a set of interpretations that consider the adjective(s) independently and as combined expressions. In addition, it includes the potential for a copular adjective form, as a unary fact type. In this way the two forms are related with one another. For example,

\[17^{17}\text{Ideally the second pattern would unambiguously identify the separate terms ‘location’ and ‘navigation buttons’; however, this is not always the case.}\]
Chapter 5. Application to Specifications using SBVR

Finally, adjective-noun compounds are disambiguated after noun-noun compounds. Therefore, if the head noun of the adjectival expression has previously been resolved to a multi-word expression due to the compound noun resolution above, then the adjectival modifiers will be resolved in the context of the compound noun. For example, if ‘statement’ were resolved to termaccount statement, then the adjective forms would be associated with the compound, i.e., new draft account statement.

5.6.2.6 Resolution of Verb Concepts

After disambiguating the compound nouns to provide a set of final candidate noun concepts, the candidate verb concepts must be resolved against them. For example, if player statistics is a selected compound noun and a verb candidate statistics are displayed has been extracted, it will be expanded to player statistics are displayed. Similar to the noun candidate resolution, this takes into account the actual occurrences of the candidate verb concept in the text.

When dealing with verb candidates, though, actual occurrence cannot be defined simply as the longest candidate annotation. Instead, actual occurrence for candidate verb concepts is based on whether or not there is a candidate verb annotation that overlaps with the largest of the final candidate noun concepts. For example, consider the annotated segment fragment below,

Each branch is included in [exactly one] local area.
in which annotations 1, 2, 3, and 4 are candidate noun concept annotations and 4 is for the candidate verb concept *branch is included in area*. Assuming that all of the candidate noun concepts are final candidates, the candidate verb concept that *actually* occurs in the given sentence is *branch is included in local area*. This is due to 4 being the longest match (or the candidate noun concept that actually occurs) as the object (i.e. second placeholder) of the verb concept. As a result, the original candidate verb concept (*branch is included in area*) will be replaced by the candidate that actually occurred (unless an actual occurrence in another part of the document prevents it from being removed). This ensures that, if a verb is only ever used by the domain experts in conjunction with a particular compound noun, then the resulting fact type will be defined with respect to that compound noun term. Furthermore, a candidate verb concept will be removed at this point if all of its placeholders reference candidate noun concepts that have not been selected as final candidates.

### 5.6.2.7 Object Type/Individual Concept Determination

For the candidate noun concepts, it is important to determine whether or not they represent object types or individual concepts. To determine the category of each candidate noun concept, we consider whether or not:

- the term is used as a proper noun
- the term is used in the plural form
- the term has any determiners (excluding ‘the’) or numeric modifiers
- the term is the placeholder in a copular adjective candidate verb concept, i.e. a ‘⟨noun⟩ is ⟨adjective⟩’ unary fact type
- the term has a high abstraction score

The information such as proper noun, plurals, and determiner usage are tallied up as they are used as a ratio of the count vs. the total number of references to the candidate noun concept being considered. A candidate noun concept is an then considered an individual concept if:

- the candidate is *not* a placeholder in a copular adjective unary fact type,
Chapter 5. Application to Specifications using SBVR

• the candidate is not used with any determiners or numeric modifiers,

• the candidate has a low abstraction score, and

• the number of proper noun references is greater than the number of non-proper noun references to the candidate.

or

• the candidate is not a placeholder in a copular adjective unary fact type,

• the candidate is not used with any determiners or numeric modifiers,

• the candidate has a low abstraction score, and

• the number of proper noun references is less than the number of non-proper noun references to the candidate, but is never used in the plural form.

otherwise

• the candidate is an object type

Put another way, a candidate noun concept is definitely an object type if:

• it is a placeholder in a copular adjective unary fact type,

• it is used with any determiners or numeric modifiers,

• it is used often in the plural form, or

• it has a high abstraction score.

otherwise

• it must have been tagged as a common noun more often than as a proper noun to be considered an object type.

The abstraction score for a candidate noun concept is the ratio of the total frequency of extractions of the candidate concept (when it is the head word) to the average number of actual occurrences of final candidates. For example, in
Each local area is located in a general area.

Assuming that ‘local area’ (Ⅲ) and ‘general area’ (Ⅴ) are selected (giving an average actual occurrence of 1), then the abstraction score for ‘area’, which has two annotations, would be 2. The abstraction score is used for two purposes, the first is as above: to help determine if a candidate noun concept is an object type or not. In this case, the idea is that is that if the term occurs frequently as the headword of many larger terms, then it is unlikely to be an individual concept. The second purpose is to present possible important abstractions (hence the name) to the user. Similarly, the intuition is that a term that is frequently used as the headword for larger terms is possibly a relevant term in its own right—even if never explicitly used by itself.

5.6.2.8 Final Filtering

In the last step, additional filtering is performed. This is currently limited to removing any candidate noun concepts that are no longer referenced by any fact types and are not the subject of any verbs in the dependency tree.

In contrast to other approaches, we do not filter based on a minimum frequency. We chose not to do so as many of the example texts from which terms are extracted are very short and, therefore, relevant terms may only be mentioned once.

5.6.2.9 Presentation to the User

Finally, the extracted vocabulary is presented to the user for review. To help the user in the review, and for traceability, the tool allows the user to view where each term was identified in the document. This will help the user to determine whether not an extracted term is really relevant. Furthermore, it may help determine where changes to the document may be required if they decide to modify the extracted vocabulary.
5.7 Summary

To realise the general knowledge-based framework for CNLs described in Chapter 4, this chapter presented CLUE4SBVR—a prototype implementation of the approach using SBVR models for the semantic representation. We began by introducing SBVR and justifying its use as it provides an expressive semantics through its logic-based meta-model and a flexible CNL through SBVR SE, fulfilling many of the criteria laid out in Section 4.1. Moreover, the SBVR meta-model incorporates the vocabulary and language aspects with a simple structure that maps cleanly to the lexicon structure of our framework.

Next we introduced the prototype type tool in the context of its use in an organisation for supporting the formalisation of natural language business or software specifications by domain experts. The components of CLUE4SBVR that are associated with the general framework were then described including the structure of the lexicon, the implementation of the syntactic analysis, and the mapping to the SBVR meta-model for semantic analysis. This application extends the description of the general framework by making concrete the aspects of the general framework that were left under-specified in the previous chapter. For example, the specification of a pruning strategy, in the syntactic analysis.

Finally, we stressed the need to quickly, easily, and precisely acquire the knowledge that the framework needs to operate. To fulfil this need we described two additional components for the acquisition of the lexicon and vocabulary for use by CLUE4SBVR. The first acquires the vocabulary from a simply defined glossary, using as little as term and definition, with the caveat—although beneficial in the context of CNLs—that verbs be specified in the SBVR fact type form, which indicates the associated noun terms alongside the verb. The second focuses on quickly acquiring vocabulary from existing natural language texts using IE techniques. To do this we defined rules synthesised from prior work that used IE to extract formal models from text, extended them with compound noun disambiguation, and focused the results to provide suitable SBVR vocabularies.

In the next chapter we will evaluate the different components of CLUE4SBVR.
Chapter 6

Evaluation of the Application

In this chapter we perform an evaluation of our knowledge-based framework for CNL Understanding; specifically, we evaluate CLUE4SBVR—the implementation of the framework for software and business specifications through SBVR described in the previous chapter. The general evaluation strategy is based on that described by Hirschman and Thompson [HT97] (and is the same as that used by similar work, e.g. [HG03; EVR11]), in which three forms of evaluation are discussed.

1. Adequacy Evaluation:
   This determines whether or not (or to what degree) a system is fit for a particular task or purpose. Adequacy evaluation is usually performed from the perspective of the user and, therefore, relates to user needs.

2. Diagnostic Evaluation:
   This is usually performed by developers and typically involves the creation of a comprehensive test suite. It is intended to test the system across a range of possible inputs.

3. Performance Evaluation:
   This type of evaluation measures system performance with respect to some aspect of interest (e.g. speed, accuracy, or error rate). Performance evaluation is typically quantitative and can be part of an adequacy evaluation.
Here we focus on performance evaluation for several reasons. First of all, we are interested in evaluating our approach in a general sense rather than for a specific task (adequacy evaluation) or for correctness (diagnostic evaluation). Second, a quantitative performance analysis allows our work to be compared to similar work that have also performed similar evaluations. Finally, time constraints and the early prototype status of our implementation meant it was not possible to perform a user study, which is what would be desired for an appropriate adequacy evaluation.

A performance evaluation consists of defining three aspects [HT97]: (1) the criterion, (2) the measure, and (3) the method. The criterion defines the specific aspect of performance being evaluated, while the measure defines what property of the system will be reported in order to determine to what degree the criterion is achieved. For example, if speed were the criterion of an evaluation, then the seconds to perform the operation might be reported. Finally, the method describes how a given measure is determined for a specific system; for example, performing an analysis of data created from execution on a benchmark task.

The evaluation described in this chapter is broken down to evaluate the individual components in an intrinsic manner: i.e., how the component performs individually rather than its contribution to the overall system. Each part of the evaluation of this work is defined in terms of the three aspects: criterion, measure, and method. In addition, the results are presented for each evaluation along with a brief discussion of their meaning. The components evaluated include: vocabulary acquisition from a glossary, vocabulary acquisition from unrestricted text, rules parsing. The evaluation of acquisition from unrestricted text includes a comparative evaluation of other approaches. Finally, a simple execution time evaluation is performed across components.

### 6.1 SBVR Lexicon Acquisition from Glossary

We start by evaluating the acquisition of an SBVR lexicon (or vocabulary) from a document in the form of a glossary. In the following we identify the criterion used for evaluation, the measure of system performance used to judge the criterion, the method, and finally present the results of the evaluation.
Section 6.1. SBVR Lexicon Acquisition from Glossary

6.1.1 Criterion

To evaluate the acquisition of an SBVR lexicon from a glossary, we look at how close the lexicon produced by the system (the system response) is compared to a lexicon produced by a human analyst (the answer key). While the lexicon produced by a human analyst is taken as an gold standard, it is commonly understood that a single gold standard does not necessarily exist when modelling. Since defining a (SBVR) vocabulary is an abstract form of modelling, the same holds here. For example, two different analysts in a domain may identify different concepts as important, use slightly different terminology, or define different structure of relations between concepts. However, in this case the acquisition of the model is based purely on the contents of the glossary; therefore, we can directly compare the produced model to that of the expected model represented by the glossary. This simplifies the comparison somewhat.

For the purposes of this experiment we have considered the SBVR EU-Rent example ([OMG08b, Annex E]) to be an appropriate gold standard since it is intended as a good example of SBVR SE and has been developed by a group of people. The answer key has been created from this glossary.

6.1.2 Measure

For the evaluation of how close the system produced lexicon is to the answer key we utilise three metrics: precision, recall, and F-measure (or F-score). While these metrics were initially used in Information Retrieval [Rij75], they have been applied to Information Extraction [GS96] and the creation of models from text (a form of information extraction, e.g. [HG03, SA14]) for quite some time. The metrics are defined as follows:

Definition 6.1 (Precision). Precision provides an indication of the accuracy of the results as a ratio of the number of correct responses produced by the system to the total number of responses it produced. That is, precision \( P \) is calculated as:

\[
P = \frac{N_{correct}}{N_{correct} + N_{incorrect}},
\]
where, $N_{\text{correct}}$ is the number of correct system responses and $N_{\text{incorrect}}$ is the number of incorrect responses.

**Definition 6.2 (Recall).** Recall measures the completeness of the system’s response by comparing the number of correct responses to those expected by the answer key. Recall ($R$) is calculated as follows:

$$ R = \frac{N_{\text{correct}}}{N_{\text{gold}}} $$

where, $N_{\text{correct}}$ is the same as for precision (above) and $N_{\text{gold}} \equiv N_{\text{correct}} + N_{\text{missing}}$ is the number of expected responses in the answer key.

**Definition 6.3 (F-measure).** Finally, F-measure gives an indication of the overall performance of the system, taking both precision and recall into account. In general, F-measure is the weighted harmonic mean of precision and recall; however, we give equal weight to both and, hence, calculate $F_1$ as:

$$ F_1 = 2 \times \frac{N_{\text{correct}}}{N_{\text{correct}} + N_{\text{incorrect}} + N_{\text{gold}}} $$

where $N_{\text{correct}}$, $N_{\text{incorrect}}$, and $N_{\text{gold}}$ are the same as defined for precision and recall (above).

### 6.1.3 Method

The metrics are calculated by comparing the response produced by the system to the answer key. In this case, the system response is represented by a set of annotations on the glossary document, which can be compared to the manually produced annotations constituting the answer key. This approach was chosen, rather than comparing the automatically generated lexicon (or the underlying SBVR model) itself, as it enabled the use of the evaluation framework built into the GATE tool.

The following describes three important aspects of the method: the selection of the test data, the creation of the answer key, and the test procedure itself.

**Test Data** The test data used by this evaluation consists of the EU-Rent vocabulary defined in the SBVR specification [OMG08b, Annex E]. Specifically, the sections E.2.2.1.1 through
Section 6.1. SBVR Lexicon Acquisition from Glossary

**Figure 6.1:** Example SBVR Glossary Test Data

E.2.2.1.11 were used. Moreover, the vocabulary was converted to plain-text format from the PDF, eliminating the SBVR SE font styling while maintaining the glossary format. A sample of the test data is displayed in Figure 6.1.

**Answer Key** The annotations representing the answer key were manually created by a single person based on the SBVR SE EU-Rent case study. The type of annotation is based on the type of the vocabulary entry: object type, individual concept, or fact type. In addition, the annotations are given a number of attributes representing information of the lexicon/vocabulary entry. The attributes include:

- **designation**[1..n]: The designation attribute(s) represent the terms used to refer to the entry. Every entry has at least one designation, regardless of its type, considered to be its primary designation (i.e., designation1). If the entry is for a fact type then the designation(s) consist of only the verb part with asterisks (*) in place of its placeholders.

- **fact type form**[1..n]: The fact type form attribute(s) represent the full form of a fact type including its verb part and its placeholders. The fact type forms are aligned with the designations: that is, fact type formi is associated with designationi (for i ∈ {1, ..., n}).

- **placeholder**[1..n][1..m]: The placeholder attribute(s) denote the placeholders of a fact type vocabulary entry. The first subscript indicates its association with a fact type form, and the second subscript indicates the position of the placeholder in that fact type form.
**Chapter 6. Evaluation of the Application**

**renter**

- **Concept Type:** situational role  
- **Definition:** 
  driver contractually responsible for a rental  
- **Synonym:** customer  
- **Synonym:** primary driver

**renter is responsible for rental**

- **Concept Type:** associative fact type  
- **Synonymous Form:** rental has renter

**Figure 6.2:** Example SBVR vocabulary and associated key annotations.

**concept type:** The concept type attribute lists any more specific concept types of the vocabulary entry, e.g. situational role for an object type. This is an optional attribute for annotations of any type; not all vocabulary entries have a more specific concept type.

**general concept:** The general concept attribute provides the primary designation of the more general concept that the vocabulary entry represented by the annotation specialises. This attribute is only applicable to object types and individual concepts.

An example of vocabulary entries and their annotations are displayed in Figure 6.2.

**Test Procedure** The basic procedure consists of, for a given document or corpus, running the process described in Section 5.6.1 up to (and including) the vocabulary acquisition part, which outputs annotations to the glossary representing the created lexicon using the approach as that for the answer key annotations. The comparison of the system generated annotations to the answer key annotations is performed automatically by the evaluation framework of GATE, which makes the following determinations when comparing annotations [CMBT+15]:
Correct annotations: An annotation, of the response, is considered correct if it is coextensive (i.e. has that same start and end offsets in the text) with an annotation of the same type in the answer key and it has all of the attribute-value pairs as the key annotation. Note: this allows the response annotation to have additional attributes that the gold annotation does not.

**Partially correct annotations:** A response annotation is considered partially correct if it overlaps (i.e. the two annotations share a span of text) a key annotation of the same type and it has all of the attribute-value pairs of the gold annotation.

**Missing annotations:** A key annotation is considered missing if (a) it is neither coextensive with nor overlapping a response annotation of the same type; or (b) one or more attribute-value pairs are not included in the response annotation.

**Incorrect (false positive) annotations:** A response annotation is considered incorrect if (a) it is neither coextensive with nor overlapping a gold annotation of the same type; or (b) one or more attribute-value pairs are not included in the gold annotation.

Finally, the evaluation is performed on the first six sections individually and then as a combined document. This provides some indication of the performance on business specifications of varying sizes and level of ambiguity. For example, some of the sections are quite self contained, while others are highly interdependent.

### 6.1.4 Results

The results of the vocabulary learning are summarised in [Table 6.1](#) and show the number of gold standard annotations, precision, recall, and $F_1$-score of each of the major categories (object type, individual concept, and fact type) as well as the totals across the categories. The totals are calculated by what [GATE](#) refers to as a *micro summary*, in which the statistics of correct, incorrect, and missing are taken across the entire document (or corpus) to calculate the metrics, rather than averaging the results of the individual categories. This provides more accurate results than averaging the result of the different categories.

