Automated Software Development and Model Generation by means of Syntactic and Semantic Analysis

Mark Meiklejohn

Department of Computer & Information Sciences

University of Strathclyde

PhD

September - 2014

Abstract

Software development is a global activity and the development of a software system starts from some requirement that describes the problem domain. These requirements need to be communicated so that the software system can be fully engineered and in the majority of cases the communication of software requirements typically take the form of written text, which is difficult to transform into a model of the software system and consumes an inordinate amount of project effort.

This thesis proposes and evaluates a fully automated analysis and model creation technique that exploits the syntactic and semantic information contained within an English natural language requirements specification to construct a Unified Modelling Language (UML) model of the software requirements. The thesis provides a detailed description of the related literature, a thorough description of the Common Semantic Model (CSM) and Syntactic Analysis Model (SAM) models, and the results of a qualitative and comparative evaluation given realistic requirement specifications and ideal models.

The research findings confirm that the CSM and SAM models can identify: classes, relationships, multiplicities, operations, parameters and attributes all from the written natural language requirements specification which is subsequently transformed into a UML model. Furthermore, this transformation is undertaken without the need of manual intervention or manipulation of the requirements specification.

Acknowledgements

I am thrilled that I have completed this thesis and brought together my largest body of work to date - at times this PhD has been very testing and I cannot believe I have come so far and grown so much, it has been a true companion and I have enjoyed this enormous journey.

I owe a great gratitude to many people that have shown and given so much support, advice and pushed me beyond all my expectations, my supervisors Dr Marc Roper and Dr Murray Wood. I would also like to give thanks to all the Computer and Informational Sciences staff and also to my fellow PhD colleagues, Sukumar Letchmunan, Inah Omoronyia and Konstantinos Liaskos for their advice, friendship and support.

Lastly, but in no way least, I dedicate this thesis to my wife, Leigh Meiklejohn, without her eternal patience, her love and commitment none of this would have been possible. I also want to send all my love to my children Caitlin, Olivia, Bean and Ethan; my friends and family that have also given me so much support through my PhD journey.

Mark Meiklejohn

September 2014

Contents

List of Figures

List of tables

Chapter 1
Introduction

1.1 Overview

The common approach to object-oriented design is a manual language analysis of the software requirements specification (SRS) typically involving domain experts to identify and create a model that represents the problem domain. There are a variety of standard methodologies used to assist the analysis process such as Noun Phrase analysis [**Abo85]**, Use Case driven **[JCJ+92]**, Common Class Patterns **[RBP91, Bah99],** and CRC cards **[WW89, WWW90]**. In practice, it is a combination of these methodologies that are used to complete the analysis. Classically this involves identifying nouns, verbs and their interrelationships, where nouns are considered good candidates for classes or attributes, verbs indicate relationships and operations that are associated with classes.

Software development is a human intensive activity with requirements analysis and preliminary design consuming 55% of a project's total effort **[NIST02]**. It is considered that this initial effort could be better spent building flexible, maintainable and reusable solutions aided by automated analysis and design.

Up till now research has been seen to apply either partially or fully automated techniques in the pursuit of software automation with varying levels of success. Most require manipulation of the language used in the specification, restriction of the sentence structure, introduction of controlled grammars, and / or the involvement of the designer during the detection process. The manual intervention in these cases only serves to negate the potential benefits that automated analysis and design aim to deliver.

This thesis proposes that the automated software analysis and the creation of a preliminary model from natural language requirements specifications without the need of manual intervention or requirements specification manipulation is a means to reduce the effort of the initial software development phases. It is therefore considered to what extent can the structural and semantic information contained within a natural language requirements specification contribute to a better preliminary design and be derived from the unrestricted use of semantic and syntactic information?

1.2 Approach & Methodology

The thesis starts by thoroughly reviewing the related literature in the key areas of both fully and semi-automated techniques. It starts with an overview of traditional requirement analysis and modelling techniques prior to focusing on these automated techniques. The main body of the review is separated into two key stages that discuss both semi and fully automated works in their chronological ordering. The review identifies that that there is a very little difference between the related works and work presented here in this thesis. For the majority of related works there is some means of syntactic analysis, but mainly in the context of extracting key word groups such as Nouns, Verbs and Adjectives in a standalone context. There is also consideration of semantic information as well. However, the connection between both syntactic and semantic information within related works is limited.

In contrast to related works, and the key differentiator, is that this thesis aims to develop the connection between word semantic analysis, and syntactic analysis. This has resulted in the creation of a prototype system, the 'Automated Software Architect' (ASA), which is a domain independent approach with no requirement for manual intervention or specification manipulation.

The prototype implementation features the 'Common Semantic Model' (CSM) and 'Syntactic Analysis Model' (SAM) techniques that are discussed in fuller detail within this thesis. The ASA is evaluated in the context of both the manual analysis and its most closely related works as identified in the evaluation **[Har00, Mic96]**.

1.3 Contributions

The work presented in this thesis makes the following contributions:

- A means to automatically create a conceptual UML Class model from unrestricted/unmodified natural language requirements specifications; identifying common features such as Classes, Relationships, Relationship Multiplicities, Operations, Operation Parameters, Operation Placement and Attributes

- Provision of a semantic and syntactic analysis model that is independent from the need for manual intervention, configuration or problem domain

- Decision traceability links identifying in the sentence where within the specification features of the class model have been identified
- Creation of flexible, maintainable and reusable design structures from semantic and syntactic analysis

The techniques employed are designed not to replace the manual development/analysis process, but are to be used as a means to reduce cognitive effort through automated analysis and conceptual model generation

1.4 Result Snapshot

The ASA is evaluated using the key measures of recall and precision. The evaluation investigates the ASA's performance in the context of the ideal model and most closely related works. Overall the ASA performs relatively well on its own with an average recall rate of 73% and precision of 60% and in the context of its most closely related works the ASA has an F-Measure of 67% in comparison to CM-Builder with an F-Measure of 77% and NL-OOPS with 62%. The key strengths of the ASA is its domain independence, utilisation of free form natural language requirements specifications, fully automated analysis aided by both CSM & SAM models and no user intervention. The key weaknesses of the ASA is the creation of incorrect class candidates, missing a minority of candidates that are actually contained within the specifications and poor performance when detecting candidate relationships.

Analysis of the weaknesses has identified that context, ambiguity, missing requirements, domain knowledge and specification noise exacerbate the situation. Future works, addressed in the final chapter, aim to deliver strategies as a means to resolve the weaknesses highlighted with the key requirement of not imposing any additional effort onto the user.

1.5 Thesis Structure

Chapter 2 Approaches to Automation: This chapter presents a constructive review of the related literature towards fully/semi-automated software modelling techniques identified from natural language requirements specifications. It first presents a view of industry standard software design methodologies and also identifies the allocation of effort over the last three decades; this is followed by an overview of the similarly standardised manual analysis techniques towards requirement modelling. With the scene set, the chapter proceeds to discusses both semi and fully automated requirement analysis techniques in the context of both controlled and uncontrolled natural language requirements specification documents; it identifies the key strengths and weaknesses of each piece of related work; what each related work identifies in terms of Classes, Relationships, Attributes, Operation Parameters, Operations, Relationship, Multiplicities and how those related works are evaluated. The chapter concludes by identifying the key research motivations.