1Note that, for this evaluation, partially correct annotations cannot occur, hence they are not considered in the calculation of the metrics.
### Table 6.1: Results of Vocabulary Acquisition from EU-Rent vocabulary

<table>
<thead>
<tr>
<th>Category</th>
<th>Section 2.2.1.X</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>1–6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object types</strong></td>
<td></td>
<td>14</td>
<td>10</td>
<td>20</td>
<td>11</td>
<td>20</td>
<td>19</td>
<td>94</td>
</tr>
<tr>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.95</td>
<td>0.86</td>
<td>0.99</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>(F_1)</td>
<td></td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.97</td>
<td>0.93</td>
<td>0.99</td>
</tr>
</tbody>
</table>

| **Individual concepts** |                 | 2  | 0  | 17 | 14 | 0  | 4  | 37  |
| #                     |                 |    |    |    |    |    |    |     |
| P                     |                 | 1.00 | —   | 1.00 | 1.00 | —   | 1.00 |     |
| R                     |                 | 1.00 | —   | 1.00 | 1.00 | —   | 1.00 |     |
| \(F_1\)               |                 | 1.00 | —   | 1.00 | 1.00 | —   | 1.00 | 1.00 |

| **Fact types**        |                 | 8  | 8  | 25 | 7  | 14 | 7  | 69  |
| #                     |                 |    |    |    |    |    |    |     |
| P                     |                 | 1.00 | 0.75 | 1.00 | 0.86 | 0.86 | 1.00 | 0.97 |
| R                     |                 | 1.00 | 0.75 | 1.00 | 0.86 | 0.80 | 0.57 | 0.96 |
| \(F_1\)               |                 | 1.00 | 0.75 | 1.00 | 0.86 | 0.83 | 0.73 | 0.96 |

| **All**               |                 | 24 | 18 | 62 | 32 | 34 | 30 | 200 |
| #                     |                 |    |    |    |    |    |    |     |
| P                     |                 | 1.00 | 0.89 | 1.00 | 0.97 | 0.91 | 0.90 | 0.98 |
| R                     |                 | 1.00 | 0.89 | 1.00 | 0.97 | 0.91 | 0.90 | 0.98 |
| \(F_1\)               |                 | 1.00 | 0.89 | 1.00 | 0.97 | 0.91 | 0.90 | 0.98 |

For the combined category, the precision and recall are always equal as no additional annotations are produced over than the number expected. Therefore, the differences between precision and recall only occur within a category, since an annotation may have ‘switched’ categories compared to what was expected.

#### 6.1.5 Discussion

There are exactly 200 terms in the vocabulary of the sections processed; almost half of which are object types, approximately one third are fact types, with the remainder individual concepts. The overall accuracy \((F_1\text{-score})\) of 98%, with the lowest scoring category (fact types) at 96%, is excellent considering verbs (which roughly correspond to fact types) are typically more difficult to learn than nouns (i.e. object types and individual concepts). The high accuracy of fact types is primarily due to the explicit method of representing them in [SBVR](#) by including the terms they relate, rather than the verb alone. This highlights the benefits of taking such an explicit approach to glossary definitions.
Although the overall accuracy is high, the individual sections exhibit varying performance—with lowest %89 for section 5. However, this is caused by the individual sections making reference to vocabulary terms defined in other sections. For example, a fact type referencing an object type as one of its roles, but the object type is defined elsewhere. As a result, the overall performance increases once the sections are combined and all of the related terms are present.

Overall, there are only three incorrectly learnt vocabulary entries, which are all due to missing vocabulary. Although this missing vocabulary is defined in the currently unprocessed sections, it is expected that these errors and the missing vocabulary could be identified by the analysis of definitions and rules. This would result in newly suggested vocabulary entries during the complete parsing process, as discussed in Section 5.5.5. Therefore, if some vocabulary is completely left out of the glossary, it does not result in a complete failure of the parsing; the two aspects work together to improve the completeness and correctness of the specification.

### 6.2 SBVR Lexicon Acquisition from “Unrestricted” Text

In the situation where a glossary is not provided it is important to enable a glossary to be produced quickly and effectively to support the parsing process. This is the purpose of the component for lexicon acquisition from unrestricted text, which we evaluate in this section using the same methodology by describing the criterion, measure, method, and results. Additionally, we perform a comparison between our approach and others that create conceptual models from unrestricted text, which is effectively what this component is doing. To perform a more meaningful comparison of approaches, we define a clearer criterion, measure, and method than used by other work and re-evaluate the results of approaches using this method.

#### 6.2.1 Preliminaries

The analysis of natural language specifications to identify concepts, attributes, and relations is essentially an Information Extraction (IE) task. Therefore, our framework uses IE evaluation techniques as a starting point and predominantly revolves around the comparison of a model
produced by the system (i.e. the system response) to a human generated model (i.e. the answer key). However, there some differences between typical IE tasks and that of creating conceptual models from specifications.

Typical IE tasks aim to extract specific information. For example, using IE competitions/conferences such as the Message Understanding Conference (MUC) [Chi97] and Text Analysis Conference (TAC) [NIS15] as relevant basis, the following are typical IE tasks:

- Named Entity recognition looks at identifying known types of entities, chiefly people, organisations, dates, times, locations, percentages, and currency [Chi97];
- Template Relation identifies instances of specific relations, e.g. employee_of, manufacturer_of, location_of [Chi97];
- Scenario Template identifies instances of specified events and their participants, e.g. launch events [Chi97];
- Slot Filling attempts to identify values for predefined slots/attributes for given entities [NIS15]; and
- (Cold Start) Knowledge-base Population uses a given schema and attempts to populate it with instances of the relevant entities, relations, and attribute values [NIS15].

In contrast, the creation of conceptual models requires the extraction of higher-level concepts and relationships: i.e. the schema that could then be used for the standard IE tasks.

In addition, the desired results of the standard tasks are more easily identifiable, at least to human analysis. For example, the human annotators for the Named Entity recognition task (arguably the most straightforward task) of MUC-7 scored very high agreement of approx. 97% [Chi97]. The most difficult task, the Scenario Template task, had annotator variability scores ranging between approx. 85%–96% [Chi97] depending on the document. In contrast, human agreement on conceptual models seems much more difficult to obtain. In [EVR11], the results of 9 software engineers, each with 5–20 years experience, for creating a conceptual model from a common case study showed accuracies ranging from approx. 47%–90% [Chi97]. Similarly, the results of [SA14], which evaluated the class models of 37 final year

---

2This is the $F_1$ score, which we will introduce when we discuss the metrics used to measure the results.
students, showed a wide variety of responses ranging between approx. 10%–90% recall and approx. 35%–100% precision—$F_1$ was not provided.

The varied responses in conceptual modelling is due to two key factors: (1) human judgement of relevance to both the domain and the system under development, and (2) integration of an individual’s prior domain- and/or general-knowledge. In the MUC tasks, relevance is explicitly ignored [Chi97, overview slides p. 34], while prior knowledge is considered, for example, in determining people or organisation type named entities. However, since the domain is newswire text, they are typically mentions of well-known people and organisations and has less of an impact then, for example, identifying a relationship between two concepts based on prior knowledge that is not mentioned in the text.

These differences, in particular the relevance judgement, makes evaluating conceptual models generated from natural language specifications more difficult and subjective. Indeed, the literature that does attempt an evaluation of the generated conceptual model against a key often argues that there is no commonly accepted correct model for a given problem; however, models can be judged as good or bad [HG03; EVR11]. Therefore, rather than an automated comparison like that used by the IE competitions, a manual comparison must be performed between system response and the key. The issue then is to limit the subjectivity to enable better comparisons between systems. A major feature of the evaluation framework presented here is to provide means to limit the subjectivity of the comparison or identify where subjectivity has an impact.

### 6.2.2 Criterion

The typical criterion used by evaluated approaches to creating conceptual models, and IE tasks in general, is the accuracy of the conceptual model produced by the system (the system response) when compared to a model produced by a human analyst (the answer key), or how close the system response is to the answer key. Since it is understood that there is no single correct conceptual model of a system, providing an appropriate answer key is problematic. Instead, a model can be considered good or bad [HG03]; however, this does not solve the

---

3Often with conceptual models from textbooks considered as good examples; this assumption will be discussed later.
problem as there is no single good model either. Moreover, a judgement of goodness adds further subjectivity to evaluation as, for example, a model could be considered good in one context and different in another.

On the other hand, providing something that can be compared against is important for supporting the comparison of approaches. If the criterion were simply “how good is the system response as judged by a human expert,” then it would be difficult to perform a meaningful comparison between approaches judged by different experts—given the variation in expert developed models reported by Elbendak et al. [EVR11], which are likely all good models given the experience (5–20 years worth) of the experts—or even the same expert over time. Therefore, we adopt the following criterion for our evaluation framework.

**Definition 6.4** (Evaluation Criterion). How accurate is the conceptual model produced by the system (the system response, SR) compared to a conceptual model produced by a human analyst (the answer key, KEY), where KEY is judged (by another analyst) to be a good, representative example model with respect to an identifiable context.

This criterion differs from that more commonly used by constraining the answer key to require agreement between human experts. Furthermore, the reference to a context introduces a small aspect of adequacy evaluation; however, it is intended to be interpreted quite broadly and not strictly as a user need. The idea is to make it easier to compare like with like; for example, some systems may be more suited to a specific context and, therefore, evaluating it against an inappropriate answer key does not reflect the system’s true performance. The exact meaning of representative and context will be discussed in Section 6.2.4.

### 6.2.3 Measure

To define the measure used to determine the accuracy of a system, we make utilise common metrics for comparing system responses to answer keys, primarily: precision, recall, and F-measure. While these metrics began in Information Retrieval [Rij75] they have been adopted by many natural language processing tasks [JM08], including Information Extraction [GS96]. Basically, precision indicates the accuracy of the system response as a ratio of the number of correct responses to the total number of responses produced by the system; recall provides an
indication of accuracy in terms of the completeness of the system responses compared to the expected responses in the answer key; and \( F\)-measure indicates the overall accuracy of the system as the (weighted) harmonic mean of precision and recall.

While these are common metrics and seem straightforward, there are several variations that may lead to confusion if not specified clearly. For example, a simple definition of precision \( (P) \) and recall \( (R) \) is shown in (6.1).

\[
P = \frac{N_{\text{correct}}}{N_{\text{correct}} + N_{\text{incorrect}}}, \quad R = \frac{N_{\text{correct}}}{N_{\text{key}}}
\]

where \( N_{\text{correct}} \) is the number of correct system responses, \( N_{\text{incorrect}} \) is the number of incorrect system responses, and \( N_{\text{key}} \) is the number of elements in the answer key.

A similar simple definition is used by Harmain and Gaizauskas [HG03]; however, without clarification it is difficult to determine exactly what \textit{correct} and \textit{incorrect} mean. Consider an alternative formulation, for example, that calculates precision and recall using the following metrics, as in [Chi97]: \( N_{\text{correct}} \), the number of correct system responses; \( N_{\text{incorrect}} \), the number of incorrect system responses (present in both response and key, but different values); \( N_{\text{missing}} \), the number of elements present in the key, but not in the system response; and \( N_{\text{spurious}} \) the number of responses produced that are not present in the key. Precision and recall are then calculated as in (6.2).

\[
P = \frac{N_{\text{correct}}}{N_{\text{correct}} + N_{\text{incorrect}} + N_{\text{spurious}}}, \quad R = \frac{N_{\text{correct}}}{N_{\text{correct}} + N_{\text{incorrect}} + N_{\text{missing}}}
\]

In particular, the issue is the correspondence between \( N_{\text{incorrect}} \) in (6.1) with \( N_{\text{incorrect}} \) and \( N_{\text{spurious}} \) from (6.2), which could lead to misinterpretation of the results. Depending on how the conceptual models are compared there are two possible correspondences, shown in (6.3).
and (6.4).

\[ N_{\text{incorrect}} \equiv N_{\text{incorrect}} + N_{\text{spurious}} \quad (6.3) \]
\[ N_{\text{incorrect}} \equiv N_{\text{spurious}} \quad (6.4) \]

The correspondence of (6.3) would mean that both situations where an element is in the response and the key, but they are not equal, and where an element is in the response, but not in the key, are considered as incorrect. In which case, \( N_{\text{key}} = N_{\text{correct}} + N_{\text{incorrect}} + N_{\text{missing}} \) would be incorrect (as \( N_{\text{spurious}} \) would also be counted), but may be assumed by a reader if clarification is not made. Whereas, the correspondence of (6.4) could indicate that the incorrect elements are being considered correct even if they are only partially correct.

In addition, the incorporation of over-specification, introduced by Harmain and Gaizauskas [HG03], may lead to more issues. Over-specification is intended to differentiate between a the language processing capability of a system and its abstraction capability. Moreover, it helps to overcome the issue of no single model being the sole correct solution. It does so by measuring the ratio of how much information not in the key is considered correct \( (N_{\text{extra}}) \) to that included in the key. However, the definition is unclear as it overlaps with that of \( N_{\text{incorrect}} \) (and/or \( N_{\text{spurious}} \), depending on the approach). Only by inspecting the values reported in the results table of [HG03] does it become clear that the intention is to exclude the additional elements considered correct from \( N_{\text{incorrect}} \). Moreover, not all evaluations that utilise over-specification take the judgement of correctness into account. For example, in [LZC13] \( N_{\text{extra}} \equiv N_{\text{spurious}} \), which typically leads to higher values than if the correctness is taken into account.

To clarify the relationships between the metrics, we define some fundamentals and the metrics in the following. The definitions are purposely left underspecified in some respects, as they will be further defined by the method (Section 6.2.4). The general idea is to categorise the elements of the answer and system response into disjoint sets and then calculate the measure based on the classification. An overview of the categories is shown in Figure 6.3.

**Definition 6.5.** Let \( \text{SR} \) the set of all model elements (concepts, relations, attributes, etc.) in the system response and, similarly, \( \text{KEY} \) the set of model elements in the answer key.
Section 6.2. SBVR Lexicon Acquisition from “Unrestricted” Text

Now $\text{SR}_{\text{correct}}$ is the set of model elements of the system response that are considered correct ($\text{correct}(x)$) and $\text{SR}_{\text{incorrect}}$ consists of those considered incorrect such that:

$$\text{SR}_{\text{correct}} = \{ e \in \text{SR} | \text{correct}(e) \}$$

$$\text{SR}_{\text{incorrect}} = \{ e \in \text{SR} | \neg \text{correct}(e) \}$$

In addition, an element is considered correct if it is judged by a human analyst to be correct (how this is decided will be discussed in the method) or it is in the answer key, i.e.:

$$\forall e \exists k : e \in \text{SR} \land k \in \text{KEY} \land e \equiv k \implies \text{correct}(e)$$

Moreover, $\text{SR}_{\text{correct}}^{(\text{key})}$ and $\text{SR}_{\text{correct}}^{(\text{extra})}$ are subsets of $\text{SR}_{\text{correct}}$, such that:

$$\text{SR}_{\text{correct}}^{(\text{key})} \cup \text{SR}_{\text{correct}}^{(\text{extra})} = \text{SR}$$

$$\text{SR}_{\text{correct}}^{(\text{key})} \cap \text{SR}_{\text{correct}}^{(\text{extra})} = \emptyset$$

$$\text{SR}_{\text{correct}}^{(\text{key})} = \{ e \in \text{SR}_{\text{correct}} | \exists k \in \text{KEY} : e \equiv k \}$$

$$\text{SR}_{\text{correct}}^{(\text{extra})} = \{ e \in \text{SR}_{\text{correct}} | \forall k \in \text{KEY} : e \not\equiv k \}$$

where the equivalence between an element of the response and the key will be determined by the method.
Similarly, $\text{SR}_{\text{incorrect}}$ is partitioned into two subsets:

\[
\text{SR}_{\text{incorrect}}^{{\neq}} = \{ e \in \text{SR}_{\text{incorrect}} \mid \exists k \in \text{KEY} : \text{match}_{\text{par}}(e, k) \} \\
\text{SR}_{\text{incorrect}}^{\text{spurious}} = \{ e \in \text{SR}_{\text{incorrect}} \mid \forall k \in \text{KEY} : \neg \text{match}_{\text{par}}(e, k) \}
\] (6.12) (6.13)

The $\text{match}_{\text{par}}$ relation represents some kind of partial match (defined later) between an element of the system response and that of the key. Therefore, the $\text{SR}_{\text{incorrect}}^{{\neq}}$ set represents the elements that are partially correct and could be further subdivided into those elements that should be given credit (i.e. considered more correct than incorrect) and those that should not. However, we believe the framework is fine grained enough (and less complicated) without giving credit to partially correct results. The set $\text{SR}_{\text{incorrect}}^{\text{spurious}}$ contains the elements of the system response that do not partially match and are not equivalent to an element of the key at all.

We place a constraint on the matching of elements in that only a single element of the key can be considered equivalent to, or partially matching, an element of the system response.

\[
\forall e_1, e_2, k : e_1 \in \text{SR} \land e_2 \in \text{SR} \land k \in \text{KEY} \land \\
(e_1 \equiv k \lor \text{match}_{\text{par}}(e_1, k)) \land \\
(e_2 \equiv k \lor \text{match}_{\text{par}}(e_2, k)) \implies e_1 = e_2
\] (6.14)

This means that, in following the method, each element of the system response must be deterministically placed in only one of the subsets, thereby maintaining their disjointedness.

Finally, the set of missing elements are those present in the key, but not in the response, i.e. :

\[
\text{KEY}_{\text{missing}} = \{ k \in \text{KEY} \mid \forall e \in \text{SR} : k \not\equiv e \}
\] (6.15)

Now precision, recall, F-measure, and over-specification can be defined more clearly.

**Definition 6.6 (Precision & Key Precision).** Precision is the ratio of the number of correct responses to the number of responses produced, i.e. :

\[
P = \frac{N_{\text{cor}}}{N_{\text{prod}}}
\]
Section 6.2. SBVR Lexicon Acquisition from “Unrestricted” Text

where \( N_{cor} = |\text{SR}_{correct}| \) and \( N_{prod} = |\text{SR}| \). This is general precision; however, there is an alternative that could be considered (and is what is used in other work, e.g. [HG03]) where the precision is considered in terms of the answer key, ignoring the extra terms. We call this key precision.

\[
P_{key} = \frac{N_{key}^{correct}}{N_{cor} + N_{inc}}
\]

where \( N_{key}^{correct} = |\text{SR}_{correct}^{key}| \) and \( N_{inc} = |\text{SR}_{incorrect}^{key}| \). Key precision is important based on the context and for comparing systems to previously published results that calculated it in this way. Note: it follows that \( N_{prod} = N_{cor} + N_{inc} \).

**Definition 6.7 (Recall).** Recall is defined as normal, the number of correct responses found in the answer key compared to the number of expected elements of the key.

\[
R = \frac{N_{key}^{correct}}{N_{key}}
\]

where \( N_{key}^{correct} = |\text{SR}_{correct}^{key}| \), as above, and \( N_{key} = |\text{KEY}| \).

**Definition 6.8 (F-measure).** F-measure provides a measure of the overall accuracy of the system as it is the (weighted) harmonic mean of precision and recall. However, we give equal weight to precision and recall and, hence, define \( F_1 \) as:

\[
F_1 = \frac{2(P_x)(R)}{P_x + R}
\]

where \( P_x \) stands for one of the precision metrics.

**Definition 6.9 (Over-specification).** Over-specification is defined as in [HG03] to include the judgement of correctness.

\[
OV S = \frac{N_{extra}}{N_{key}}
\]

where \( N_{extra} = |\text{SR}_{correct}^{extra}| \) and \( N_{key} \) is defined as above.

### 6.2.4 Method

While the measure has specified how the metrics are calculated in terms of the subsets of elements from the system response and answer key, we have not defined how to classify
the elements as members of the those subsets. That is the purpose of the method. For this, we must define what it means for elements to be equivalent ($\equiv$) and to partially match ($\text{match}_{\text{par}}(x, y)$).

Before defining the equivalency and partial match relations, we should clarify what it is we are determining to be equivalent or matching. To provide a common model for comparison that is agnostic to any particular conceptual modelling language, we assume that all the entities of $\text{SR}$ and $\text{KEY}$ are one of two types\(^4\) concepts or relations. Concepts are represented by a tuple of synonymous names (e.g. $\langle \text{Loan Item}, \text{Item} \rangle$, $\langle \text{title} \rangle$), while relations are represented by a signature including the name of the relation and the types of their arguments (e.g. $\text{is-on-loan}(\langle \text{Loan Item}, \text{Item} \rangle)$, $\text{has}(\langle \text{Loan Item}, \text{Item} \rangle, \langle \text{title} \rangle)$). Relations. Relations can also have synonyms, including potentially the inverse relation name and, hence, argument order: e.g. $\langle \text{has}(\langle \text{Loan Item} \rangle, \langle \text{title} \rangle), \text{is-of}(\langle \text{title} \rangle, \langle \text{Loan Item} \rangle) \rangle$. For simplicity and brevity, we will often only refer to a concept or relation using its primary form.

This is a simple approach that allows relations to be considered explicitly, rather than indirectly as part of the concept evaluation. Moreover, the results of the evaluation should be more accurate rather than marking a concept incorrect, for example, if it has three relations and two are incorrect, which would effectively take the relations into account twice.

A simple equivalence relation between an element of the response set and an element of the key set would be an exact match. For example, considering only concepts, a concept from the response set would be equivalent to that of the key set if and only if they had the exact same name. While this approach would better support automation of the comparison, it is too strict to be useful in this evaluation framework for several reasons:

- Natural language processing is hard and far from a solved problem;
- The same model can be described as text in multiple ways, one or more of which may occur in any given text;

\(^4\)All the key elements of different conceptual modelling languages can arguably be mapped into these two types. Classes are clearly concepts, while instances (e.g. members of an enumeration) can be considered singleton concepts. Attributes, associations, and methods can all be considered relations—in fact, it could be argued that the difference is a design choice, or an implementation/modelling language detail and should not even appear at the highest-level conceptual models.

\(^5\)These examples are taken from the Library case study \cite{Cal94}.

\(^6\)This could also be considered the binary relation $\text{isOnLoan}(\langle \text{Loan Item}, \text{Item} \rangle, \langle \text{Boolean} \rangle)$.
• Different approaches and human analysts may use different word forms, e.g., the root form of a word may or may not be used, the model may utilise a specific naming convention, a relation may be extracted from the text in passive form but the key is created in active form because the modeller preferred it, etc.;

• A human analyst may model things slightly differently than is described in the text, e.g., through adding (implied) abstractions, removing words from a name considered irrelevant;

• The information being extracted is not being mapped into a predefined set of expected types and relations.

However, using a more approximate matching may lead to violations of the constraint that a key element may only be matched once. Therefore, we say that only the closest match counts, any other matches would result in element being classified as either $SR_{extra \ correct}$ or $SR_{spurious \ incorrect}$. One of the benefits of this approach is that it allows differentiation between approaches with different capabilities. For example, an approach that does not resolve synonyms would have higher over-specification (assuming the synonymous concepts are classified as $SR_{extra \ correct}$) than an approach that does resolve synonyms. With this in mind we define equivalence for concepts and relations in the following.

**Definition 6.10 (Concept Equivalence).** A concept $c_1 \in SR$ is equivalent to ($\equiv$) a concept $k \in KEY$ if it fulfils one of the following match criteria and is a better match than any other concept $c_2 \in SR$:

1. One or more of the names are approximately equal and there are no conflicting names. This takes into account different word forms, naming conventions, etc. For example, $\langle Loan\_Item, Item\rangle$, $\langle Loan\_Items\rangle$, $\langle Loan\_Item\rangle$, and $\langle Item\rangle$ are equivalent; $\langle Loan\_Item\rangle$ and $\langle Loan\_Item, Customer\rangle$ are not.

2. One or more names are considered synonyms (given the context of the conceptual model), but no names are approximately equal and there are no conflicting names, e.g. $\langle Customer\rangle$ and $\langle Member\rangle$ are equivalent.
where \textit{better match} is decided by the order of the criteria (earlier criteria are better) or, if the same criterion is fulfilled, the judgement of the person performing the comparison (e.g. \langle\text{Library Item}\rangle could be considered a better match to \langle\text{Loan Item}\rangle than \langle\text{Item}\rangle or \langle\text{Borrowed Item}\rangle).

\textbf{Definition 6.11} (Relation Equivalence). A relation \(r_1 \in SR\) is equivalent to a relation \(k \in KEY\) if it fulfils the following match criteria and is a \textit{better match} than any other concept \(c_2 \in SR\):

1. One or more of the name must be either:
   
   (a) approximately equal (e.g. \textit{is-on-loan}, \textit{is_on_loan}, and \textit{isLoaned} are approx. equal), or
   
   (b) considered synonyms, including inverse forms such as active vs. passive terms (e.g. \textit{is-made-up-of}, \textit{consists-of}, and \textit{is-part-of} are synonyms, where the last is the inverse of the prior two); and

2. No names are conflicting (e.g. \langle\text{borrow}\rangle and \langle\text{borrow, renew}\rangle are conflicting);

3. The argument types of the relations are equivalent, as defined in [Definition 6.10] or can be considered synonyms of an equivalent concept\footnote{This allowance is made to prevent double penalisation for not resolving synonyms, i.e., the synonymous but not equivalent concepts will already contribute to over-specification so it would be unfair to indicate that the relation is incorrect, even though it was correctly extracted between the terms used in the text.} and

4. No arguments have conflicting order (e.g. \langle\textit{has}(\langle\text{Loan Item}\rangle, \langle\text{title}\rangle)\rangle and \langle\textit{is-of}(\langle\text{Loan Item}\rangle, \langle\text{title}\rangle)\rangle have conflicting argument order, where \textit{‘is-of”} is considered the inverse synonym of ‘\textit{has’}).

where approximately equal names are considered a better match than synonymous names or, if the same criterion is fulfilled, the judgement of the person performing the comparison (e.g. \textit{is_on_loan} would be considered a better match to \textit{is-on-loan} than \textit{is_loaned_out}).

These definitions support a process where concepts are aligned first, followed by relations. Moreover, they take relevance and implicit knowledge into account due to the \textit{best match} criterion. For example, assuming that the answer key includes the concepts and relations considered most relevant, if an approach is able to limit the produced elements to the most relevant
terms (e.g. by collapsing synonyms correctly) it will show better performance (through lower
over-specification and higher precision) than an approach that produces separate elements for
each synonymous term. Similarly, if implicit knowledge is included in the key (e.g. a relation
not explicitly mentioned in the text) and a approach can produce it, then that approach will
score better (wrt. recall) than an approach that cannot. Conversely, if an approach were to
infer a lot of implicit relations that are not in the key (and therefore not considered relevant),
it would score worse (wrt. over-specification and/or precision) than an approach that does not
create the extra relations.

Next, we define the partial match of concepts and relations similarly to equivalence.

**Definition 6.12** (Concept Partial Match). A pair of concepts \((c \in SR, k \in KEY)\) partially
match \((\text{match}_{\text{par}}(c, k))\) if one of the following criteria is fulfilled:

1. One or more names approximately match or are synonymous (see [Definition 6.10]), but
   there is a name that is conflicting. For example, \(\langle\text{Loan Item, Customer}\rangle\) partially matches
   \(\langle\text{Loan Item}\rangle\) and \(\langle\text{Customer, Member}\rangle\).\(^8\)

2. The name(s) are *truncated* [FB00], i.e., the name of \(c\) has been shortened compared to
   that of \(k\). For example, \(\langle\text{date}\rangle\) is truncated compared to \(\langle\text{date-of-birth}\rangle\).

3. The name(s) are *expanded* [FB00], i.e., the name of \(c\) has been lengthened compared to
   that of \(k\). For example \(\langle\text{number of subject sections}\rangle\) is expanded compared to \(\langle\text{subject
   section, section}\rangle\).

*Truncation* and *expansion* differ from approximate matching and synonymous terms in that
they result in a change of meaning, which could be small and so considered *partially correct*,
or it could result in a completely different meaning and be clearly *incorrect*.

**Definition 6.13** (Relation Partial Match). A par of relations \((r \in SR, k \in KEY)\) partially
match \((\text{match}(r, k))\) if one of the following criteria is fulfilled:

1. One or more names approximately match or are synonymous (see [Definition 6.11]) and
   the argument types are correct (i.e. equivalent and non-conflicting), but the there is a

\(^8\)Note that the multiple partial matches, if they exist, to not affect the measure as the underlying sets that the
metrics are based on do not allow duplicates.
conflicting name. For example, $\langle \text{borrows}((\text{Customer}), (\text{Loan Item})), \text{renews}((\text{Customer}), (\text{Loan Item})) \rangle$ partially matches $\langle \text{borrows}((\text{Customer}), (\text{Loan Item})), \text{renews}((\text{Customer}), (\text{Loan Item})) \rangle$.

2. The names approximately match or are synonyms, and there are no conflicting names, but there is some argument type that it not equivalent or the argument order is incorrect. In addition, the equivalent argument must not be a common/primitive concept such as boolean, string, integer. For example, $\text{has}(\langle \text{Loan Item} \rangle, \langle \text{title} \rangle)$ and $\text{is-of}(\langle \text{Loan Item} \rangle, \langle \text{title} \rangle)$ partially match, $\text{has}(\langle \text{Loan Item} \rangle, \langle \text{title} \rangle)$ and $\text{has}(\langle \text{Loan Item} \rangle, \langle \text{title language} \rangle)$ partially match, but $\text{is-on-loan}(\langle \text{Loan Item} \rangle, \langle \text{Boolean} \rangle)$ and $\text{is-on-loan}(\langle \text{Customer} \rangle, \langle \text{Boolean} \rangle)$ do not.

3. The name(s) are truncated (but not considered approximately equal nor synonymous) and the arguments are equivalent, or the name(s) approximately match or are synonyms but there are less arguments in $r$ than in $k$. For example, both $\text{made}(\langle \text{Library} \rangle, \langle \text{Section} \rangle)$ and $\text{made-up-of}(\langle \text{Library} \rangle)$ truncate $\text{made-up-of}(\langle \text{Library} \rangle, \langle \text{Section} \rangle)$. This purposely excludes relations that have different names and different number of arguments from partially matching, i.e., $\text{made}(\langle \text{Library} \rangle)$ does not partially match $\text{made-up-of}(\langle \text{Library} \rangle, \langle \text{Section} \rangle)$.

4. The name(s) are expanded (but not considered approximately equal nor synonymous) and the arguments are equivalent, or the name(s) approximately match or are synonyms but there are more arguments in $r$ than in $k$. For example, $\text{may-borrow-up-to-maximum-of-8}(\langle \text{Customer} \rangle, \langle \text{Loan Item} \rangle)$ expands $\text{borrow}(\langle \text{Customer} \rangle, \langle \text{Loan Item} \rangle)$, and $\text{is-renewed-to-extend}(\langle \text{Loan Item} \rangle, \langle \text{Current Loan} \rangle)$ expands $\text{is-renewed}(\langle \text{Loan Item} \rangle)$ (where $\text{is-renewed-to-extend}$ is considered approximately equal to, or synonymous with, $\text{is-renewed}$).

While truncation of relation names may change the meaning to a completely different relation, expansion often incorporates constraints into the name of the relation (but is otherwise the same underlying relation).
Finally, we define correctness to allow the complete assignment of elements into specific sets. Note that a concept or relation does not need to partially match to be considered correct (for those concepts and relations that are not equivalent).

**Definition 6.14** (Correctness). A concept or relation \( e \in \mathcal{SR} \) is considered correct if:

1. it is equivalent to an element of the answer key, i.e., \( \forall e, k : e \in E \land k \in \text{KEY} \land e \equiv k \implies \text{correct}(e, k) \); or
2. it is judged correct by a human analyst, taking into consideration the semantics of the concept/relation with respect to the context of the domain and the text the concept/relation was extracted from (including the location of the text from which it was identified, i.e., is it a reasonable extraction given the input).

**Classification of Extra, Errors, and Missing**

While the method has attempted to provide some clear definitions for the classification of elements of the system response into the core sets of elements with which the measure can be determined, there is still a large subjective element. We propose a qualitative classification of the elements of the extra (\( \mathcal{SR}_{\text{extra}}^{\text{correct}} \)), incorrect (\( \mathcal{SR}_{\text{incorrect}} \)), and missing (\( \text{KEY}_{\text{missing}} \)) sets. This finer grained classification attempts to identify specific reasons for a concept or relation to be judged as correct or incorrect, or why it was not extracted at all.

By identifying the category to which an extra, incorrect, or missing element belongs, it provides a greater insight into how close the system generated model is to that expected by a human analyst. Moreover, it requires that the person performing the evaluation justify the results within more restrictive framework, which helps to control the subjectivity of the analysis. The classifications provided here are a suggested, but not exhaustive, set of classifications.

**Categories of Extra Elements** The following are the most basic categories considered for elements that are judged *correct* but have no equivalents in the answer key.
synonym: classifies a concept or relation that has been judged correct based on it being a reasonable synonym of an element of the answer key. This is most likely to occur for systems that do not resolve synonyms into a single concept.

generalisation: (or hypernym) classifies a term, concept, or relation that is considered more general than a relevant term from the answer key.

specialisation: (or hyponym) classifies a term, concept, or relation that is considered more specific than a term from the answer key.

truncation: classifies an element that truncates (as defined for partial matches) a concept or relation from the answer key.

expansion: classifies an element that is expands (as defined for partial matches) a concept or relation from the answer key.

in text & valid: classifies an element that has been correctly extracted from the text, due to it being present, and it is considered a valid domain term.