Chapter 3 Techniques towards Automation: This chapter discusses the 'Common Semantic Model' (CSM) and 'Syntactic Analysis Model' (SAM) used to address the key findings and limitations as identified in the literature review. It presents the key syntactic features of the natural language and identifies what these mean in the context of UML modelling. The chapter proceeds to discuss how both the CSM and SAM models are used to identify, extract and manage the information contained within a free-form and unrestricted natural language requirements specifications and decision-making model behind Class, Relationship, Attribute, Operation, Parameter and Multiplicity detection techniques. Finally, the chapter concludes with a discussion of the key software requirements specification issues in the context of automated software development, its implementation and a detailed view the architecture.

Chapter 4 Evaluation: This chapter discusses the methodology of how the approach towards automated requirements analysis and design performs through the analysis measures of Recall, Precision and Over-Specification. The key motivation of this evaluation is to identify how well the approach performs in the context of both Class and Relationship detection of free-form natural language requirements in the context of the ideal model. The chapter concludes with a comparative evaluation of the most closely related works of CM-Builder **[Har00]** and NL-OOPS **[Mic96]**.

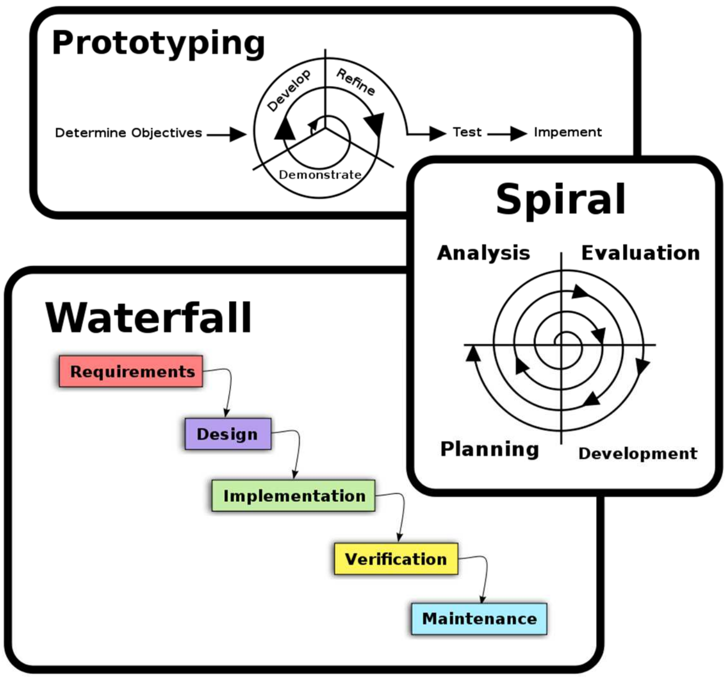
Chapter 5 Conclusion and Future Works: This chapter presents a summary of the work carried out in this research. The key limitations of the semantic and syntactic analysis approach of free-form natural language analysis for conceptual UML model design are presented and complementing research avenues are also identified and discussed within this chapter.

Chapter 2
Approaches to Automation

2.1 Introduction

This chapter reviews the related work in the field of automated software development in the context of natural language analysis and model generation.

The development of software has many well-defined processes: being either an iterative or linear process that drives the development of the software forward.



(Image Source: Wikipedia)

Figure 2.1-1 Software Development Frameworks

The most common frameworks are waterfall, prototyping and spiral, which have their own advantages and disadvantages. These frameworks can allow the successful delivery of a software product and all have common features such as requirements analysis, design, development and testing. Therefore, no matter what framework is utilised for what purpose or what benefit it brings, there is still the hurdle of manually analysing and modelling the customer requirements in an efficient and effective manner that will identify all of the relevant features of the described software system.

In the majority of cases, the requirements document is written in natural language text, but other techniques are possible such as requirements specification languages. Moreover, natural language requirement documents require analysis and transformation into a model prior to its implementation; this process consumes the majority of project effort **[NIST02]**.

The distribution of project effort has changed through time as more and more focus has been given to the initial phases of the software development lifecycle. This movement of effort has been the key goal to reducing the errors introduced because of poor software planning, but it comes with a substantial cost. Table 2.1-1 details the allocation of project effort over the last four decades.

Table 2.1-1 Allocation of Effort [NIST02]

	Requirements Analysis	Preliminary Design	Detailed Design	Coding & Unit Testing	Integration & Test	System Test
1960s – 1970s	10%			80%	10%	
1980s	20%		60%		20%	
1990s	40%	30%		30%		

Therefore, the overall aim of the related works discussed is to reduce this allocation of effort, whilst preserving/enhancing the effectiveness and efficiency of the requirements analysis and preliminary design process.

2.2 Requirements Analysis and Modelling Techniques

A critical activity in the creation of software is the capture of software requirements. Arguably, this is the most important activity in the software development process. For most software systems of any size, the requirements are captured in a natural language written document, but can also be in other forms. This can range from a few simple paragraphs to a complex document detailing information regarding relevant stakeholders, functional and non-functional requirements.

The next challenge in the software development process is to create an initial design for the software system from this specification. This activity is fraught with potential problems due to issues such as the misunderstanding of requirements, lack of domain knowledge, analyst's bias, overlooking and / or missing requirements.

In an object-oriented modelling domain, there are long-standing guidelines based on analysis of the specification that can aid the designer in identifying a 'first-cut' software design. These include considering nouns as potential classes and verbs as candidate operations / methods.

Related research, Section 2.3, has seen this as an opportunity to apply automated natural language analysis techniques that simulate and aid the developer in constructing the initial design. To-date most of this work requires intervention through simplification of the natural language, restricting the sentence structure allowed, and / or also requiring the involvement of the designer before, during and after the analysis process. Even so, some of these techniques give automation a simpler problem to manage when detecting the relevant model features.

2.2.1 Natural Language to Object Oriented Models

All software requirements specification (SRS) documents require transformation into a software/class model prior to implementation. This is the cornerstone of object-oriented system design and sets the foundation upon which the state and behaviour of the system can be observed.

The traditional starting point of class modelling, once the customer requirements have been elicited, is to transform the requirements into a model of the system prior to its implementation. This was originally achieved through some defined methodology such as the object modelling technique (OMT) or Object Oriented Software Engineering (OOSE), both a precursor to Unified Modelling Language (UML) **[BJR00]**, which is utilised to extract and model the key components of the system. *"The fundamental assumption is that object-oriented thinking represents a more natural and intuitive way for people to reason about reality."* **[RBP91]**

Today, this model is usually defined in the UML, which offers a variety of different structural (Static) or behavioural (Dynamic) diagrams that define the overall view of the software system itself.

Once the model of the system has been generated, it is then possible to construct and implement the required software from this high-level plan. However, the analysis process and model construction consumes the majority of project effort **[NIST02]** and is the source of many defects **[Son09]**.

2.2.2 Manual Techniques for Class Modelling

There are varieties of different manual techniques employed when transforming a natural language SRS document into an initial model of the defined software system. The most popular are identified as the *Noun Phrase, Common Class Patterns, Use Cases and Class-Responsibility-Collaborators Cards*.