There is some overlap between generalisation/specialisation and truncation/expansion; therefore, when both are applicable we prefer generalisation/specialisation to maintain a single classification per element. In this case, truncation and expansion are more applicable to relations than concepts as they can include a change in the number of arguments, not only the term used. Furthermore, generalisation and specialisation of relations can result from a more general/specific relationship (name) or have arguments that are more general/specific. For example, given the system response and answer key shown in Figure 6.4, the following classifications could be made (based on the context):

\[\text{⟨item⟩: synonym to ⟨Loan Item⟩}\]
\[\text{⟨tape⟩: generalises ⟨Language Tape⟩}\]
\[\text{⟨book bar-code⟩: specialises ⟨Bar-code⟩}\]
\[\text{⟨loan⟩: truncates ⟨Loan Item⟩}\]
\[\text{⟨unique member number⟩: expands ⟨Member Number⟩}\]
Section 6.2. SBVR Lexicon Acquisition from “Unrestricted” Text

\[
SR^\text{key}_{\text{cor.}} = \{ \langle \text{author} \rangle, \langle \text{bar-code} \rangle, \langle \text{book} \rangle, \langle \text{class mark} \rangle, \langle \text{current loan} \rangle, \langle \text{language tape} \rangle, \langle \text{library} \rangle, \\
\langle \text{loan item} \rangle, \langle \text{section} \rangle, \langle \text{title} \rangle, \text{has}(\langle \text{item} \rangle, \langle \text{bar-code} \rangle), \text{made-up-of}(\langle \text{library} \rangle, \langle \text{section} \rangle), \\
\text{has}(\langle \text{section} \rangle, \langle \text{class mark} \rangle) \}
\]

\[
SR^\text{extra}_{\text{cor.}} = \{ \langle \text{book bar-code} \rangle, \langle \text{item} \rangle, \langle \text{loan} \rangle, \langle \text{tape} \rangle, \langle \text{unique member number} \rangle, \\
\text{is-of}(\langle \text{bar-code} \rangle, \langle \text{item} \rangle), \text{has}(\langle \text{loan item} \rangle, \langle \text{author} \rangle), \text{has-part}(\langle \text{library} \rangle, \langle \text{section} \rangle), \\
\text{denotes}(\langle \text{class mark} \rangle, \langle \text{section} \rangle), \text{has}(\langle \text{book} \rangle, \langle \text{title} \rangle), \text{issues}(\langle \text{library} \rangle, \langle \text{loan item} \rangle), \\
\text{is-renewed-to-extend}(\langle \text{loan item} \rangle, \langle \text{current loan} \rangle) \}
\]

\[
\text{KEY} = \{ \langle \text{Loan Item} \rangle, \langle \text{Language Tape} \rangle, \langle \text{Book} \rangle, \langle \text{Bar-code} \rangle, \langle \text{Member Number} \rangle, \langle \text{Author} \rangle, \\
\langle \text{Library} \rangle, \langle \text{Section} \rangle, \langle \text{Class Mark} \rangle, \langle \text{Title} \rangle, \langle \text{Member} \rangle, \text{has}(\langle \text{Loan Item} \rangle, \langle \text{Bar-code} \rangle), \\
\text{has}(\langle \text{Book} \rangle, \langle \text{Author} \rangle), \text{made-up-of}(\langle \text{Library} \rangle, \langle \text{Section} \rangle), \text{has}(\langle \text{Section} \rangle, \langle \text{Class Mark} \rangle), \\
\text{has}(\langle \text{Loan Item} \rangle, \langle \text{Title} \rangle), \text{issues-to}(\langle \text{Library} \rangle, \langle \text{Loan Item} \rangle, \langle \text{Member} \rangle), \\
\text{is-renewed}(\langle \text{Loan Item} \rangle) \}
\]

\text{FIGURE 6.4: Example system response (by correct key and extra) and answer key.}

\text{is-of}(\langle \text{bar-code} \rangle, \langle \text{item} \rangle):
\text{synonym to has}(\langle \text{Loan Item} \rangle, \langle \text{Bar-code} \rangle)\text{\textsuperscript{9}}\]

\text{has}(\langle \text{loan item} \rangle, \langle \text{author} \rangle):
\text{generalises has}(\langle \text{Book} \rangle, \langle \text{Author} \rangle)\text{\textsuperscript{10}}\]

\text{has-part}(\langle \text{library} \rangle, \langle \text{section} \rangle):
\text{generalises made-up-of}(\langle \text{Library} \rangle, \langle \text{Section} \rangle)\]

\text{denotes}(\langle \text{class mark} \rangle, \langle \text{section} \rangle):
\text{specialises has}(\langle \text{Section} \rangle, \langle \text{Class Mark} \rangle)\]

\text{has}(\langle \text{book} \rangle, \langle \text{title} \rangle):
\text{specialises has}(\langle \text{Loan Item} \rangle, \langle \text{Title} \rangle)\]

\text{issues}(\langle \text{library} \rangle, \langle \text{loan item} \rangle):
\text{truncates issues-to}(\langle \text{Library} \rangle, \langle \text{Loan Item} \rangle, \langle \text{Member} \rangle)\]

\text{is-renewed-to-extend}(\langle \text{loan item} \rangle, \langle \text{current loan} \rangle):
\text{expands is-renewed}(\langle \text{Loan Item} \rangle)\]

\textsuperscript{9}For relations, an argument can refer to a concept that has been considered as a synonym of a concept in the key.

\textsuperscript{10}Assuming that in the context of the domain other types of \langle Loan Item \rangle can have an author, only in the key the modeller has determined that it is only really relevant for \langle Book \rangle.
Categories of Errors  Similarly, erroneous concepts and relations can be classified. In this case, the classifications follow, in large part, from the definitions for partial matches (remember that partial matches could be considered partially correct or completely incorrect, and no partial match means the term is spurious). The categories are as follows:

*synonym conflicts*: classify concepts or relations that include a correct name and a conflicting name, i.e., a name that is not reasonably considered a synonym of the other(s).

*argument conflicts*: classify relations that have incorrect arguments.

*parse errors*: classify elements that have been extracted as the result of an incorrect parse by an underlying parser/tagger.

*truncation*: as above, the difference being that the truncated term is judged incorrect.

*expansion*: as above, the difference being that the truncated term is judged incorrect.

*irrelevant*: terms are those that may be valid concepts are relations in general, but given the context of the domain and the text are completely irrelevant. These are often the result of filler words or multi-word expressions, e.g., ⟨order⟩ extracted from ‘in order to’.

*just plain wrong*: how did these terms get extracted in the first place? These errors indicate the possibility that bugs are present in the system or that the term does not fit in any other category.

Categories of Missing  Finally, we propose a classification of missing elements. This is important for distinguishing approaches with similar results for the metrics. Knowing why concepts and relations have been missed by the system allows the identification of situations in which a particular system may be particularly strong or weak.

*parse errors*: indicate concepts and relations that were unable to be extracted due to errors caused by an underlying parser/tagger, e.g. incorrect prepositional attachment.
Section 6.2. SBVR Lexicon Acquisition from “Unrestricted” Text

**implicit:** concepts and relations are those that are not explicitly present in the text, but can still be identified in the text by human with some analysis and domain or general knowledge. For example, ‘other details on a customer’ could give rise to the concept ⟨customer details⟩, ‘the library must support the facility for an item to be searched’ implies a relation searches(⟨library⟩, ⟨item⟩).

**not in the text:** identifies concepts and relations that are not explicitly nor implicitly present in the text, but require human reasoning and general/domain knowledge to elicit. This may include additional details that a modeller thought were relevant, e.g. has(⟨Book⟩, ⟨Subject⟩), or abstractions that are not identified in the text, e.g. an association class that is introduced solely by the modeller as best practice.

**incomplete rules:** indicate that a concept or relation was missed due to incompleteness of the system (either intentional or not), e.g. no extraction rules were implemented to identify a relationship from a specific syntactic pattern.

**filtered:** terms are those that are initially identified by the system, but were subsequently filtered from the final results.

**partial matches:** indicate terms that have not had equivalents extracted by the system, but a partial match has been extracted. For example, has(⟨Book⟩, ⟨Subject⟩) is in the key, but hasSubject(⟨Book⟩) is in the response. Note that this only applies if one of the previous categories does not.

**Test Data**

The text that is used to perform the configuration is an important aspect for providing an informative and useful performance evaluation that is suitable for the comparison of different approaches. When selecting or developing a text to serve as the basis for evaluation a number of factors must be considered to ensure that the text is appropriate, for example: the length and style of the text, language complexity, the source of the text, whether or not it is representative of the texts that intend to be processed.
Several approaches make use of texts from conceptual modelling (or software development, etc.) textbooks as they are provided with example model solutions (i.e. an answer key) enabling evaluation against them. However, these texts are typically very short, informal problem statements of well-known domains and not necessarily the most appropriate for several reasons:

1. the length is too short to provide meaningful results;
2. the informal nature of the texts is not necessarily representative of those encountered in the real-world;
3. the target audience, i.e. students, is not representative of the real-world target audience, leading to the inclusion of irrelevant text, e.g. instructions to the readers;
4. the situations described are simplified examples;
5. the intention of the textbook is different to that of real-world specifications: for example, the use of a well-known domain allows students to bring their common-knowledge to analyse an incomplete description (possibly one of the reasons why the variation in conceptual models produced by students is so high);
6. the example model solutions do not necessarily match as they may be the result of performing additional elicitation of requirements through the process described by the textbook.

Due to these possible issues, careful consideration must be made before utilising such a text for evaluation purposes. Moreover, it is important to be able to make meaningful generalisations and comparisons from the results of an evaluation. Therefore, the following characteristics of evaluation texts should be considered when performing an evaluation.

**Length**  How long is the text?

*very short:* One page or less, i.e., $\leq 500$ words.

*short:* Up to ten pages, 500–5,000 words.
average: Ten to 100 pages, 5,000–50,000 words.

long: 100 to 250 pages, 50,000–125,000 words.

enormous: You managed to find a billion word corpus in a single specification... Congratulations!

**Style** To what degree is the text formal or informal?

totes informal: The text is not just informal, it is ridiculously so, including: slang, txt speech, emoticons, etc. This type of text is more suited to social media than software or business specifications.

informal: The text has very little or no restrictions to style, grammar, and vocabulary and has no document structure (in terms of headings and sections). For example, the text may be closer to the way a person may talk, using contractions, first person, etc. However, it could also use a more formal style with the key difference being a lack of structure.

formal: The text is written such that it conforms to conventions regarding style, grammar, vocabulary, and structure: e.g. avoiding use of the first person, favouring the passive voice, no contractions, no slang, document sectioning, and specified headings.

very formal: The text adheres to restrictions on the language that can be used and the structure of the document, such as restrictions to grammar (e.g. simple subject-verb-object constructions), vocabulary (e.g. use of specific terms with defined meanings), and predefined document structure/template.

overly formal: The text is in a formal form such as a Controlled Natural Language with a formal grammar. This type of text can be parsed using automated means. Why are you using IE methods on it?
**Sentence Length**  How long (on average) are the sentences in the text?

An often reported metric is the average length of the sentences in the text. While it is a pretty weak measure for language complexity, it can serve as indicator as to how difficult a text may be to parse—usually longer sentences are problematic in NLP. Common guidelines (for English) recommend sentence lengths of 15–20 words, with more than 25 words often a considered a problem. Therefore, we define the following categories:

- **very short**: < 10 words
- **short**: 10–14 words
- **average**: 15–20 words
- **long**: 21–24 words
- **very long**: 25+

**Completeness**  Does the text contain the information to be modelled?

- **no**: Is this text even for the desired domain?
- **a little**: Most of the text is irrelevant, the important details have been left implicit or left out completely. Most of what needs to be modelled must be determined from other sources.
- **average**: About half of the concepts and relations that need to be modelled can be found in the text explicitly. Additional information may be implicit in the text, but quite a lot is missing.
- **mostly**: Most of the important and relevant information to be modelled is explicitly present in the text. Some information may be left implicit or excluded.
- **completely**: The text explicitly includes those things that need to be included in the model.

For the purpose of evaluating conceptual models produced by text analysis, the text used should fall into one of the last two categories. There is no point, for example, to analyse the
Section 6.2. *SBVR Lexicon Acquisition from “Unrestricted” Text*

ATM problem statement and compare the results to the answer key for the Library system. At the same time, texts in the *average* category could be appropriate for differentiating approaches that are able to extract more implicit information and apply more general knowledge reasoning to create the model.

**Answer Key**

Another important aspect is the creation of the answer key itself. For the evaluation criterion we stated that the answer key should be judged by another analyst, and be a good and representative example in a specific context. By *good* we mean that the answer key should be free from errors, conform to good conceptual modelling principles and practices. A *representative* model is one that includes many (if not all) of the most relevant concepts for the system or domain being modelled and conforms to a domain expert’s conceptualisation as much as practical. The *context* involves the purpose of the model, the text it is associated with, and how the model is created. These criteria are very subjective; therefore, it is important that there be *agreement* between experts on the adequacy of a conceptual model for use as an answer key.

These attributes are not necessarily guaranteed by conceptual models taken from textbooks, as is commonly done. For example, an example from a textbook written by a single author may be oversimplified (not representative), demonstrate modelling features that are not necessarily needed (not good), and has not been cross-checked by a second expert (no agreement). While the authors no doubt endeavour to produced good example models, the motivation (i.e. the context) behind the examples (e.g. demonstrating a particular modelling or analysis technique) are not necessarily the same as for the production of real-world conceptual models (e.g. capturing all of the relevant domain concepts and relations).

The description of an answer key should indicate which category it belongs to for each of the criteria. Keep in mind that the categories are here to provide some guidelines on judging an answer key and to allow a simple summary. The classifications must still be justified.

**Quality**  What is the quality level of the key? Does it contain any errors?
low: the model is riddled with errors and inconsistencies and does not follow any clear modelling principles at all.

medium: the model has some errors or inconsistencies, or violates some modelling principles (possibly for the sake of example).

high: the model is largely free of errors and inconsistencies, and it conforms to good modelling principles.

**Representative**  Is the answer key representative of the concepts that need to be modelled for the domain?

*not representative:* very few of the relevant domain concepts are present in the model and it does not provide an accurate conceptualisation compared to that of a domain expert.

*partially representative:* the model includes most of the relevant concepts and mostly conforms to a domain expert's conceptualisation, but there are some deviations and unnecessary modelling abstractions.

*representative:* the model includes practically all of the relevant concepts and appears to be consistent with the conceptualisation of a domain expert.

Note that a model may not be representative simply because it is too small, not necessarily because it is a *bad* model.

**Context** In what context was the answer key produced? Does it affect the resulting model?

We identify two contexts that could give rise to two very different models related to a particular evaluation text: text analysis based, and knowledge-based. The former indicates the model, or answer key, was primarily designed by analysing the text and, therefore, represents the key terms, concepts, and relations identified in the text. In this context, the answer key is likely to better represent the concepts and relations that can be extracted by an automated IE
tool; however, it may be less representative of the larger domain knowledge, depending on the coverage of the text.

The knowledge-based approach creates the model largely based on the world and domain-knowledge of the modeller. As such it may include additional concepts, relations, and abstractions that are not explicit, or even implicit, in the text. In this context, the model is possibly more representative of an expert’s domain knowledge; however, it is likely to diverge drastically from the text that it is supposedly based on.

Neither context is inherently better than the other, instead they potentially demonstrate different aspects of approaches to extracting models from the text. For example, in the text analysis context, the evaluation results demonstrate how much information an approach can extract from the text, while in the knowledge-based on context allow the demonstration of how well an approach can generate abstractions and infer implicit relations based on the information given (and possibly other background knowledge). While the latter is not necessarily achievable at the moment, it is worth considering producing an answer key in both contexts for each evaluation text so that the evaluation supports research into the future as NLP methods improve.

**Agreement** (To what extent) is there community agreement on the model represented by the answer key?

*no agreement:* The answer key appears to have been produced by a single author for their own purposes without any input from any other source. For example, this category would apply to models extracted from single author textbooks.

*little agreement:* The answer key may have had input from several authors, but there are very few (2-5) collaborators or it is unclear how much the model was actually developed jointly. For example, a model from a multi-author textbook would fall into this category.

*moderate agreement:* The model has been produced with a fair amount of collaboration between a number of people, including both technical analysts and domain experts.
However, it has not necessarily been developed through an explicit process for the purpose of creating a common conceptual model.

Community agreement:

The answer key has been produced by a large collaborative effort, including a process, for the purpose of creating a common conceptual model. Models produced (open) standards groups would fall into this category.

Test Procedure

There are two parts to the test procedure for vocabulary acquisition from unrestricted text. The first analyses the documents and extracts the vocabulary using the method described in Section 5.6.2. The second analyses the results of previous approaches within the framework described here. In both cases the vocabulary/model elements output by the approaches must be converted into the common format. This transformation is described in the following.

The transformation to the common format is straightforward for the SBVR-based vocabularies generated by our approach. This is due to SBVR's declarative, attribute-free, fact-based modelling approach. All ⟨object type⟩s and ⟨individual concepts⟩ are considered to be concepts, while all ⟨fact type⟩s are considered to be relations in which the ⟨role⟩s are the arguments.

The conversion of the other analysed approaches is a little more complex, but only in that there are more types of model element to consider. Since the other approaches use UML Class diagrams or ER models, which are both object-oriented approaches to modelling that include types, relationships, and attributes (UML models may also include methods). In these models, classes (UML) and entity types (ER) are mapped to concepts in the common model. On the other hand, relationships (or associations in UML), attributes, and methods are mapped to relations. For relationships, the ends of the relationship become the arguments of the relation. The arguments of attributes are taken from the class or entity type the attribute is defined on and the type of the attribute. Moreover, the arguments and return type of a method all become arguments of the equivalent relation in the common model.
Section 6.2. SBVR Lexicon Acquisition from “Unrestricted” Text

6.2.5 Results

The analysis of the source texts and answer keys is summarised in Tables 6.2 and 6.3, respectively. The EU-Rent answer key [OMG08b] is judged to be of high quality, even though it represents some (unnecessary) things for the sake of example. However, it is only partially representative as some of the example-based aspects are not strictly necessary, and it leaves some domain concepts out and simplifies others to reduce the size of the case study.