These approaches allow the analyst to capture all the relevant information from the SRS document, which is used to build an initial model including features such as classes, relationships, attributes and operations. The main points of each are outlined below.

The *Noun Phrase Approach* can be considered the mainstream approach towards model generation, where this technique help to identified the possibility of detecting design features directly from a natural language SRS specification [**Abo85]**. Additional approaches have refined this technique **[SP99, Boo94, Mac01, RBP91, RSB93, Bah99]** and in the context of Entity-Relationship Diagrams the grammatical features of the sentence can also aid the overall analysis and modelling process **[Che83]**.

The principle idea of noun phrase analysis requires the analyst to read through and then collate each of the noun phrases contained within the SRS document. Once every noun is identified it is then compiled into a list. Consideration of whether a noun contained within the list is a candidate class requires validation. Therefore, each potential class candidate is subsequently classified into three distinct groups defined as *Relevant, Irrelevant and Fuzzy* classes.

Relevant classes: these typically appear frequently within the SRS documents **[Mac01]** and their inclusion as a relevant item is confirmed by the analyst's knowledge of the problem domain and supporting material that may also be available during the classification phase.

Irrelevant classes: these do not have a statement of purpose contained within the SRS or one cannot be formalised and they are typically outside the problem domain. The inclusion of irrelevant classes within the compiled list is unlikely with an experienced analyst, but is considered a key problem for automated analysis.

Fuzzy classes: these are on the fringes of being relevant or irrelevant and are a direct result of there not being enough information contained within the SRS to make an informed decision for their inclusion within the final model.

The classifications of *Relevant, Irrelevant, and Fuzzy* are employed to identify what should and should not be included within the initial design and what candidates require further investigation.

Similar to the *Noun Phrase* approach, the *Common Class Patterns* (CCP) is based upon classification theory, which is utilised to extract class candidates from a set of pre-defined classifications. These are partitioned into useful classes so that they can be reasoned about more efficiently and candidates identified. There are two classification themes; one proposed that has 5 classifications **[Bah99]** and the other having 6 classification types **[RBP91]** see Table 2.2-1.

Table 2.2-1 Common Class Pattern Classifications

CCP Classes [Bah99]	**CCP Classes [RBP91]**
Concept	Physical
Event	Business
Organization	Logical
People	Application
Places	Computer
	Behavioural

Although these classifications provide a means to potentially identify the class candidates from an SRS document, the approach does not offer a reliable means to identify a complete set of class candidates for a given problem statement. It is still possible that information relevant to the overall analysis process may be lost due to bias, misunderstanding or may not even be present within the specification itself to start with.

In the *Use Case Driven* approach **[JCJ+92]**, each use-case defines an actor within the system, which is derived from the requirements specification. The approach can be defined by a set of graphical notations that depicts the behaviour of the system as it responds to requests from actors outside the system. In addition, each use case also has supplementary information which identifies its role/usage, interactions/relationships and whatever may be involved with. The use-cases themselves, along with this supplementary information, are used in the discovery of the candidate classes for the initial class model. Discovery of these use-cases utilises a similar technique to that of the *Noun Phrase* Approach.

Finally, *'Class-Responsibility-Collaborators'* CRC cards **[WW89, WWW90]** are typically used to help teach object-oriented design concepts, but now are a prominent feature in *'Extreme Programming'* (XP) practice as a design technique.

All of these approaches have their own advantages and disadvantages, which will not be discussed, as they are well documented, **[SP99, Boo94, Mac01, RBP91, RSB93, Bah99, WW89, WWW90, JCJ+92]**. Nonetheless, none of these approaches by themselves is a complete answer to the overall analysis and model generation process.

It is typically more realistic to utilise a combination of these techniques throughout the analysis and modelling process. In addition, *Noun Phrase, Use Case, CCP and CRC* techniques can also aid in the identification of relationships, attributes and operations, typically as a by-product of the class detection process. Utilisation of these techniques will aid the discovery of a comprehensive first-cut design, but only at the expense of considerable effort expended by the analyst.

2.3 Automated Approaches

The key principle of the automated analysis of a natural language SRS document is a simulation of these manual processes. Natural Language Processing (NLP) techniques have made it possible to consider these manual techniques in the context of automation. There are two key branches to software automation, those being either semi or fully automated techniques; although for the majority of these techniques, they still require some form of developer involvement or transformation to assist the overall analysis process

There are many approaches towards automating the initial phases of the software development lifecycle (SDLC) that all achieve the same goal of producing a model from the natural language specification. A state of the art review in the domain of automated requirements elicitation by Meth, Brhel and Meadche [MBM13] classifies potential related works in the domains of abstraction identification, requirements quality analysis, requirements identification and the most relevant requirements model generation.

The focus in this literature review is towards how the process achieves its goal rather than what is achieved by the process and Table 2.3-1 reviews the current state of the art of both fully and semi-automated approaches.

Table 2.3-1 Automated Analysis Outcomes

Reference	Class	Attr	Rel	Param	Op	Multi	Auto	Eval
MHH89	✓	✓	✓		✓			
FGR+93	✓		✓		✓			
GB94	✓							✓
BV95, BR96, BV97	✓	✓	✓			✓		
NR95	✓	✓	✓		✓		✓	
Mic96, MMZ02, MG02, KZM+04	✓	✓	✓				✓	✓
Mor97, JM00, JM00a	✓	✓	✓		✓			
AG97, AG99, GN02, AG06	✓	✓	✓	✓	✓			
SBB99	✓	✓			✓			
Bry00, LB02, LB02a, LB02b, LB02c, LB03, BLC+03	✓	✓	✓		✓			
OLR01	✓	✓	✓		✓			
Per02, PKS+05	✓	✓	✓		✓		✓	
Har00, HG02	✓	✓	✓			✓	✓	✓
ZZ03	✓	✓	✓			✓	✓	
LDP04, LDP05, LDP05a	✓	✓	✓		✓		✓	
IO05, IO06, OI06a	✓	✓	✓		✓		✓	✓
Kof05, Kof05a, Kof07, Kof07a, Kof08, Kof09	✓		✓					✓
BSC06, BCA06, BSM09	✓	✓	✓		✓			✓
PRM+07	✓	✓	✓		✓		✓	✓
CHK07	✓		✓		✓			
GT07, GK08								
DR08, DR09, DB09	✓	✓	✓		✓		✓	
VAD09	✓	✓	✓					✓
SOS08	✓		✓				✓	
SRC+07								✓

The majority of these related approaches identify classes (*class*), relationships (*rel*), attributes (*attr*) and operations (*op*), but a minority in addition to these also identify parameters (*param*) and multiplicities (*multi*). Nine out of twenty-three approaches could be considered as fully automated (*auto*) but some may include minimal developer involvement at either the start or end of the automated analysis process. Overall, few approaches demonstrate some form of formal evaluation (*eval*) related to their techniques.

There is a clear distinction between fully and semi-automated works. This distinction can be identified by how these approaches manage the detection process. Fully automated approaches have a set of generalised rules and or predefined knowledge bases that allow analysis on any specification type. Whereas, for semi-automated, there is a reliance on the analyst during the detection process either to specifically identify model features or build custom models.

The next major distinction is between controlled (user modified) and uncontrolled (original/as-is/unmodified) language analysis techniques. However, at the next level down the actual extraction technique such as rule based, knowledge based (KB) driven and others, the line between these starts to get a bit blurry as some approaches may be entirely focused on a rule-based approach, whereas others may utilise a combination of both.

The split between both controlled and uncontrolled language analysis approaches for the semi-automated techniques is relatively even see, Table 2.3-2. This result was unexpected as it was thought that it would be more weighted towards uncontrolled requirements specification documents because of the interaction required before, during and after the detection process.

Table 2.3-2 Semi-Automated (Controlled vs. Uncontrolled Language Analysis)

Reference	Semi-Automated (language)	
	Controlled	Uncontrolled
MHH89	-	✓
FGR+93	✓	-
GB94	-	✓
BV95, BV96, BV97	✓	-
Mor97, JM00, JM00a	✓	-
AG97, AG99, GN02, AG06	-	✓
SBB99	-	✓
Bry00, LB02, LB02a, LB02b, LB02c, LB03, BLC+03	-	✓
OLR01	-	✓
Kof05, Kof05a, Kof07, Kof08, Kof09	-	✓
BSC06, BCA06, BSM09	✓	-
CHK07	✓	-
GT07, GK08	✓	-
VAD09	✓	-

Conversely, the majority of fully automated approaches use the original version (uncontrolled) of the requirements specification, which was also unexpected, as it was thought that these techniques would require more language manipulation because of less developer interaction and more generalised techniques, see Table 2.3-3

Table 2.3-3 Fully-Automated (Controlled vs. Uncontrolled Language Analysis)

Reference	Fully-Automated (Language)	
	Controlled	Uncontrolled
NR95	✓	
Mic96, MMZ02, MG02, KZM+04	-	✓
Per02, PKS+05	-	✓
Har00, HG02	-	✓
ZZ03	-	✓
LDP04, LDP05, LDP05a	✓	-
IO05, IO06, OI06	-	✓
PRM+07	✓	-
DR08, DR09, DB09	✓	-
SOS08	✓	
SRC+07		✓

Semi-Automated

The review of the semi-automated works is split into two key sections: controlled and uncontrolled. Additional, groupings of related techniques is unsuitable as they either utilise a combination of those techniques or are solely dedicated to one method of approach. The following reviews are chronologically ordered.

Controlled Language:

Fantechi *et al* **[FGR+93]** discuss an interactive approach towards automated software development based on transforming natural language into a temporal logic (NL2ACTL). This is aided by interaction with the analyst whom will reduce/eliminate the ambiguities contained within the natural language specification. They are effectively modifying the specification, which may result in a loss of vital information. The process is iterative where automated transformations take place, which are subsequently validated by the analyst, and finally result in the development of a formal specification from an informal one.

The approach transforms simple natural language with only one clear interpretation into temporal logic, which is then subsequently transformed into an extended natural language description again with only one interpretation. This may involve splitting a sentence into separate parts to achieve this format.

For Example: *'It is always possible to insert a coin. After the coin is inserted it is possible to have a tea'*

Sentence: *'It is always possible to insert a coin'*

Extended Form: *'For all states there exists a computation path starting with the action 'coin''*

NL2ACTL Transformation: *AG <coin> true*

The construction of the NL2ACTL formulae is the basis of building a grammar, which embodies precisely the meaning without ambiguity. This grammar along with both domain and domain independent dictionaries are the foundations of the translation tool.

The domain dependent dictionary contains specific terms one would reasonably expect to be within a conceptual model or formal specification, which are user defined and identified from the specification. The domain independent dictionary manages common word sets that are typically not related to any specific domain, but have implications for logical operations such as pronouns, verbs and conjunctions. As a result, any missing information can be contained within these dictionaries and queried interactively with the user.

The interactive transformations create a set of entirely unambiguous specifications that can be used to derive testing criteria and support the actual implementation of the software system. Although this does come at a cost, these manual interactions can lead to a situation where the amount of effort expended in the process outweighs the benefits of the approach.

Burg and Van de Riet **[BV95, BV96, BV97]** propose a linguistically based object-modelling tool that can be used during the conceptual modelling phase to construct both static and dynamic models from natural language specifications, where the natural language is manually transformed into their conceptual prototyping language (CPL).

CPL has been specifically developed to be as close to natural language as possible thus allowing the requirements to be specified in a precise and unambiguous way. The CPL specification can therefore be automatically transformed into a series of differing logics based on the modality of the verb, but the resulting notation itself is difficult to work with. Therefore, a graphical layer, based upon the object modelling technique (OMT), has been built on top, which hides the resulting notation from the user so that models are easier to work with. Due to CPL's formality, it is also used to generate natural language sentences which can be used as a means to validate the resultant model generated from the CPL specifications.

Overall, the approach requires a manual transformation from natural language into CPL however, a set of rules have been developed to assist this processes. The primary rules are based upon the discovery of linguistic features, where nouns identify classes/objects and attributes and where verbs discover relationships.

In addition, a knowledge base is used interactively in conjunction with the primary rules for model feature detection. The knowledge base utilised is WordNet **[Mil95]**, a large database of words, where each word has a set of senses that are semantically linked through synonym sets for differing contexts. Their approach is to utilise WordNet's lexicon database interactively and for the user to undertake disambiguation of a given word. This ensures that the model will be both semantically and syntactically correct.

Additional rules have also been identified that constrain this process for relationship detection, and are defined as follows:

- Relationship types require certain class types
- Class types cannot be related in some systems

The first of these rules puts constraints on the type of classes that can be involved with certain relationships for example '*buying-something*' would require a class type of *'person'*. The second of these rules, for example '*marry-relationship*' forbids a relationship between two different species.

For example, *'The man married the dog'* would not be allowed since *'man'* is a decedent of *'person',* whereas *'dog'* is a decedent of *'animal'*. The process of detection is interactive, reliant on the analyst, where the knowledge base assists in governing the invocation of these rules ensuring that the features of the model are correctly defined in a semantic and syntactic context.

Verb discovery also allows detection of specific relationship types such as generalisations, aggregation and attributes. These are defined as Standard Static Relationships based on the presence of *is_a, has_a,* exists and *consists_of,* but the onus is placed on the analyst to discover these relationships and features.

In further work **[BV96],** the introduction of additional facts such as the sentence structure and define additional rules such as *subjects, predicates and objects* of the sentence, based on manual grammatical analysis. These features aid the process of making a correct decision in the discovery of model features and transformation into their CPL notation.

Additional knowledge bases are also introduced defined as domain and application specific models. The domain model consists of high-level concepts and the relationships between concepts, whereas the application model refines the domain level information into bespoke definitions and gives access to concepts relating to that domain, both are manually developed.

Overall, the analyst has to create, sift, refine and understand the specification prior to transforming the requirements into the conceptual prototyping language and to this automatically being transformed into an actual model of the system. It therefore requires manual identification of the key concepts that play an important role, such as identifying the key classes and relationships, but assistance is given through the models developed and existing knowledge bases that may also bring additional understanding to the process. This is a key challenge of the approach and consumes considerable amount of effort during the transformation and information gathering phase.