The answer key for the Library case study [Cal94] is considered to be of only medium quality as it appears to contain some errors. In particular, the `borrows` relationship only allows `<Book>`s to be borrowed, in contrast to the text and the superclass `<Loan Item>`—for which the name seems to imply that all loan items should be borrowable. In addition, the answer key appears to have been created by a single author and, hence, has very little agreement.

While the ATM [RBPE+91] answer key is considered to be of high quality and has some agreement, it is considered only partially representative. This is due to the note in the text that indicates that more work is needed to complete the design of the system/model.

An example of a representative answer is provided by the EFP case study [Der95]. This is classified as representative as it appears to be very thorough in describing the (albeit) limited domain. However, it is knowledge-based and appears to have very little agreement.

The last answer key, EFP-text [Der95], is an alternative to the previous that illustrates a text-based answer key. This is possible as Kurt Derr explains the process of analysing the text to identify candidate concepts and relations and then gradually refines them by applying guidelines for excluding candidates and introducing additional relevant concepts through identifying implicit information and general knowledge. The previous answer key, EFP, is the result of the final analysis, whereas EFP-text is the result of the initial text analysis. As a result, it is considered to be of medium quality due to the “error” produced by the text analysis and the example-based aspects of the model.

In Tables 6.5 and 6.6, the results of the vocabulary/model extraction are displayed, while Table 6.4 shows the original results reported in the cited approaches for comparison. Table 6.5 shows metrics and measures for each approach and Table 6.6 reports the categorisations of the correct extra, incorrect, and missing results. Although the approach of [SA14] was applied to
Chapter 6. Evaluation of the Application

### TABLE 6.2: Summary of evaluation text characteristics.

<table>
<thead>
<tr>
<th>Text</th>
<th>Length (words)</th>
<th>Style</th>
<th>Sent. Length (words/sent., min, max)</th>
<th>Completeness</th>
</tr>
</thead>
<tbody>
<tr>
<td>EU-Rent 1 [OMG08b]</td>
<td>v. short (472)</td>
<td>v. formal*</td>
<td>ave. (16.3, 7, 32)</td>
<td>mostly†</td>
</tr>
<tr>
<td>EU-Rent 1–6 [OMG08b]</td>
<td>short (3,642)</td>
<td>v. formal*</td>
<td>ave. (17.4, 6, 38)</td>
<td>mostly†</td>
</tr>
<tr>
<td>EU-Rent All [OMG08b]</td>
<td>ave. (7,057)</td>
<td>v. formal*</td>
<td>ave. (19.1, 6, 56)</td>
<td>mostly†</td>
</tr>
<tr>
<td>Library [Cal94]</td>
<td>v. short (217)</td>
<td>informal</td>
<td>short (14.5, 7, 31)</td>
<td>mostly</td>
</tr>
<tr>
<td>ATM [RBPE+91]</td>
<td>v. short (168)</td>
<td>informal</td>
<td>ave. (16.8, 7, 27)</td>
<td>mostly</td>
</tr>
<tr>
<td>ATM-mod [SA14]</td>
<td>v. short (142)</td>
<td>informal</td>
<td>short (10.9, 7, 16)</td>
<td>mostly</td>
</tr>
<tr>
<td>EFP [Der95]</td>
<td>v. short (241)</td>
<td>informal</td>
<td>long (21.9, 11, 43)</td>
<td>mostly</td>
</tr>
</tbody>
</table>

* Includes some less formal aspects.
† It is not complete due to the way the text was created from the definitions and rules.

### TABLE 6.3: Summary of answer key properties.

<table>
<thead>
<tr>
<th>Text</th>
<th>Quality</th>
<th>Representative</th>
<th>Context</th>
<th>Agreement</th>
</tr>
</thead>
<tbody>
<tr>
<td>EU-Rent [OMG08b]</td>
<td>high</td>
<td>partial</td>
<td>mixed* community</td>
<td></td>
</tr>
<tr>
<td>Library [Cal94]</td>
<td>medium</td>
<td>no/partial</td>
<td>knowledge-based</td>
<td>none/some</td>
</tr>
<tr>
<td>ATM [RBPE+91]</td>
<td>high</td>
<td>partial</td>
<td>knowledge-based</td>
<td>some</td>
</tr>
<tr>
<td>ATM-mod [SA14]</td>
<td></td>
<td></td>
<td>Uses the same answer key as the original ATM case study with the exclusion of ⟨Customer⟩ and related relations as they are not included in the modified text.</td>
<td></td>
</tr>
<tr>
<td>EFP [Der95]</td>
<td>high</td>
<td>yes</td>
<td>knowledge-based</td>
<td>none/some</td>
</tr>
<tr>
<td>EFP-text [Der95]</td>
<td>medium</td>
<td>yes</td>
<td>text-based</td>
<td>none</td>
</tr>
</tbody>
</table>

* The nature of SBVR SE means the text is related to the model; however, there is a disconnect due to the modification of the text and the intention is to be knowledge-based.

the EFP case study as well, we excluded it from the comparative analysis as the generated model included many elements that were not explicitly, nor implicitly, included in the text. Therefore, we suspect that the authors used a different version of the source text than the one we had available. As a result, it would have been nonsensical to report the results.
### Table 6.4: Original case study evaluation results.

<table>
<thead>
<tr>
<th>Case Study</th>
<th>Source</th>
<th>$N_{cor}$</th>
<th>$N_{inc}$</th>
<th>$N_{mis}$</th>
<th>$N_{ext}$</th>
<th>REC %</th>
<th>PRE %</th>
<th>OVS %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library</td>
<td>[HG03]*</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>75</td>
<td>75</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>[BC12]†</td>
<td>35</td>
<td>3</td>
<td>2</td>
<td>—</td>
<td>88</td>
<td>92</td>
<td>—</td>
</tr>
<tr>
<td>ATM</td>
<td>[EVR11]*</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>91†</td>
<td>91</td>
<td>16</td>
</tr>
<tr>
<td>ATM-mod</td>
<td>[SA14]*</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>91</td>
<td>91</td>
<td>8</td>
</tr>
<tr>
<td>ATM-mod</td>
<td>[SA14]§</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>50</td>
<td>100</td>
<td>60</td>
</tr>
<tr>
<td>EFP</td>
<td>[EVR11]*</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>70</td>
<td>77</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>[SA14]*</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>85</td>
<td>94</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>[SA14]§</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>61</td>
<td>88</td>
<td>61</td>
</tr>
</tbody>
</table>

* Concepts only; † Includes classes, attributes, associations, and methods; ‡ Corrected from source; § Relations only.

Note: The individual values were not provided in [SA14]. Moreover, the results reproduced here include the *implicit* elements, i.e., the elements in the key that are not present in the text are not ignored; this is consistent with the results of the other approaches.
### Table 6.5: Case study evaluation: results using the framework.

<table>
<thead>
<tr>
<th>Case Study</th>
<th>$N_{key}$</th>
<th>Method</th>
<th>$N_{cor}$</th>
<th>$N_{key}^{cor}$</th>
<th>$N_{inc}$</th>
<th>$N_{mis}$</th>
<th>$N_{ext}$</th>
<th>REC %</th>
<th>PRE %</th>
<th>PRE$_{key}$ %</th>
<th>OVS %</th>
<th>$F_1$ %</th>
</tr>
</thead>
<tbody>
<tr>
<td>EU-Rent 1</td>
<td>21</td>
<td>CLUE</td>
<td>59</td>
<td>21</td>
<td>8</td>
<td>0</td>
<td>38</td>
<td>100 (100)</td>
<td>88</td>
<td>72</td>
<td>181</td>
<td>94 (94)</td>
</tr>
<tr>
<td>EU-Rent 1–6</td>
<td>200</td>
<td>CLUE</td>
<td>383</td>
<td>156</td>
<td>113</td>
<td>44</td>
<td>227</td>
<td>78 (99)</td>
<td>77</td>
<td>58</td>
<td>114</td>
<td>78 (87)</td>
</tr>
<tr>
<td>EU-Rent All</td>
<td>380</td>
<td>CLUE</td>
<td>636</td>
<td>274</td>
<td>213</td>
<td>106</td>
<td>362</td>
<td>72 (92)</td>
<td>75</td>
<td>56</td>
<td>95</td>
<td>73 (83)</td>
</tr>
<tr>
<td>Library</td>
<td>42</td>
<td>CLUE</td>
<td>50</td>
<td>22</td>
<td>10</td>
<td>20</td>
<td>28</td>
<td>52 (79)</td>
<td>83</td>
<td>69</td>
<td>67</td>
<td>64 (81)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[HG03]</td>
<td>25</td>
<td>15</td>
<td>8</td>
<td>27</td>
<td>10</td>
<td>36 (52)</td>
<td>76</td>
<td>65</td>
<td>24</td>
<td>49 (61)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[BC12]</td>
<td>40</td>
<td>28</td>
<td>10</td>
<td>14</td>
<td>12</td>
<td>67 (93)</td>
<td>80</td>
<td>74</td>
<td>29</td>
<td>73 (86)</td>
</tr>
<tr>
<td>ATM</td>
<td>32</td>
<td>CLUE</td>
<td>47</td>
<td>16</td>
<td>7</td>
<td>16</td>
<td>31</td>
<td>50 (89)</td>
<td>87</td>
<td>70</td>
<td>97</td>
<td>64 (88)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[EVR11]</td>
<td>17</td>
<td>7</td>
<td>11</td>
<td>25</td>
<td>10</td>
<td>22 (39)</td>
<td>61</td>
<td>39</td>
<td>31</td>
<td>32 (47)</td>
</tr>
<tr>
<td>ATM-mod</td>
<td>32</td>
<td>CLUE</td>
<td>40</td>
<td>13</td>
<td>9</td>
<td>16</td>
<td>27</td>
<td>45 (81)</td>
<td>82</td>
<td>59</td>
<td>93</td>
<td>58 (81)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[SA14]</td>
<td>32</td>
<td>14</td>
<td>3</td>
<td>15</td>
<td>18</td>
<td>48 (82)</td>
<td>91</td>
<td>82</td>
<td>62</td>
<td>63 (87)</td>
</tr>
<tr>
<td>EFP</td>
<td></td>
<td>CLUE</td>
<td>42</td>
<td>7</td>
<td>13</td>
<td>14</td>
<td>35</td>
<td>33 (70)</td>
<td>76</td>
<td>35</td>
<td>167</td>
<td>46 (73)</td>
</tr>
<tr>
<td>EFP-text</td>
<td></td>
<td>CLUE</td>
<td>43</td>
<td>23</td>
<td>12</td>
<td>18</td>
<td>20</td>
<td>56 (56)</td>
<td>78</td>
<td>66</td>
<td>49</td>
<td>65 (65)</td>
</tr>
</tbody>
</table>

* Values in brackets are the Recall % excluding missing elements in the ‘implied’ and ‘not in text’ categories.
Table 6.6: Case study evaluation: extra, missing, and error categorisation.

<table>
<thead>
<tr>
<th>Case Study</th>
<th>Method</th>
<th>Extra</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>syn</td>
<td>gen</td>
<td>spec</td>
<td>trunc</td>
<td>exp</td>
<td>in txt</td>
<td>arg</td>
<td>parse</td>
<td>err</td>
<td>trunc</td>
<td>exp</td>
<td>irr</td>
<td>wrong</td>
<td>parse</td>
<td>impl</td>
<td>not in</td>
<td>inc</td>
</tr>
<tr>
<td>EU-Rent 1</td>
<td>CLUE</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>32</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EU-Rent 1–6</td>
<td>CLUE</td>
<td>27</td>
<td>9</td>
<td>39</td>
<td>5</td>
<td>13</td>
<td>134</td>
<td>6</td>
<td>21</td>
<td>31</td>
<td>6</td>
<td>29</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>33</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>EU-Rent All</td>
<td>CLUE</td>
<td>40</td>
<td>22</td>
<td>74</td>
<td>13</td>
<td>34</td>
<td>179</td>
<td>8</td>
<td>44</td>
<td>41</td>
<td>15</td>
<td>73</td>
<td>32</td>
<td>1</td>
<td>8</td>
<td>74</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>Library</td>
<td>CLUE</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>16</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>6</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>[HG03]</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>6</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>[BC12]</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>ATM</td>
<td>CLUE</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>20</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>8</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>[EVR11]</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>6</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>ATM-mod</td>
<td>CLUE</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>23</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>[SA14]</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>5</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>EFP</td>
<td>CLUE</td>
<td>8</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>21</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>EFP-text</td>
<td>CLUE</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>0</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>

* For the cited approaches, errors that have been placed in the other categories may have been caused by parsing errors.

† For the cited approaches, some missing elements may be caused by parsing errors; however, if it appears that there is a rule to extract the term it is categorised as filtered, otherwise it has been categorised as incomplete.
6.2.6 Discussion

In general, the different approaches perform more poorly than their original results indicate. This is, in large part, due to most approaches only evaluating class/concept extraction, while here we have included relation extraction. However, this is not to say that the approaches do not extract valid relations, only that they are not equivalent to the relations defined in the key. As the answer keys used in these evaluations are all knowledge-based, they include a large amount of implied or additional relations that are not included in the text.

Overall our approach performs very well, outperforming two of the four alternatives. The approach of [SA14] performs very similar to ours, which is not surprising as it was the source of many of the extraction rules incorporated into our solution; however, it outperforms ours slightly. This indicates the set of extraction rules used in [SA14] has good coverage, while the incorporation of additional rules from other sources into our own approach has created more noise than benefit (while they have quite high over-specification, ours is yet higher with an accompanying increase in incorrect results). Moreover, it seems our approach is hampered by parser errors not present in other approaches, even those using the Stanford Parser as well. This may be a result of the choice of parser model used (we use the RNN model, it is unknown which model is used by [SA14]) and/or our use of GATE's tokenisation and POS-tagging components rather than those of the Stanford NLP tools. Finally, incorporating a rule to correctly identify the composition relationship between ⟨Consortium⟩ and ⟨Bank⟩ would result in a slightly higher recall performance than that of [SA14].

The only other approach that outperforms our own is that of [BC12]. However, some assumptions were made in their favour during the conversion of their results into the common framework: chiefly in the association of attributes with a concept. This was due to inconsistencies between the table [BC12 top of p. 232] detailing the extracted elements, with the diagram [BC12 bottom of p. 232] displaying them. Moreover, of the 5 case studies evaluated in [BC12] the Library example showed the highest performance by a significant margin: the average results reported were 80.73% recall, 85.27% precision, and 82.95% F₁ with the closest result at 83.67% recall, 85.41% precision, and 84.53% F₁. Finally, they utilise a rule-based NLP system that resolves analyses against a knowledge-base [BC06] and it is unclear from [BC12] how much a priori knowledge has been added to the knowledge-base. Therefore, additional
common case studies need to be evaluated to determine the true relationship between the two approaches.

The results of the EU-Rent case study are much better than the other case studies in terms of recall. This is likely due to the more formal nature of the text making it easier to extract the correct terms in general; however, some aspects and restrictions of the SBVR-based text cause more parsing errors. For example, the ‘if and only if’ construction and the longer sentences with nested relative clauses often result in incorrect dependencies. Moreover, more formal nature means the text is closer to the expected model in that the text is written predominantly using the terms defined in the vocabulary.

The analysis of the EU-Rent case studies indicate a decrease in precision as the length of the text increases. The increase in spurious results is most likely caused by the lack of filtering based on frequency in our approach. While many other approaches place a threshold on the frequency of a term for it to be extracted (even if that threshold is as low as 2), we chose not to since a relevant term may only occur once in the short texts being analysed. However, in the longer texts, a large proportion of the spurious results (≈ 44%) are from terms extracted with a frequency of 1. Filtering such terms would lead to an increase in precision. Moreover, it would lead to a decrease in over-specification, as many of the correct extra terms occur with a frequency of 1.

Reducing the over-specification is possibly a good outcome as it appears very high for the EU-Rent case study, with 181% for the analysis of the first section. While further research is needed to determine what a good level of over-specification is (some work, e.g. [EVR11], indicates that over-specification should be as low as possible, while the conventional wisdom is that over-specification is good as it is easier for a person to filter irrelevant terms than to identify new ones [Ful01]), it can be observed that the over-specification of the complete EU-Rent case study drops to approximately the same level as the other case studies processed by our approach (≈ 95%). This is due to the partial case studies making references to the vocabulary defined by the other sections, i.e., they are not independent. Furthermore, the case study references terms from the core SBVR vocabulary, which are considered correct extractions. Finally, over-specification can be benefited by incorporating better identification of generalisations for concepts, since many of the correct extra relations are specialisations.
of relations involving the generalisations of the arguments. For example, \( \text{includes}((\text{advance rental}), (\text{car movement})) \) is a specialisation of \( \text{includes}((\text{rental}), (\text{car movement})) \), where \( (\text{advance rental}) \) is a specialisation of \( (\text{rental}) \). Performing synonym resolution would also lead to a decrease in over-specification.

### 6.2.7 Limitations of this Comparison

The analysis of previous approaches is limited to those that provide information on the source case study and the produced models. If one or the other is not available then the approach is not included in the comparison. Unfortunately this leaves very few approaches with which to perform a detailed comparison; however, it is the first steps towards providing more comparable results between approaches. When analysing the output models of approaches, where something is not clear, e.g. due to a lack of information in the source, the decision was made in their favour. This seemed fair as aspects of the comparison, particularly the categorisation of the results, is still largely subjective.

In addition, the analysis of the source texts and answer keys is largely subjective and hampered by incomplete information. However, the chosen categorisations reflect our analysis based on the information that was available and considering the context that the answer key and texts were created within. Ideally, we would be able to perform comparisons across the spectrum of possible text and answer key types; however, we were limited by those that were available and comparable in the literature.

Due to the generalisation of the conceptual model to consist of only concepts and relations, the evaluation framework defined here does not take into account certain aspects that may be relevant for different applications—for example, identifying attributes vs. entities in ER models, identifying operations (or methods) in addition to associations in object-oriented class models (such as UML). Such important evaluation can be produced alongside that described by the proposed evaluation framework. Moreover, there is nothing preventing the inclusion of modelling language or application specific results in the evaluation of a system. The evaluation framework described here attempts to abstract out the common elements of these different systems to allow a comparative performance evaluation across them.
Furthermore, this evaluation framework does not consider specific relations (e.g. generalisation, composition, or aggregation). However, this does not prevent these relationships from being extracted nor from being included in the answer key. For example, generalisation is simply a relation with a common name, such as \( \text{is-a}(\langle \text{Book} \rangle, \langle \text{Loan Item} \rangle) \), where the synonym rules would allow the detection of equivalence to, for example, \( \text{subtype-of}(\langle \text{Book} \rangle, \langle \text{Loan Item} \rangle) \). Moreover, the evaluation is still able to report the identification of specific relations if desired and relevant to the application.

A potentially important aspect that is not covered at all in the current framework is the cardinalities, or multiplicities, of relations. However, very few approaches extract cardinality—and since we purposely ignore it in our approach as the primary NLU component will identify them—we decided to exclude the cardinality of relations from the evaluation framework at the present time.

### 6.3 Cognitive Linguistic Rule Parsing of SBVR

In this section we evaluate the performance of parsing SBVR SE-based rules using our cognitive linguistic and configuration based framework, CLUE4SBVR. As before, we describe the criterion, measure, method, and results of the evaluation.

#### 6.3.1 Criterion

To evaluate the rule parsing we compare how close the output of the system is to the intended formal interpretation of a set of rules. We continue to use the SBVR EU-Rent case study as a (human created) answer key, from which the text styling serves to identify the intended formal interpretation of a rule.
Chapter 6. Evaluation of the Application

6.3.2 Measure

The comparison between the system output and answer key is measured using the three metrics Precision, Recall, and $F_1$-measure. For the evaluation of the parsing, these metrics use the simple definitions described in Definitions 6.1, 6.2, and 6.3 respectively.

6.3.3 Method

To calculate the measure, we compare the system response to the answer key. We represent the system response and answer key as sets of annotations on the processed documents, rather than comparing SBVR models directly. This was done to leverage the automated evaluation framework built into GATE and to avoid some of the complexity of aligning and matching object-oriented models directly. Moreover, this approach eliminates the impact of the vocabulary on results, since the vocabulary is not included in the annotations used for the parsing evaluation but is present in the models due to the linking of logical forms to the vocabulary/concepts used. In the following we describe how the evaluation was performed including the selection of test data, creation of the answer key, and comparison of the annotations sets.

**Test Data** For the parsing evaluation we extracted the structural rules included in the vocabulary sections of the EU-Rent case study (section E.2.2.1.1–E.2.2.1.11) and the all of the rules included in the separate rules section of the case study (sections E.2.2.2.1–E.2.2.2.11). The rules were extracted as plain-text from the PDF, without any of the SBVR SE text styling, however, structural aspects remain intact.

The rules included in the EU-Rent case study demonstrate a variety of phenomena occurring in SBVR-based specifications including:

- rules of varying length and complexity;
- different types of quantification, e.g. universal, existential, and exactly-n;
Section 6.3. Cognitive Linguistic Rule Parsing of SBVR

- various logical operations, e.g. conjunction (‘and’), disjunction (‘or’), implication (‘if’), and equivalence (‘if and only if’);
- rules of different modality, e.g. necessity for structural rules and obligation for operational rules; and
- higher-order logic through the categorisation of concepts by others.

Answer Key The annotations for the answer key were manually created by a single person based on the EU-Rent case study, in which the text styling indicates the correct interpretation of a term. Where multiple syntactic analyses are possible that may lead to different (but equally correct) semantic compositions, a single candidate was chosen for the answer key that reflected the desired properties: for example, certain parses are expected to be preferred due to the catch/caught weights of keywords. For the semantics, SBVR permits the existence of different models representing the same meaning; in these cases only one possibility was chosen for inclusion in the answer key. Moreover, where multiple interpretations are possible, due to SBVR not completely eliminating ambiguity, only the interpretation considered to be the intended interpretation is incorporated into the answer key.

The types of annotation used in the parsing evaluation include annotations for references to object types, individual concepts, fact types, or keywords, and annotations for the syntactic parse tree. Note that when an object type is referred to as the concept itself, as opposed to a thing of that type, it is annotated with an individual concept annotation. The attributes of the annotations are as follows:

- **kind**: Is an attribute used by annotations that reference an object type, individual concept, fact type, or keyword to indicate the specific type(s) of the term. Similar to the concept type attribute for the vocabulary annotations.

- **designation**[1..n]:
  The designation attribute(s) provide the term(s) used to refer to the term (object type, individual concept, or fact type) being referenced.

- **fact type form**[1..n]:
  The full textual form of a fact type reference. The fact type form attributes are aligned with the designations.
Each car movement has exactly one movement-id.

F I G U R E 6.5: Example answer key for rule parsing.

**bracketing:**

The **bracketing** attribute is defined for syntax annotations and presents the syntactic analysis in a textual, bracketed form.

An example rule and its annotations are displayed in Figure 6.5.

**Test Procedure** The evaluation is performed on a particular document or corpus by running the process described in Section 5.4 and using the evaluation component of GATE to compare the generated annotations with those of the answer key. As before, GATE determines the correct, missing, and incorrect (false positive) annotations to calculate precision, recall, and $F_1$ values. The vocabulary/lexicon used in this evaluation is a correct lexicon, rather than using the potentially incorrect output of the vocabulary acquisition process. Finally, to provide some indication of the performance on business specifications of varying sizes and complexity, the evaluation is performed on each of the first six sections of data individually and then combined.
TABLE 6.7: Parsing accuracy results

<table>
<thead>
<tr>
<th>Category</th>
<th>Section 2.2.1.X</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>1–6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rules</td>
<td>#</td>
<td>13</td>
<td>8</td>
<td>15</td>
<td>8</td>
<td>16</td>
<td>20</td>
<td>80</td>
</tr>
<tr>
<td>Object types</td>
<td>#</td>
<td>17</td>
<td>36</td>
<td>24</td>
<td>11</td>
<td>75</td>
<td>43</td>
<td>206</td>
</tr>
<tr>
<td></td>
<td>P</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.97</td>
<td>1.00</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>F₁</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.99</td>
<td>1.00</td>
<td>0.99</td>
</tr>
<tr>
<td>Individual concepts</td>
<td>#</td>
<td>17</td>
<td>0</td>
<td>20</td>
<td>10</td>
<td>1</td>
<td>26</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>P</td>
<td>1.00</td>
<td>N/A</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>1.00</td>
<td>N/A</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td>1.00</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>F₁</td>
<td>1.00</td>
<td>N/A</td>
<td>1.00</td>
<td>1.00</td>
<td>0.00</td>
<td>1.00</td>
<td>0.99</td>
</tr>
<tr>
<td>Fact types</td>
<td>#</td>
<td>16</td>
<td>23</td>
<td>19</td>
<td>10</td>
<td>52</td>
<td>33</td>
<td>153</td>
</tr>
<tr>
<td></td>
<td>P</td>
<td>1.00</td>
<td>1.00</td>
<td>0.90</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.96</td>
<td>1.00</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>F₁</td>
<td>1.00</td>
<td>1.00</td>
<td>0.95</td>
<td>1.00</td>
<td>0.98</td>
<td>1.00</td>
<td>0.99</td>
</tr>
<tr>
<td>Keywords</td>
<td>#</td>
<td>23</td>
<td>51</td>
<td>32</td>
<td>15</td>
<td>111</td>
<td>62</td>
<td>294</td>
</tr>
<tr>
<td></td>
<td>P</td>
<td>1.00</td>
<td>0.84</td>
<td>1.00</td>
<td>1.00</td>
<td>0.86</td>
<td>1.00</td>
<td>0.91</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.97</td>
<td>1.00</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>F₁</td>
<td>1.00</td>
<td>0.91</td>
<td>1.00</td>
<td>1.00</td>
<td>0.91</td>
<td>1.00</td>
<td>0.95</td>
</tr>
<tr>
<td>Syntax tree</td>
<td>#</td>
<td>13</td>
<td>8</td>
<td>15</td>
<td>8</td>
<td>16</td>
<td>20</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>P</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>F₁</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Total</td>
<td>#</td>
<td>86</td>
<td>118</td>
<td>110</td>
<td>54</td>
<td>255</td>
<td>184</td>
<td>807</td>
</tr>
<tr>
<td></td>
<td>P</td>
<td>1.00</td>
<td>0.92</td>
<td>0.98</td>
<td>1.00</td>
<td>0.93</td>
<td>1.00</td>
<td>0.96</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.97</td>
<td>1.00</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>F₁</td>
<td>1.00</td>
<td>0.96</td>
<td>0.99</td>
<td>1.00</td>
<td>0.95</td>
<td>1.00</td>
<td>0.98</td>
</tr>
</tbody>
</table>

6.3.4 Results

The results of parsing the rules are summarised in [Table 6.7] which displays the number of answer key annotations, precision, recall, and \( F_1 \)-score for each of the categories of annotation (object type, individual concept, fact type, keywords, and syntax) as well as the totals across the categories. In addition, the first non-header row displays the total number of rules. As for the vocabulary learning evaluation, the totals are calculated by the *micro summary*, in which the statistics of correct, incorrect, and missing are taken across the entire document (or corpus) to calculate the metrics. This provides a more accurate result than averaging the results of the categories.
6.3.5 Discussion

The accuracy of the system is excellent (overall $F_1$-score of 98%), with the correct interpretation of almost every rule being found (99% recall). While this analysis is performed on the SBVR SE CNL, the fact that this level of accuracy is being achieved without a fully specified formal grammar suggests the efficacy of our approach. Moreover, since a formal version of SBVR SE [FCGE14] can only analyse 84% of rules provided by domain experts, the flexibility provided by our approach is promising.

Although the results show a lower precision than recall (96% precision overall), this is due to finding multiple semantic solutions for some rules and, therefore, demonstrates a desired behaviour of our approach. In a complete system, the alternative interpretations would be presented to the user, allowing them to select the intended interpretation with which to update the specification. It should be noted that these results are achieved by assuming a correct lexicon and that future work will investigate the error handling capabilities of the system. However, it shows that if the lexicon can be learnt correctly, our approach is capable of determining the correct formal model of rules using that vocabulary.

Finally, there is one exception preventing a perfect recall score of 100%, which occurred in section 2.2.1.5 of the case study. Since a correct syntactic analysis was found for all rules (100% recall for syntax annotations), the issue is that the configuration process was unable to find a semantically sound interpretation of the rule. We determined that the reason for this is that the rule specifies an amount with a unit of measure; however, the amount is incompatible with the role of the relationship as it is defined in terms of the unit of measure. As a result, the type constraints are violated and the configuration fails. Further investigation is needed to determine whether or not this is an issue with our mappings to SBVR or if it is a problem with SBVR itself. At present we are unsure of the intended SBVR model that should result from such a construction since a similar example in the SBVR specification itself [OMG08b, p. 71] seems to circumvent the issue by defining the relationship with respect to thing (the most general concept of SBVR), thereby allowing an instance of any type (whether it be an amount or the unit of measure) to be referenced in the relationship. Obviously, this goes against our goal of being able to perform semantic consistency/conformance checking and violates good modelling practice.
6.4 Run-time Performance

Although our current focus is on accuracy rather than the efficiency of our prototype, it is important to consider its execution time as an indicator of the feasibility of the approach in real-world scenarios. Therefore, to identify possible limitations to this aim we performed a brief evaluation of the run-time performance of the approach. The following explains the criterion, measure, and method for this analysis.

6.4.1 Criterion

In order to evaluate the run-time performance of the approach we look at how quickly the prototype performs the text analysis. To help identify where possible performance issues exist, we consider several aspects: suggestion time (i.e. the time to perform the syntactic analysis), configuration time (i.e. the time to perform the semantic analysis of a single suggestion), and total execution time.

6.4.2 Measure

There are three main factors influencing the execution time:

1. the time to perform the configuration, which is in turn affected by:

   (a) the size of the model being configured,

   (b) the number of constraints, and

   (c) the number of backtracks performed before reaching a valid configuration,

2. the number of suggestions in the suggestion list (controlled through pruning and early accommodation), and

3. the number of suggestions with equal ranking (caused by various ambiguities, primarily polysemy).
Therefore, to take these factors into account, we use the following metrics to evaluate the run-time performance:

**Suggestion time:** For the analysis of the time to perform the syntactic analysis we consider the following metrics.

1. Maximum number of suggestions in the suggestion list at any given time.
2. Number of suggestions sent to the configurator for configuration.
3. Run-time taken to generate the suggestions.

**Configuration time:** The time it takes the configurator to perform the configuration is taken into account by the metrics below.

1. Number of variable assignments made during configuration.
2. Number of constraints evaluated during configuration.
3. (Minimum, Maximum, and Average) Number of backtracks performed.
4. Number of components generated by the configurator.
5. Configuration run-time (seconds).

**Total time:** Finally, the overall execution time, encompassing both syntactic and semantic analysis and everything in between, is captured using the following metrics.

1. Total run-time.
2. Average time per suggestion.

### 6.4.3 Method

For this evaluation we apply the prototype to two example rules: a short, simple rule with limited ambiguity (1), and a longer more complex rule with nested restrictions and greater ambiguity (2).

(1) Each branch is included in exactly one local area.
Section 6.4. *Run-time Performance*

(2) The country that is of a branch is the country that is of the operating company that includes the local area that includes the branch.

Furthermore, the overall execution time is determined based on a minimal vocabulary needed to parse each rule, as well as a combined vocabulary. By comparing the two, it provides an indication of the effects the processing due to a larger vocabulary model and the added ambiguity caused by polysemy (e.g. from the multiple `includes` fact types).

Finally, as the prototype is a relatively simple implementation of the approach it is worth noting that this evaluation provides a “worst-case” analysis of the execution time. In particular, the accommodation of suggestions is left until the end of a rule is reached, rather than performing it iteratively. This means that there are more suggestions in the suggestion list than necessary, leading to potentially increased execution times. There is one optimisation, however, due to the collapsed representation of polysemous lexical entries (discussed in Section 5.4) which are left under-specified in the suggestion representation (as they are considered variants) until the suggestion is configured. This helps maintain a suggestion list of a reasonable size without iterative accommodation. On the other hand, performing the configuration at the end, rather than iteratively, reduces the number of configurations that occur as only the top-ranked suggestions are sent to the configuration process. This is because configuring the top-ranked suggestions configures the smaller (lower-ranked) suggestions as needed due to their incorporation in the larger suggestions). Overall, however, it still represents more of a worst-case scenario, due to the possibility of multiple top-ranked suggestions resulting in more configurations of larger models—since some of the top-ranked suggestions could have been pruned when they were small if configuration were occurring iteratively.

This effect due polysemous entries can be seen in the metrics for the maximum number of suggestions, which is the maximum number of suggestions that were in the suggestion list at any one time, and the number of suggestions that were configured, which is the number of semantic variants of the top-ranked suggestion configured at the end.
6.4.4 Results

The results of the execution time experiments are summarised in Tables 6.8 and 6.9. The configuration time results are separated out into Table 6.8 while Table 6.9 includes both the suggestion and total time metrics.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Rule 1</th>
<th>Rule 2</th>
</tr>
</thead>
<tbody>
<tr>
<td># Variable Assignments</td>
<td>267</td>
<td>859</td>
</tr>
<tr>
<td># Constraints</td>
<td>114</td>
<td>434</td>
</tr>
<tr>
<td>Min. # Backtracks</td>
<td>9</td>
<td>93</td>
</tr>
<tr>
<td>Max. # Backtracks</td>
<td>11</td>
<td>118</td>
</tr>
<tr>
<td>Ave. # Backtracks</td>
<td>10</td>
<td>106</td>
</tr>
<tr>
<td># Components Generated</td>
<td>6</td>
<td>24</td>
</tr>
<tr>
<td>Ave. Configuration Time (s)</td>
<td>0.08</td>
<td>0.42</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Individual</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rule 1</td>
<td>Rule 2</td>
</tr>
<tr>
<td>Suggestion Time (s)</td>
<td>0.01</td>
<td>1.34</td>
</tr>
<tr>
<td>Max. # Suggestions</td>
<td>11</td>
<td>134</td>
</tr>
<tr>
<td># Suggestions Configured</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>Total Time (s)</td>
<td>0.93</td>
<td>9.73</td>
</tr>
<tr>
<td>Seconds per Suggestion</td>
<td>0.93</td>
<td>0.61</td>
</tr>
</tbody>
</table>

6.4.5 Discussion

The statistics displayed in Tables 6.8 and 6.9 show some interesting results. First of all, the configuration itself executes quite quickly, while the entire process takes quite a large amount of time for the second rule. This shows that repeated configurations of the entire sentence cause significant overhead, which incremental accommodation should help to improve. Furthermore, it indicates that future effort should be put into minimising the number of suggestions that are sent to configuration. Since many of the additional suggestions are created by treating polysemous lexical entries as different suggestions come configuration time, the overall time could be reduced by relaxing the catch constraint on a specific lexical entry. This would allow
the configurator to search for the appropriate one during a single execution of the configuration. Another possibility is to use parallelisation to perform configuration on multiple suggestions at the same time; the prototype currently configures each suggestion in sequence.