Juzgado and Moreno **[Mor97, JM00, JM00a]** propose an object oriented modelling technique based on the use of linguistic information taken from informal requirements specifications. Their key objective is to analyse this linguistic information from a semantic and syntactic standpoint and extract, by means of a formal procedure, the key components to develop an object oriented and behavioural model.

The approach is based on Spanish language specifications. Their process relies on a subset of natural language, thus restricting the expressive nature of the language, but defining clearly the syntax of the requirements. This is achieved through their utility language, which defines a set of patterns where the original requirements have to be manually transformed and where the separation of both dynamic and static requirements is undertaken. Once transformation is complete, it is then possible to construct both static and behavioural models.

The utility language gives a direct mapping between conceptual patterns allowing identification of classes, attributes, multiplicity, single/multiple inheritance hierarchies, relationships and behaviours.

In a static context, the requirements are restructured to replace pronouns and noun phrases to one explicit noun. Noun Modifiers such as adjectives are discarded, but these are typically a key source of candidate attributes. Once the specification has been sanitised, every sentence is transformed to conform to a linguistic pattern defined as follows:

- Subject-Verb-Object (SVO),
- Subject-Verb-Object-Complement (SVOC)

These are simple sentence construct rules, but this restructuring makes it possible to consider nouns (subjects/objects/complements) as classes and verbs as relationships.

Attribute detection is built upon the premise that if a class only participates in one relationship then that class will be an attribute of the other. However, it is left entirely to the developer to decide upon which class should be the attribute from their interpretation of the rule.

In addition, class multiplicities are also detected during the process. The extraction is based only upon the presence of a determiner such as *a, the* and *an*. Determiners themselves, only ever express possible or definite existence of a certain *thing*, which identifies a single multiplicity for the candidate. Furthermore, multiplicities extracted only applied to the relationship between the classes discovered in the patterns, rather than considering each individual class and its role within the relation: for example, determiner type, and noun plurality.

The behavioural/dynamic features of the system are also discovered by a defined pattern *'if-then'* structure. The requirements have to be manually transformed to conform to this pattern. The restructuring also follows a similar requirements normalisation to that of static requirements. Overall, the manual transformation requires an in-depth understanding of the requirements specification.

The key challenges of the approach is the need to transform the original specification into the utility language, where interpretation of their rules and the prior understanding required of the specification make the manual transformations process difficult for the analyst. Ambiguity and the inconsistencies contained within the specification only exacerbate the situation and the majority of the effort expended is still consumed by the analyst, which is compounded by specification issues. However, the approach appears to achieve its goal by producing an OOM model, but as with many of the approaches discussed here, there is no evaluation to support their findings.

Bajwa *et al* **[BSC06, BCA06, BSM09]** propose an approach towards automated requirements specification analysis using natural language analysis techniques. The key goal is to create UML models and usable software code from textual specification documents, where nouns represent classes, verbs represent operations and adjectives represent attributes. The key models the approach creates are Class, Activity and Sequence Diagrams.

The key steps in their approach are defined as follows:

1. Text Specification Acquisition
2. Natural Language Analysis
3. Knowledge Extraction
4. UML Model Generation
5. Code Generation

During specification acquisition, the user inputs the information contained in the specification only using relevant information by means of simple declarative sentences e.g. *The players dribble, pass and shoot the ball*. Using only relevant information avoids one of the key challenge that is an issue for many approaches, ambiguity. The usage of simple declarative sentences also reduces the requirement for formal intervention by the user.

Language analysis is undertaken with this input, which discovers all the relevant parts of speech such as nouns, verbs and adjectives; this is achieved through part-of-speech tagging. During this phase, the thematic role of each component contained within the sentence is also discovered. Thematic roles aim to uncover the Actor, the one causing the action, Co-Actor, Recipient and others, which is very similar to understanding what the Subject, Predicate and Objects of the sentence are. It is not obvious how this information is utilised during the knowledge extraction phase, if at all.

The knowledge extraction step aims to uncover the classes, operations and attributes from the specification under consideration, which are respectively symbolised in natural language as nouns, verbs and adjectives each of which map to one of these definitions. Similarly, the generation of both UML models and code generation also relies on this mapping, where detected classes and other features are easily transformed into their respective formations i.e. classes, operations and attributes.

One weakness of the approach that is not addressed is relationships between components. They do state that relationships are extracted through the presence of prepositions. However, the verb can also express relational information between candidate features as well and it is not discussed why verbs characteristics have not been exploited.

Given the approach and its removal of ambiguous information beforehand, its accuracy is relatively high from their evaluation: on average 83%. This loss in accuracy is perplexing and can only be attributed to loss of information during the specification acquisition phase, which requires the user to transform the specification into simple declarative form. It is at this point, where there is a high risk of information loss or misunderstanding.

The evaluation itself is based upon four criteria: *Objects/Classes, Attributes, Methods* and *Relationships*. There is a maximum score definition of 25, but what this actually means or how the maximum has been derived is not discussed. Each of the components *(classes, attributes, methods and relationships)* are individually counted, where correct identification gives 1 point and an incorrect identification results in a -1 point. However, where the validated results come from is unknown, as is the question as to whether the results represent only one or many sets of evaluated data.

Christiansen *et al.* **[CHK07]** propose an approach towards UML class diagrams through Definite Clause Grammar (DCG) and domain knowledge expression through Constraint Handling Rules (CHR).

The key aims of the approach can be defined as follows:

1. Capture of user requirements, in restricted natural language
2. Development of CHR Handling Rules
3. Transformation to DCG Clause Grammar
4. UML Class diagram generation through GraphViz

The approach is made possible through simple sentence constructs in the form of subject (S), object (O) typically nouns, and verb (V) that come together to form SVO triplets. These are derived from Use-Case descriptions, which are widely utilised to map customer requirements. The use-case descriptions, typically extracted from an initial informal specification, require an additional processing step to ensure simple SVO sentence construct order.

The SVO ordered sentence constructs are then transformed into DCGs defined as follows:

- Basic Sentence
    - The basic sentence is a SV or SVO triplet, where consideration of the verb is the key staring point. Where both S and O represent classes and V represents relationships or operations. In the case of SV, it would solely represent an operation contained within the subject class
- Property Sentences
    - These typically indicate properties of the subject itself that are expressed through possessive verbs, indicating ownership. This leads to the object, i.e. *the noun following the verb of the sentence,* becoming an instantiation property/attribute of the sentence's subject only if both subject and objects are nouns and have been created as classes.
- Inheritance Sentence
    - Investigates verb *is_a* construct, which subsequently leads to the detection of sub-/super-type relationships, *e.g. A student is a person*, indicating a possible abstract/generalisation, i.e. *a student is a type of person*.
- Instantiation Sentences
    - Still using *is-a* constructs as an identifier, but where a Proper Noun (*i.e. a noun representing a unique entity such as London)* is used instead of a noun *e.g. John is a student.* The Proper Noun in this case is taken as an instantiation of type student.
- Adjectives
    - These provide more information about the noun they modify, typically being attributive of the noun they are attached to. Even though it is something consider by the technique it is subsequently ruled-out due to possible ambiguities it may introduce. The consideration of adjectives is defined as either representing a sub-class or property in terms of modelling. However, given the grammatical meaning of adjectives if they are attached to the noun then they can only be considered as a property of that noun, and not as a subclass type.