Secondly, the time per suggestion statistics show that less time is spent per suggestion on the longer example than the shorter one (0.61s vs. 0.93s). At first glance this seems odd since we know that the configuration takes longer on the larger models. The apparent contradiction is caused by many of the alternative suggestions creating malformed [SBVR] models, which fail the configuration process quickly. This reiterates the need to minimise the number of suggestions that make it to the configuration process in order to improve run-time performance.

Finally, the large discrepancy between the actual time to configure rule [1] and the time taken for the overall processing of the rule (0.08s vs. 0.93s) is an indication that there is a large amount of overhead in the prototype implementation. Through analysis of the program execution, we determine that approximately 60% of the time to perform the configuration is used for the serialisation and de-serialisation of large [SBVR] models, due to the loose coupling between the [GATE] components and the [COCOA]. Therefore, future work will need to look at more tightly integrating the two aspects of the process to reduce the overhead and obtain acceptable performance.

### 6.5 Summary

While the framework and prototype implementation, CLUE4SBVR, that we have introduced in this thesis has been designed to achieve specific goals, it must be tested to determine whether or not it can achieve them. To that end we presented a performance evaluation of CLUE4SBVR’s individual components in this chapter. Each component of the prototype was evaluated starting the with lexical acquisition components and then the parsing component. Moreover, the evaluation of the component for lexical acquisition from unrestricted text included a comparison to related work based on an evaluation framework developed in this chapter. This framework is intended to overcome the shortcomings of the disparate and incomparable evaluations performed by such tools to allow more accurate comparisons to be made. Finally,
Chapter 6. *Evaluation of the Application*

we presented a run-time performance evaluation to determine the implementation’s efficacy in a real-world environment.

The results presented in this chapter are very promising; in particular those for the lexicon acquisition from glossary and the subsequent rule parsing. While the sub-par run-time performance may be an issue for real-world applications, we have identified a number of ways in which it can be improved in future versions of the prototype. In the following chapter we will discuss what these results mean with respect to the overall goals of the thesis and conclude the dissertation.
Chapter 7

Discussion and Conclusions

In this thesis we focused on developing an approach to supporting the formalisation of natural language requirements specifications by both technical experts and non-technical business (domain) experts in the context of MDE. To achieve this, previous approaches to creating models from natural language specifications were investigated and determined to be inadequate. An alternative was proposed that is based on a flexible, knowledge-based approach to processing Controlled Natural Languages (CNLs). A prototype implementation, CLUE4SBVR, of the general approach was then created, which provided good results in creating precise, formal interpretations of SBVR SE-based vocabularies and rules.

A set of criteria was formulated with which to analyse previous approaches to creating models from natural language specifications in the context of supporting domain experts to formalise their specifications. This analysis revealed that none of the previous work was suited to supporting non-technical users in formalising their requirements. Instead, existing tools were found to be suited to different situations: either a formal “start from scratch” approach, or a minimal effort attempt to reveal information hidden in an organisation’s documentation.

This revelation led to development of an approach that could fulfil the criteria, which include: an appropriate level of abstraction for domain experts, a high-level of expressiveness, an intuitive and natural language utilising no technical notations, the ability to incorporate existing documentation, and the support for providing high-quality feedback on errors, inconsistencies, and ambiguities found in the specification. This new approach was based on the theory of
Cognitive Grammar and integrates MDE and configuration such that knowledge is embedded into, and exploited by, the parser. The general framework for this approach was illustrated on a simple CNL utilising a simple Conceptual Graphs formalism for its semantics, demonstrating the flexibility of the approach, before being applied to the intended case of requirements specifications.

The application to specifications produced the prototype tool CLUE4SBVR, which implemented the general approach based on the SBVR meta-model for semantics and SBVR SE as the supported controlled language. In addition, the prototype incorporated two modules for acquiring the initial vocabulary and knowledge with which it uses to process the rules of a specification. The first acquisition component allows users to explicitly specify terms in a glossary using minimal special notation, while the second attempts to acquire the vocabulary from existing documentation using IE techniques. The resulting tool fulfils the defined criteria in the following ways:

H1 a high-level of abstraction is supported by the models created by CLUE4SBVR as the underlying SBVR models can be at the CIM level as desired. This is the level most appropriate for business and domain experts as it is abstracted from system considerations and focuses on their domain model and requirements.

H2 the SBVR SE controlled language, on which the application is based, is a very natural and intuitive CNL with a relatively small set of guidelines to learn to use it effectively.

H3 the underlying SBVR meta-model was selected as the target semantics due to its ability to handle the requirements of the users, including:

H3.1 requirements for both structural and operational aspects of a specification;
H3.2 requirements based on higher-order concepts and categories, and
H3.3 requirements with different modalities, i.e. necessity, possibility, obligation, and permission.

H4 the tools includes components that allow it to be used with arbitrary documents, e.g. the vocabulary acquisition from unrestricted text, and the inherent flexibility of the parsing process should allow it to make some sort of sense out of existing documentation through an iterative process parsing and revision of errors/ambiguities.

232
Section 7.1. Comparison to Surveyed Approaches

H5 the use of technical notation is severely limited due to the basis in SBVR SE; the glossary entries for verbs (fact types) do require the user to specify the verb form including the terms that it relates, though. However, this does not really require any technical notation as it is still simply natural language.

H6 the use of configuration to perform semantic analysis allows detailed feedback on errors (through constraint violations, etc.) and ambiguities (i.e. multiple solutions) in a specification.

Moreover, the subsequent performance evaluation of CLUE4SBVR suggests that the tool is capable of performing highly accurate semantic interpretation of CNL specifications. In terms of acquiring the domain vocabulary from a glossary and processing the rules of a specification the approach performed excellently. In addition, the extraction of vocabulary from unrestricted text performed better than many approaches and around the same level as the best of the approaches that could be compared using the comparison framework we developed. The one aspect that the prototype does not achieve the desired results is in the area of run-time performance as it appears there are a number of factors preventing it from being usable in real-world applications for the time being.

In the final sections of this thesis we reiterate our solution’s relationship to those surveyed earlier in the thesis, look to the future this work, and finally conclude this dissertation.

7.1 Comparison to Surveyed Approaches

In Chapter 3 we presented a survey of tools for the creation of formal models from natural language specifications. For comparison purposes we reproduce the table as Table 7.1 with the inclusion of CLUE4SBVR (highlighted in blue).

We then filtered out approaches to determine which tools were suitable for use by business and domain experts, rather than just technical experts. For this we applied several criteria, including: high abstraction level, natural and intuitive reading and writing, limited or no technical notations, ability to handle static and dynamic aspects of specifications, capacity to handle modalities, and not requiring additional inputs that should be generated by the tool.
This filtering left us with a short list (again reproduced here as Table 7.2 with our approach included) with approaches at opposite ends of the spectrum: IE approaches that are fully automated, but have limited error/inconsistency detection, produce only initial models that must be manually refined, and are more suited to an organisation looking to quickly gain insights into their documentation; and Formalist approaches that require extensive manual effort, but provide a formalised approach and result in more complete, and higher quality models.

With the inclusion of CLUE4SBVR this gap has been closed. In the middle now lies our Naturalist approach, which attempts to strike a compromise between the unrestricted and restricted texts to allow the use of existing documentation while using deep parsing techniques to provide high-quality feedback on erroneous, ambiguous, or inconsistent statements. It is aimed at actively supporting business users in formalising (possibly existing) specifications into a form more suited to integration in the software development process. While requiring more initial effort on the part of the user than the IE approaches, the possibility of identifying issues in existing documentation can help to focus the attention of the user to particular problems rather than requiring them to manually read and formalise everything themselves, which is what would be necessary with the Formalist approaches. By providing this middle ground, we open up new options for users and organisations looking to take control and begin formalising their natural language specifications themselves.
### Table 7.1: Comparison of Related Work including CLUE4SBVR

<table>
<thead>
<tr>
<th>Approach</th>
<th>CNL Class</th>
<th>Rule Types</th>
<th>DoA Analysis Type</th>
<th>Complete Inputs</th>
<th>Outputs</th>
<th>LoA</th>
<th>Adaptability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Formalist</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[TC06]</td>
<td>$P^5 E_{\leq 2m}^{4} N^3 N^3 w S^4$</td>
<td>S+O</td>
<td>Semi</td>
<td>S</td>
<td>✓</td>
<td>G*</td>
<td>SBVR, Prolog</td>
</tr>
<tr>
<td>[RPH08]</td>
<td>$P^5 E_{\leq 2m}^{4} N^3 N^3 w S^4$</td>
<td>S+O</td>
<td>Semi</td>
<td>S</td>
<td>✓</td>
<td>G*</td>
<td>UML (Class, Activity, Sequence)</td>
</tr>
<tr>
<td>[NSSS+10]</td>
<td>$P^5 E_{\leq 2m}^{4} N^3 N^3 w S^4$</td>
<td>S+O</td>
<td>Semi</td>
<td>S</td>
<td>✓</td>
<td>G*</td>
<td>SBVR, UML/OCL (Class)</td>
</tr>
<tr>
<td>[SPI10]</td>
<td>$P^5 E_{\leq 2m}^{4} N^3 N^3 w S^4$</td>
<td>S</td>
<td>Semi</td>
<td>S</td>
<td>✓</td>
<td>G*</td>
<td>sSBVRMM, UML (Class), BPMN</td>
</tr>
<tr>
<td>[NCGEI3]</td>
<td>$P^5 E_{\leq 2m}^{4} N^3 N^3 w S^4$</td>
<td>S+O</td>
<td>Semi</td>
<td>S</td>
<td>✓</td>
<td>G*</td>
<td>SBVR</td>
</tr>
<tr>
<td>[RCV12]</td>
<td>$P^5 E_{\leq 2m}^{4} N^3 N^3 w S^4$</td>
<td>S+O</td>
<td>Semi</td>
<td>W</td>
<td>✓</td>
<td>G, T*</td>
<td>ECA-rules</td>
</tr>
<tr>
<td>[Lin06]</td>
<td>$P^5 E_{\leq 2m}^{4} N^3 N^3 w S^4$</td>
<td>O*</td>
<td>Semi</td>
<td>W</td>
<td>✓</td>
<td>DM*</td>
<td>OCL</td>
</tr>
<tr>
<td>[SP14]</td>
<td>$P^5 E_{\leq 2m}^{4} N^3 N^3 w S^4$</td>
<td>S+O</td>
<td>Semi</td>
<td>W</td>
<td>✓</td>
<td>G*,T*</td>
<td>OWL2</td>
</tr>
<tr>
<td>[SNPS12]</td>
<td>$P^5 E_{\leq 2m}^{4} N^3 N^3 w S^4$</td>
<td>S</td>
<td>Semi</td>
<td>S</td>
<td>✓</td>
<td>G*</td>
<td>SBVR, OWL2/SWRL/SPARQL</td>
</tr>
<tr>
<td>[Sch08; Sch10]</td>
<td>$P^5 E_{\leq 2m}^{4} N^3 N^3 w S^4$</td>
<td>S</td>
<td>Semi</td>
<td>S+</td>
<td>✓</td>
<td>L*</td>
<td>TPTP</td>
</tr>
<tr>
<td>[FKK08]</td>
<td>$P^4 E_{\leq 2m}^{4} N^3 N^3 w S^4$</td>
<td>S</td>
<td>Semi</td>
<td>S+</td>
<td>✓</td>
<td>L*</td>
<td>DRS</td>
</tr>
<tr>
<td>[Mic96]</td>
<td>$P^4 E_{\leq 2m}^{4} N^3 N^3 w S^4$</td>
<td>S</td>
<td>Auto</td>
<td>D</td>
<td>✓</td>
<td>×</td>
<td>OMIT Object Diagram</td>
</tr>
</tbody>
</table>

Continued on next page.
<table>
<thead>
<tr>
<th>Approach</th>
<th>CNL Class</th>
<th>Rule Types</th>
<th>DoA Type</th>
<th>Analysis Type</th>
<th>Complete</th>
<th>Inputs</th>
<th>Outputs</th>
<th>LoA</th>
<th>Adaptability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Naturalist</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLUE4SBVR</td>
<td>$P_3E_1^4N_4^4N_3^4S_2$</td>
<td>S+O</td>
<td>Semi</td>
<td>D</td>
<td>✓</td>
<td>G</td>
<td>SBVR</td>
<td>CIM</td>
<td>D+L</td>
</tr>
<tr>
<td>KAB09b</td>
<td>$P_3E_1^4N_4^4N_3^4S_2$</td>
<td>S+O</td>
<td>Auto</td>
<td>D</td>
<td>✓</td>
<td>L*</td>
<td>SBVR, UML (Class)</td>
<td>CIM</td>
<td>D+L</td>
</tr>
<tr>
<td>CMHT10</td>
<td>$P_3E_2^4N_4^4N_3^4S_2$</td>
<td>S</td>
<td>Semi</td>
<td>S+</td>
<td>✓</td>
<td>DM*</td>
<td>Prolog</td>
<td>CIM</td>
<td>×</td>
</tr>
<tr>
<td>AG06</td>
<td>$P_3E_1^4N_4^4N_3^4S_3$</td>
<td>S+O</td>
<td>Semi</td>
<td>S+</td>
<td>✓</td>
<td>G*, T*</td>
<td>Various static and dynamic</td>
<td>CIM</td>
<td>D+L</td>
</tr>
<tr>
<td>GH09</td>
<td>$P_3E_1^4N_4^4N_3^4S_4$</td>
<td>O</td>
<td>Semi</td>
<td>S</td>
<td>✓</td>
<td>×</td>
<td>Live Sequence Charts (LSCs)</td>
<td>PIM</td>
<td>×</td>
</tr>
<tr>
<td><strong>Information Extraction</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AU06</td>
<td>$P_1E_1^3S_2^2N_5^5N_5^5S_3$</td>
<td>S+O</td>
<td>Auto</td>
<td>S</td>
<td>×</td>
<td>×</td>
<td>UML (Class)</td>
<td>PIM</td>
<td>D</td>
</tr>
<tr>
<td>BBL10</td>
<td>$P_1E_2^4S_2^2N_5^5N_5^5S_3$</td>
<td>S+O</td>
<td>Auto</td>
<td>S+</td>
<td>×</td>
<td>DM*</td>
<td>OCL</td>
<td>CIM</td>
<td>D</td>
</tr>
<tr>
<td>ABB11</td>
<td>$P_1E_2^4S_2^2N_5^5N_5^5S_3$</td>
<td>S</td>
<td>Auto</td>
<td>S+</td>
<td>×</td>
<td>×</td>
<td>UML (Class)</td>
<td>PIM</td>
<td>D</td>
</tr>
<tr>
<td>BLB11</td>
<td>$P_1E_2^4S_2^2N_5^5N_5^5S_3$</td>
<td>S+O</td>
<td>Auto</td>
<td>S+</td>
<td>×</td>
<td>DM*</td>
<td>SBVR</td>
<td>CIM</td>
<td>D</td>
</tr>
<tr>
<td>FE12</td>
<td>$P_1E_2^4S_2^2N_5^5N_5^5S_3$</td>
<td>S+O</td>
<td>Auto</td>
<td>S+</td>
<td>×</td>
<td>G</td>
<td>SBVR</td>
<td>CIM</td>
<td>D</td>
</tr>
<tr>
<td>AS13</td>
<td>$P_1E_2^4S_2^2N_5^5N_5^5S_3$</td>
<td>S+O</td>
<td>Auto</td>
<td>S</td>
<td>×</td>
<td>G, T*</td>
<td>RSL-IL</td>
<td>CIM</td>
<td>D+L</td>
</tr>
<tr>
<td>FMP11</td>
<td>$P_1E_2^4S_2^2N_5^5N_5^5S_3$</td>
<td>O</td>
<td>Auto</td>
<td>S+</td>
<td>×</td>
<td>×</td>
<td>World Model, BPMN</td>
<td>CIM</td>
<td>D+L</td>
</tr>
<tr>
<td>FKMS+07</td>
<td>$P_1E_2^4S_2^2N_5^5N_5^5S_3$</td>
<td>S+O</td>
<td>Semi</td>
<td>S+</td>
<td>×</td>
<td>×</td>
<td>KCPM, UML (Class, Activity, Statechart)</td>
<td>CIM</td>
<td>D</td>
</tr>
<tr>
<td>SDGG13, SDGG14</td>
<td>$P_1E_2^4S_2^2N_5^5N_5^5S_3$</td>
<td>O</td>
<td>Auto</td>
<td>D</td>
<td>×</td>
<td>DM*, T*</td>
<td>OWL/Reified Requirement Templates</td>
<td>PIM</td>
<td>D</td>
</tr>
</tbody>
</table>

*Continued on next page.*
Table 7.1 – continued.