- Pronouns
    - A grammatical reference designed to manage anaphoric references, pronouns such as *he, it,* and *she* typically refer to some other noun which has already been introduced. They utilise a simple heuristic that considers the most recent occurrence of a single subject contained within the current or previous sentence. However, if there is more than one candidate subject, then resolution is abandoned.

The DCGs at the lowest level are actual language constructs such as nouns, verbs, and their individual parts-of-speech. This allows the extraction of the sentence components that can be extracted by the constraint handling rules.

The CHRs will therefore discover relevant sentence components such as nouns, and extract and define these according to its rules. For example, *A dog is an animal*, will result in the creation of *class(dog), class(animal),* and will also discover a generalisation between *dog* and *animal,* defined by *extends(dog, animal).* A separate knowledge base can also be built from CHR constructs and used as an additional source of information to complement analysis however, this is manually developed and prior to actual processing.

The resulting output from the CHRs is then subsequently transformed into another DCG defined for GraphViz. As a result, the GraphViz syntax model can be used to generate the actual UML class diagram; this can then be presented to the user for analysis.

Largely, the approach relies on the Use Case descriptions that have already been extracted from the original user specification. As a result, any missing requirements will not be analysed nor considered by the approach, which could lead to a situation where important design features could be missing from the proposed software system and is the same for any other system as well. Additionally, the approach has the requirement to ensure that the Use Case statements are in the form of SVO triplets; a manual transformation process which may also introduce ambiguities and lead to a loss of system specification information.

Nonetheless, the approach does produce a result that is a step forward in automated software development, although this is done at the expense of an assumption; one that believes the majority of the information contained within the Use Case specification is relevant and finally, there is no evaluation of this approach.

Gelhausen and Tichy **[GT07];** Gelhausen and Korner **[GK08]** present a technique towards a semi-automated approach for automated software development that will form the starting point of the Model Driven Development (MDD).

Their approach is based upon an intermediate transformation of the specification into SENSE (Software Engineers Natural language Semantics Encodings) a kind of super-graph that is an extension of hyper-graphs, where edges not only connect to nodes, but can also connect to other edges. The key intention of Sense is to encode the semantics of natural language and not to distinguish between word order, voice or tense. In addition, SENSE carries no understanding of word definitions therefore all words could equally mean the same thing.

The semantic encoding is achieved through purpose built annotations known as SALe (SENSE Annotation Language for English) roles. These roles can define actions (AG), message passing (HAB), recipients (RECP), donors (DON); that something is acted upon (PAT) and removal of superfluous information by attaching a (#) tag. This set of annotations allows the key aspects of the user specification to be identified prior to SENSE processing.

SALe requires a manual application of these annotations to the plain text requirements specification. This transformation process could be considered synonymous with a typical software analysis approach, where the key components of the model such as classes, relationships and operations are discovered and annotated. The only difference being that the SALe document produced is then automatically processed by SENSE through a set of transformation rules based upon these SALe annotations. The resultant output is the super-graph detailing all the relevant classes, operations, relationships and inheritance hierarchies, which can then be transformed into a UML model through a set of transformation rules.

This approach is a human manual analysis technique, where the key decisions of what to include and exclude are decided upon by the analyst. There are no means to provide an automated detection technique based upon the linguistics or semantics of the specification. It is therefore possible for relevant information to be overlooked and excluded during the annotation process if deemed unimportant by the analyst.

Vinay *et al* **[VAD09]** have developed the R-TOOL, designed to analyse elicited English natural language requirements specifications, which is used to extract classes, attributes, operations and relationships. This is achieved through the application of NLP and rules designed to analyse the specification.

As with other approaches, the R-TOOL performs full natural language processing tokenisation, pronoun resolution and part of speech detection. However, a prerequisite of the approach requires the specification to be written in the active voice and to be in simple sentence form (i.e. SVO Triplets) prior to any formal class model detection.

The approach takes a rule-based approach similar to other techniques, where nouns identify candidate classes, verbs identify relationships and operations and certain noun structures (namely noun-noun) to find attributes.

The noun-noun rule is an interesting approach to attribute detection, where the first noun is considered the class and the subsequent noun is considered its attribute. However, in terms of linguistic analysis, the first noun is considered as a modifier to the second noun/head noun. So in this case, there is a greater likelihood that the first noun is actually some kind of attribute rather than the second or it may even represent some form of subtype-supertype relationship.

In addition to attribute detection, certain verb constructs are also utilised to determine candidate attributes through the identification of possessive verbs such as *have, denote* and *identify.* Furthermore, nouns preceding prepositional phrases such as *cost of soup*, where *cost* would be considered as the attribute, are also detected during the attribute detection phase. However, not all prepositional phrases may indicate attributive qualities; it is also possible that they identify spatial, temporal or comparative references.

With the initial list of class candidates discovered, additional rules are also applied to prune the class list; rules such as, frequency analysis and candidates that have no attributes , which results in those being removed from the final list.

Frequency analysis is based on an individual words frequency of occurrence count within the document. However, no frequency threshold is actually discussed within the approach. Therefore, it is difficult to establish how this is actually defined and whether or not it is user definable. Nonetheless, both candidates with a low frequency and those having no attributes will lead to them being discarded and not included in the final design.

Finally, the approach to relationship detection can identify simple association relations defined by simple *noun-verb-noun* constructs. However, the approach also goes as far as to identify generalisations and aggregations as well.

Generalisations are discovered through a top-down search for nouns that are also composed with adjective modifiers, and once discovered a generalisation is created. Adjectives typically modify the noun that they are attached to and have several definitions depending on their context. Their most common usage is to be attributive to the noun they modify. The only case where an adjective may be considered as being or acting as a noun is in the nominal case, but the approach does not provide any justification or a present a mechanism to determine the adjective's actual function.

Aggregations are discovered through a simple pattern matching rule primarily defined by the verb of the sentence such as *something [contains, is made up of, is part of] something*. Therefore, upon detection of such constructs an aggregation relationship is utilised instead of an association.

Overall, the key strengths of this approach is to utilise not just the types of language constructs in the discovery of class features, but to enhance their approach through rule based analysis as well. However, the rules applied such as frequency analysis could potentially lead to a loss of vital model features and discarding of class candidates. The approach towards frequency analysis is not fully disclosed; it is therefore difficult to infer whether this is threshold based or if only high-value words are considered as candidates. If so, and as said, it could lead to the loss of potential high-value candidates as these may only appear once or twice within a specification, but are ones that play a pivotal role within the specification. However, because of their low frequency of occurrence within the specification, also known as an under specification, it would require manual investigation of discarded terms to evaluate their value. An under specification relates to a situation where a key the feature should be included within the resulting model/analysis, but since it has a low frequency of occurrence within the specification it is ignored by the approach.

The focus of their evaluation is qualitative, a comparison of manually collated results vs. features identified by automation. The evaluation is limited to only one specification and there is no information on how the manually identified features were collated. In addition it is unknown whether the specification utilised in this approach demonstrates the approach in its best light as the results comparison is very good. To understand how well the approach does perform a more extensive evaluation is needed, but one that is not available.

Uncontrolled Language:

Motoshi *et al* **[MHH89]** define a technique to extract a formal specification from an informal one, where they extract a set of candidate nouns and verbs, from the informal specification. This extraction is achieved through a noun and verb dictionary that is automatically applied to the specification texts. The set extracted nouns and verbs are manually classified into product sets such as classes, attributes or actions from nouns and relational, state, action or action/relational from verbs.

The key argument for manual classification rather than automated is that it is difficult for computers to identify what words are important and relevant to the specification. In addition, sentence structure is also taken into consideration during the manual classification process such as the subject (S), verb (V) and object (O). The subject of the sentence identifies the sender of a message and may strongly indicate a class. Verb patterns based within grammar such as S+V, identifying intransitive verbs, can allow the manual consideration of objects that modify their own state. The approach also demonstrates how other simple verb structures defined as *'action verb rules'* can also aid in the manual identification of additional design features such as relationships, operations and attributes.

These rules typically result in SVO triples where the subject is joined to the object of the sentence via the main verb, where the verb identifies the relationship and/or operation. All of this extracted information can then be utilised to extract a formal specification from the informal one.

The key drawback of the approach is the burden of the manual analysis activity. This is compounded by informal specification as it increases in size and complexity. The approach is entirely dependent on manual analysis, which is subject to experience, understanding and bias to identify the relevant features that should and must be modelled.

Goldin and Berry **[GB94]** note that for many methodologies that allow the transformation from requirements analysis to initial design, in the majority of cases requirements are often ambiguous, ill defined, incomplete or just simply wrong with respect to the users' needs.

They have identified that abstraction identification is a key problem within requirements analysis process. *Where an abstraction is the ability to ignore enough of the details contained within the specification and only capture the main ideas or concepts.* Therefore, abstractions are key to understanding what is actually required, but the abstractions themselves are surrounded by a mass of natural language texts from where they must first be discovered.

The manual elicitation of abstractions takes the form of identifying nouns and noun phrases; this is aided through the identification of the grammatical subjects and objects contained within the sentence and most importantly, the analyst's understanding. It is entirely a human thought process, where each element has to be considered for inclusion. However, as the document grows in size, the more complex the task becomes, the greater the chance of important features being overlooked and why a semi-automated approach was sought.

Initially, automation sifts the texts through natural language analysis techniques and frequency analysis. The results can then validated by a human. The key benefit of automating this process is that it can be guaranteed that no element contained within the specification will be overlooked, whereas it cannot with a manual analysis approach.

The automation process proposed is based on repetition or frequency analysis, which is the basis of their key assumption stating that key abstractions are discussed more often within the specification document. This statement can be held true with many search and retrieval techniques where it is possible to identify the importance or relevance of a particular term contained within the document itself. This assumption gives rise to a situation where important abstractions, which have a low frequency of occurrence, will be overlooked by this technique, which may indicate that these are of greater importance than the high frequency terms **[Jon72]**.

Automated linguistic classification, through utilisation of a natural language parser, discovers every noun and noun phrase, even if the word is derived from the same stem such as *purchased* and *purchase.* The frequencies are identified and added together. In addition commonly utilised stop-words such as *a, an,* or *the*, are ignored during analysis phase.

Additional issues identified are acronyms. These are introduced to replace longer phrases, primarily only of use to an analyst, and to avoid repetition. Acronyms cannot be managed through threshold analysis and therefore require manual identification by the analyst and added to an exclusion dictionary, which then allows their identification and addition at the end of processing.

The threshold based analysis has two specific issues. A low frequency threshold increases the number of terms identified, but incurs a penalty through introduction of irrelevant words. Whereas, a threshold set too high may miss important features.

This approach is one of the few that also have some form of validation of the approach where a non-domain expert using the tool was compared against three domain experts not using the tool, independently of each other. The key conclusion was abstraction identification completed the same task as these experts in one day in comparison to three months worth of their work and was able to identify features that the experts overlooked.

The approach demonstrated that through means of frequency analysis and their evaluation it is possible to identify the key features faster and more efficiently in comparison to three domain experts. This is an interactive process, requiring constant involvement, consideration and frequency threshold re-balancing to achieve the best results, but it only identifies the key abstractions (candidate classes) and no more.

Gervasi *et al* **[AG97, AG99, GN02, AG06]** present a web-based tool for requirements gathering, elicitation, selection and validation which is utilised to build models of a proposed software system. What is of interest is how they actually select and identify relevant components to be included within their model of the proposed software system. The model generated from this information is supported by an automated processing technique. This requires analyst support, via a costly manual activity involving development of a System Glossary and MAS (Model, Action, and Substitution) rules.

The System Glossary developed contains all of the key abstractions, significant terms and flows of information that are contained within the requirements and are considered important by the user. For example, given the phrase, *When the server receives from the terminal the password, the server stores the signature of the password in the system log,* the system glossary as defined by the user would contain *{terminal/IN/OUT, server/IN/OUT/ELAB, password/INF, signature/ATTR, system log/STORE}*, where *IN/OUT/ELAB/INF/ATTR/STORE* are domain specific terms representing data flows. However, during automated processing the glossary is ignored and is more for the benefit of the user during the development of MAS rules.

Table 2.3-4 demonstrates the MAS rules utilised by the process for the identification of model components. Although each requirement is manually transformed into a series of these MAS rules this could potentially lead to a loss, oversight or misunderstanding of the original requirement and it is the analyst who is still making the decision as to what is included or not through development of these rules.

Table 2.3-4 User Developed MAS Rules

Model	Action	Substitution
WHEN event/EVT action/ACT	Out DEPEND $action $event	-
Receiver/IN RECEIVES data/INF FROM sender/OUT	Out DFLOW $sender $data $receiver	$ID/EVT
Agent/ELAB STORES data/INF IN datastore/STORE	Out DSTORE $agent $data $datastore	$ID/ACT
Attribute/ATTR OF object	Out ATTR $attribute $object	$ID/INF

The rules themselves are applied through fuzzy matching, where automated processing will decide if a fragment, a feature identified from the requirements specification, matches one of the MAS rules. If so, given the model, the action is executed and the matching requirement is replaced by its substitution, thus generating the required design components, (but these have been pre-determined by manual means, automation is only processing the subsequently developed rules).

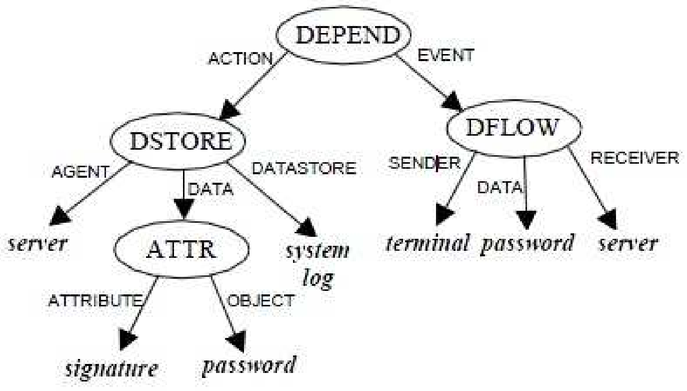


Figure 2.3-1 Parse Tree Corresponding to Sample Requirements [AG99]

The process itself comes into its own through its views, which allow metric applications, various checking and model transformations including Object Oriented, Data Flow Diagrams, Dynamic Models, Entity Relationships and more. This is where the real strength of this approach lies and all can be derived from a set of similar parse tree structures, see Figure 2.3-1. There is an evaluation of how well the technique performs in processing requirements (i.e. how fast it can process requirements) and how many lines of code the approach was written in but the evaluation does not provided any consideration or meaningful understanding of how well the actual technique performs.