<table>
<thead>
<tr>
<th>Approach</th>
<th>CNL Class</th>
<th>Rule Types</th>
<th>DoA</th>
<th>Analysis Type</th>
<th>Complete</th>
<th>Inputs</th>
<th>Outputs</th>
<th>LoA</th>
<th>Adaptability</th>
</tr>
</thead>
<tbody>
<tr>
<td>[LZC13]</td>
<td>$P^1 E^3 N^5 N_{w}^5 S^3$</td>
<td>S</td>
<td>Auto</td>
<td>S</td>
<td>×</td>
<td>×</td>
<td>UML (Class)</td>
<td>PIM</td>
<td>D+L</td>
</tr>
<tr>
<td>[SA14]</td>
<td>$P^1 E^3_{\leq 2} N^5 N_{w}^5 S^3$</td>
<td>S</td>
<td>Auto</td>
<td>S</td>
<td>×</td>
<td>×</td>
<td>UML (Class)</td>
<td>CIM</td>
<td>D</td>
</tr>
<tr>
<td>[SSA12]</td>
<td>$P^1 E^3 N^5 N_{w}^4 S^3$</td>
<td>S+O</td>
<td>Auto</td>
<td>S</td>
<td>×</td>
<td>G</td>
<td>UML (Class, Component)</td>
<td>PIM</td>
<td>D</td>
</tr>
<tr>
<td>[HG03]</td>
<td>$P^1 E^3 N^5 N_{w}^5 S^3$</td>
<td>S</td>
<td>Semi</td>
<td>S</td>
<td>×</td>
<td>×</td>
<td>UML (Class)</td>
<td>CIM</td>
<td>D</td>
</tr>
<tr>
<td>[HG03]</td>
<td>$P^1 E^3 N^5 N_{w}^5 S^3$</td>
<td>S</td>
<td>Auto</td>
<td>S+</td>
<td>×</td>
<td>×</td>
<td>XI World Model, UML (Class)</td>
<td>CIM</td>
<td>D</td>
</tr>
<tr>
<td>[IO06]</td>
<td>$P^1 E^4 N^5 N_{w}^5 S^3$</td>
<td>S+O</td>
<td>Auto</td>
<td>S</td>
<td>×</td>
<td>×</td>
<td>UML-like (Class, Activity)</td>
<td>CIM</td>
<td>D</td>
</tr>
<tr>
<td>[EVR11]</td>
<td>$P^1 E^3_{\leq 2} N^5 N_{w}^5 S^3$</td>
<td>S</td>
<td>Auto</td>
<td>S</td>
<td>×</td>
<td>×</td>
<td>ER</td>
<td>CIM</td>
<td>D</td>
</tr>
<tr>
<td>[LKM10]</td>
<td>$P^1 E^2 N^5 N_{w}^5 S^3$</td>
<td>S</td>
<td>Auto</td>
<td>S</td>
<td>×</td>
<td>G</td>
<td>ER</td>
<td>PIM</td>
<td>D+L</td>
</tr>
<tr>
<td>[CS08]</td>
<td>$P^1 E^3 N^4 N_{w}^4 S^3$</td>
<td>O</td>
<td>Semi</td>
<td>D</td>
<td>×</td>
<td>L*, DM*, T</td>
<td>CSP process algebra</td>
<td>PIM</td>
<td>×</td>
</tr>
<tr>
<td>[YBL10]</td>
<td>$P^1 E^3 N^4 N_{w}^4 S^2$</td>
<td>O</td>
<td>Auto</td>
<td>S</td>
<td>×</td>
<td>×</td>
<td>UCMeta (intermediate model), UML (Class, Activity, Sequence)</td>
<td>PIM</td>
<td>×</td>
</tr>
</tbody>
</table>

Legend:
- **Rule Types:** Structural (S), Operational (O);
- **Analysis Type:** Shallow (S), Shallow+ (S+), Deep (D), Wizard (W);
- **Inputs:** glossary (G), lexicon (L), domain model (DM), rules or templates (T), special notation (*);
- **Adaptability:** document structure (D), language (L);

- a Has the specified features only if appropriate patterns/templates have been provided;
- b Restricted permission rules only;
- c Various via transformations from logical representation;
Table 7.2: Short-list of Related Work including CLUE4SBVR

<table>
<thead>
<tr>
<th>CNL Class</th>
<th>DoA</th>
<th>Analysis Type</th>
<th>Complete Adaptability</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1 E2 N5 W3 S3</td>
<td>Auto</td>
<td>S</td>
<td>×</td>
<td>DUL-like (Class, Activity)</td>
</tr>
<tr>
<td>I006</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1 E4 N5 W3 S3</td>
<td>Auto</td>
<td>S+</td>
<td>×</td>
<td>D</td>
</tr>
<tr>
<td>FE12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1 E3 N5 W3 S3</td>
<td>Auto</td>
<td>S</td>
<td>✓</td>
<td>D</td>
</tr>
<tr>
<td>CLUE4SBVR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1 E2 N4 W4 S2</td>
<td>Semi</td>
<td>D</td>
<td>✓</td>
<td>D+L</td>
</tr>
<tr>
<td>TC06</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5 E4 N3 W3 S4</td>
<td>Semi</td>
<td>S</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>NSSS+10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5 E3 N3 W3 S4</td>
<td>Semi</td>
<td>S</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>NCGE13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5 E2 N3 W3 S4</td>
<td>Semi</td>
<td>S</td>
<td>✓</td>
<td>×</td>
</tr>
</tbody>
</table>

7.2 Limitations and Future Work

While we have presented a promising approach to formalising natural language specifications as models in this thesis, there are several issues open for investigation.

Prototype Tool to Real-World Tool The current state of the prototype CLUE4SBVR is not yet suitable for use by domain experts in real-world applications. This is for several reasons, including:

- it is heavily dependent on operating within the GATE development environment;
- the results generated by the configurator, particularly when errors occur, are not understandable by domain users as they simply report the constraint violations;
- the alternatives produced due to ambiguities are currently only reported, the user cannot select the desired interpretation;
- the results of vocabulary acquisition, particularly from unrestricted text, need to be displayed better;
- the capacity for the approach to scale to large sets of specifications and associated material needs to be ensured; and
- the run-time performance of parsing process needs to be improved significantly.
Section 7.2. Limitations and Future Work

The majority of these issues, except the last two, can be solved by taking the time to develop a proper set of user interfaces for the tool. For the scalability of the approach, further evaluations need to be performed and features of the tool need to be extended, for example, to include a repository of requirements and associated documents rather than the limited ability of the prototype to process one document at a time. The issue of run-time performance is discussed below.

Run-time Performance As we discovered during the evaluation of CLUE4SBVR, the run-time performance of our approach, particularly on long complex sentences, is inadequate for real-world use. We noted, in Section 6.4, that there were several causes of this poor performance such as: (1) the execution of the configuration process only after complete syntactic analysis is performed, (2) the inadequate handling of polysemous lexical entries when it comes to configuration, (3) the sequential processing of alternative suggestions, (4) the configuration of too many alternatives, particularly the excessive number of obviously incorrect SBVR models sent to configuration (i.e. those that fail quickly), and (5) the overhead involved in calling the configurator caused by the constant serialisation/deserialisation of the models.

The solutions to these problems range from the purely technical to those requiring additional research. Examples of the former include modifying the implementation to execute the configurator incrementally alongside the syntactic analysis, adjusting the constraints generated for polysemous lexical entries, executing multiple configurations in parallel, and solving the serialisation/deserialisation overhead issue.

An issue that will require additional research is that of reducing the number of suggestions that need to be sent to the configuration process. The complexity involved with this aspect revolves around determining ways of reliably pruning the suggestions without over pruning and causing no interpretation to be found. This is also a problem with traditional NLP (and search problems in general); therefore, we will investigate solutions developed for traditional NLP and hopefully adapt them to our approach.

Another interesting research direction will be the integration of the syntactic and semantic analysis so that they can be performed in a joint fashion. The current separation of syntactic
and semantic analysis in our approach is largely conceptual but also pragmatic due to the use of technologies such as GATE, which provides the framework for the tool in Java, and COCOA, which is the configurator written in Smalltalk. Therefore, integrating the two in some fashion would most likely result in vastly improved performance. One possible avenue is to investigate an integration approach similar to that of [BDJM14], in which linked hypergraphs (some representing syntax while others represent semantics) are incrementally transformed in conjunction with one another. Since we utilise graph-based models extensively for the semantic aspect, it may be possible to integrate the syntactic aspect into the models. We could then perform configuration on the linked models without explicitly defining all of the allowed graph transformations manually, which is what is required in [BDJM14].

**User Evaluation** The evaluation of CLUE4SBVR presented in this thesis focused on a performance evaluation of the individual components. To truly prove the efficacy of our approach, we need to perform an adequacy evaluation on a group of real-world users. This is heavily dependant on the previous challenge of moving the tool beyond that of a simple prototype. This is important since the quality and maturity of MDE tools has traditionally been a hindrance to the adoption of MDE [MD08] (although, more recently, the adoption of MDE appears to be improving [BCW12]). Moreover, evaluating a substandard tool would not provide representative results on whether or not the approach is truly effective. Therefore, once a more mature tool has been developed we will be able to pursue investigations of its application in real-world contexts.

**Improved Vocabulary Acquisition from Unrestricted Text** Another component in which there is room for improvement is the acquisition of vocabulary from unrestricted text. In particular, the filtering of terms should be improved. One avenue of investigation is the incorporation of pruning based on frequency. While other approaches do this, it is often based on simple assumptions and threshold; however, there are two aspects that should be considered.

The first is whether or not different thresholds should be used for concepts and relations—and possibly finer grained categories such as object types vs. individual concepts. This is based on the observation that, while relevant relations may only be mentioned once in a short
text, concepts may still be mentioned several times as arguments to multiple relations. This difference is not considered in work that uses frequency-based filtering when creating models from text—a single threshold is usually considered for all types of terms.

The second is to determine what the threshold(s) should be. In addition, should they be static or dynamic (potentially increasing with the size of the text), or different for the different thresholds. For this, empirical evaluations of different filtering strategies need to be investigated.

In addition, would be interesting to investigate if the parsing approach developed in this thesis could be used to perform vocabulary suggestion. Since the closed class words must be defined in the lexicon a priori, various parts of sentences can already be identified leaving the rest as potential candidates for vocabulary. Potentially, by bootstrapping the parser with existing lexical resources (e.g. WordNet, VerbNet, FrameNet) similar to [SM05], it may be possible to more accurately identify candidate vocabulary entries in the text. Moreover, this potentially goes part way to achieving the final goal for future work, below.

**Generalisation to Unrestricted Natural Language** While the general framework based on Cognitive Grammar is as applicable to unrestricted natural language as it is to controlled languages, there are some limitations due to the tailoring more towards the controlled language end of the spectrum. For example, the knowledge-requirement is difficult to fulfil for unrestricted language if it must be defined manually in specific glossaries. However, the use of generally available resources and the prevalence of ontologies (both general and domain specific) may overcome that restriction.

Furthermore, the selectional restrictions enforced by the configurator are strict, whereas it is known that they can be violated in natural language and still produce understandable utterances [All95]. A simple solution could be similar to that of [Hol93], where alternative reasoning is performed if the primary reasoning is unable to find a satisfactory solution. In this case, if the basic configuration process failed then it could trigger reasoning modules for metaphor and analogy, for example, to try to make sense of an expression.

Finally, aspects of pragmatics are not handled due to the current focus of application. For example, although the configuration can support any degree of context by expanding the
model that is being configured, there is no explicit discourse model to control how that context is expanded. Integrating such pragmatic considerations will be essential to expand the scope of our approach beyond that of highly precise, but flexible and natural, controlled languages.
Bibliography


<table>
<thead>
<tr>
<th>Reference</th>
<th>Title and Authors</th>
</tr>
</thead>
</table>


<table>
<thead>
<tr>
<th>Reference</th>
<th>Authors</th>
<th>Title</th>
<th>In:</th>
<th>Volume</th>
<th>Series</th>
<th>Publisher</th>
<th>ISBN</th>
<th>DOI</th>
<th>URL</th>
</tr>
</thead>
</table>
Bibliography


BIBLIOGRAPHY


Bibliography


[SD00] H Saiedian and R Dale. “Requirements engineering: making the connection between the software developer and customer”. In: Information and Software Technology 42.6 (2000), pp. 419–428. ISSN: 0950-5849. DOI: 10.1016/S0950-5849(99)00101-9


Bibliography


Appendix A: PENS Classification Framework

This appendix contains an abridged description of the PENS classification framework of Kuhn [Kuh14] for reference purposes. It is used in the survey of tools for generating models from natural language requirements (Chapter 3).

The class of a CNL can be considered in terms of the PENS scheme proposed by Kuhn [Kuh14]. This is a method of classifying the nature of a CNL using four dimensions: precision, expressiveness, naturalness, and simplicity. Each dimension is separated into 5 distinct categories along a continuum, with formal languages (e.g. propositional logic at $P^5, E^1, N^1, S^5$) at one end and natural languages (e.g. English at $P^1, E^5, N^5, S^1$) at the other. The classes for each dimension are defined in the following.

**Precision** is the degree to which the symbols of the language can be translated into its meaning with or without ambiguity and defines its classes as:

- $P^1$ *Imprecise languages*, such as all natural languages, where more complex sentences are almost always ambiguous.

- $P^2$ *Less imprecise languages*, typically reduce or remove a large portion of the ambiguity of their parent natural language but do not have any formal (e.g. mathematical) foundation.

- $P^3$ *Reliably interpretable languages* have restricted (but not formally defined) syntax and have some logical or formal foundation to their semantics; however, the mapping between the two is incomplete and requires the resolution of ambiguity.
Appendix A: PENS Classification Framework

\( P^4 \) Deterministically interpretable languages have formally defined syntax (possibly using a formal grammar) and can be deterministically parsed into a (possibly underspecified) logical formalism.

\( P^5 \) Languages with fixed semantics are formally specified in terms of both syntax and semantics (without the need for heuristics or external resources), and each text has exactly one meaning.

**Expressiveness** is the range of propositions supported by the language and defines its classes based on a minimal set of propositions:

\( E^1 \) Inexpressive languages lack universal quantification, restrict relations to an arity of 1 (i.e. unary relations), or both

\( E^2 \) Languages with low expressiveness support both universal quantification and relations with an arity > 1 (i.e. at least binary relations, but does not have to be ternary or relations of any arity)

\( E^3 \) Languages with medium expressiveness support general structures (e.g. if-then statements) and some form of negation in addition to the features of \( E^2 \) languages

\( E^4 \) Languages with high expressiveness support second-order logic (i.e. universal quantification over concepts and relations) in addition to the features of \( E^3 \) languages

\( E^5 \) Languages with maximal expressiveness can ‘…express anything that can be communicated between two human beings.’ [Kuh14]

**Naturalness** is the degree to which the language is readable and understandable with respect to natural language:

\( N^1 \) Unnatural languages are languages that do not appear natural with extensive use of symbols, brackets, and unnatural keywords.

\( N^2 \) Languages with dominant unnatural elements may make extensive use of natural language words or phrases but are overridden by unnatural elements (e.g. symbols,
bracketing), unnatural syntax, or unnatural semantics, so that it is difficult to intuitively understand statements in the language.

\( N^3 \) Languages with dominant natural elements can be intuitively understood to a large degree, but the presence of unnatural elements means the sentences are not necessarily considered well-formed natural sentences.

\( N^4 \) Languages with natural sentences consist of sentences that are considered well-formed natural sentences and their meaning can be intuitively understood; however, there may be some minor exceptions or use some means for clarification (e.g. text styling, indentation, hyphenation, capitalisation), and full texts (as opposed to individual sentences) typically do not flow like the text of a natural language equivalent.

\( N^5 \) Languages with natural texts appear completely natural in terms of syntax, semantics, sentences, and full texts.

**Simplicity** measures the complexity of a description of the language (including a complete description of the syntax and semantics using acceptable linguistic and mathematical notations) based on the number of pages that would be needed to document such a description (if one is even possible):

\( S^1 \) Very complex languages cannot be described in such a comprehensive manner

\( S^2 \) Languages without exhaustive descriptions are considerably simpler than natural languages (\( S^1 \)) but still cannot be described in such a comprehensive manner; rather they are often described as restrictions on a natural language.

\( S^3 \) Languages with lengthy descriptions can be defined in a comprehensive manner in more than 10 pages

\( S^4 \) Languages with short descriptions can be defined using between 2 and 10 pages (inclusive).

\( S^5 \) Languages with very short descriptions can be defined in a single page.
Appendix B: Configuration Art

This appendix records a few of the interesting results produced by the configuration process when they were visualised using graphviz.
Each car movement has exactly one sending branch.

TO INFINITY AND BEYOND!
Each car movement has exactly one sending branch.
Each car movement has exactly one sending branch.
Each car movement has exactly one sending branch.
The “waterfall” model.