Sylvain *et al* **[SBB99]** propose a technique utilising natural language texts and semantic analysis towards the creation of candidate lists of classes, operations and attributes. The approach is utilised during the initial stages of the analysis, which allows the engineer to concentrate on the collection and preparation of accurate textual descriptions of the problem domain. This is undertaken without having to utilise any object-oriented tools, diagrams or techniques during this analysis phase.

The key to the approach is the extraction of nouns, which indicate classes, verbs that indicate processes, and identifying attributes from adjectives, all which is aided by their domain independent natural language and a semantic analysis tool.

The domain independent parser processes each individual sentence and identifies all of the relevant parts of speech, and produces several parse trees; the analyst then has to select what they think is the correct parse structure for the sentence which is then passed on for semantic analysis. However, the complexity and the structure produced by automated analysis may be (initially) confusing, which could lead to an incorrect parse tree selection that would subsequently have an adverse effect on the overall analysis process.

Nonetheless, once the parse tree has been selected, semantic analysis is undertaken by a separate module. The key to the semantic analysis is centred round the verb of the sentence. This is not a semantic analysis that generates an artificial understanding, but one which primarily looks towards case relations that represents semantic relationships between the main verb and the key sentence components (such as *subjects, objects, prepositional phrases*). As with language processing, the user has to select the relevant semantic analysis which they feel best represents the relation semantics of the current sentence.

Along with both language and semantic analyses, the user is finally presented the key details such as, the main verb, nouns and any adjectives. It is then the responsibility of the user to generate candidate lists (such as *classes, attributes and operations*) and make the decision as to what are the best candidates, as a preliminary analysis step prior to full (manual) model creation.

This semi-automated approach has been evaluated by means of comparing the results identified by the process to what is discovered by human experts. Overall, 76% of classes, with a majority containing their attributes, and 66% of the operations were identified in comparison to an expert’s preliminary model.

The majority of this work is manual when considering that the key decision as to what is a class or an operation is a human decision process. In addition, identifying the correct parse trees and semantic analysis is a crucial step in the process were misunderstanding the requirements and language constructs will have a detrimental effect on the overall process and outcome of the candidate lists produced.

Lee and Bryant **[Bry00, LB02, LB02a, LB02b, LB02c, LB03, BLC+03]** propose a technique towards semi-automated software development, which is capable of generating both models and software code from natural language requirements specifications.

Their approaches utilises techniques such as domain knowledge (DK) and domain specific knowledge (DSK) that are manually developed to assist automated analysis and extraction of class features. Additional supporting techniques applied to their approach include syntactic analysis, semantic analysis and a custom-built part of speech analyser used to identify the relevant language features such as nouns, verbs and adjectives.

Syntactic analysis identifies the subject and objects contained within the sentence, based on the premise that the first noun discovered in the sentence is the subject and where all others are objects although this assumption is not entirely true when considering passive sentence constructs. The detection of syntactic features aims to better aid and increase the accuracy of their approach. Semantic analysis, aided by WordNet **[Mil95]**, assists in co-reference resolution, where the semantic groupings of WordNet **[Fel98]**, such as *'Animal, Person'*, can be used to find words that are a possible candidate of the co-reference.

However, prior to any form of linguistic analysis, the requirements specification is manually transformed into an XML representation, which is a means to simplify what automation has to process, and through the definition of additional meta-information it can then guide the relevant execution path. The structure is derived from the common document structure by means of *sections, sub-sections, paragraphs and individual sentences*. This enhances the overall quality, where contextually related information is grouped together. The addition of meta-information also included during the manual transformation of the specification helps to identify important sections and sentences contained in the specification. The XML representation is the first level of domain knowledge, known as a contextual document model.

Each of the top-level tags *{section, subsection and sentence}* allow management of contextual information, which can be used to aid user-specific queries, but primarily aims to simplify automated analysis.

- The *section* tag is used to identify the overall context through a meta-attribute defined as *object* and a descriptive title such as *ATM*
    - A *section* tag can also contain *section* tags (realistically representing a paragraph) to maintain context and also identifies what information is contained within that tag for example a *'withdrawal service'.*

With a *section* defined, it is possible to identify sentence-level meta-information utilising descriptive tags such as *{head, pre-condition and sub}.*

- The *pre-condition* tag identifies conditions that must be met for each of the sentences that are contained within sections or sub-sections *i.e. bank verifies ID and PIN giving the balance* indicating that any features identified within the section/subsection must meet the defined pre-condition
- The *head* tag is utilised to mark a sentence containing a function signature, for example, *ATM withdraws an amount with ID and PIN giving the balance*, where function identifies the containment of an operation *i.e. withdraw*
- The *sub* tag identifies *a post-condition,* which must be met after conclusion of the operation, *e.g. And then it updates the balance in the bank with ID*

In addition, domain specific knowledge also has to be manually defined, also in XML, but carries a significant amount of detailed information such as, inheritance hierarchies, associations, compositions, attributes, types as in *{integer, string}*, specific values and synonymously related words, which is defined for each concept within the domain.

The domain specific knowledge does not have to be extensive, but providing information that is more detailed will surely ensure the success of the automated analysis process. However, going to the extent of defining so much operational, structural and relational information, prior to the automated analysis, is of concern. It seems rather counter-intuitive to expend the effort manually capturing this information and it would be better spent actually creating the model (on the part of the analyst), rather than defining a domain specific model for automation.

With the domain models created, class detection is based on the rules of nouns indicating classes, verbs implying operations and adjectives demonstrating attributes. Therefore, upon detection of these language features, the DSK can be queried for any additional/supporting information during the analysis phase. Furthermore, structural aspects of the sentence are also taken into consideration when detecting a class. The structural features considered within the sentence are subjects and objects, which support the case for class creation, since both sentence subjects and objects identify important aspects of the sentence.

A simple rule is utilised to detect sentence subjects, where the first noun in the sentence is always the subject and all subsequent nouns are objects. Given a passive sentence construct, *The balance was given after verification by the bank*, even though *balance* is the subject, the true subject of the sentence is *bank,* as it is the bank who is calling/performing the action *verification*. This does not really affect class creation in any particular way, but it can be detrimental when deciding which class should contain an operation. This may be resolved through their defined knowledge bases.

In addition, cases can arise in written texts where the subject may be unknown due to pronoun usage, such as *{it, that, them, they, he}*. The key approach is semantic analysis aided by WordNet and recency constraints to undertake co-reference resolution (i.e. the last seen noun).

Their approach to co-reference resolution considers the most recently seen noun as the candidate, which then checks its semantic definition with WordNet. If it represents a living thing, the pronoun resolution can take place, otherwise the next most recently seen noun is taken into consideration. This is not an endless search process, and is kept within the confines of the contextually related section/subsection of the specification. Although a problem does exist with this approach to co-reference resolution as the most recently seen noun may not be the correct reference and it may not have a *living thing* semantic as well. Take the case of an *ATM machine,* it does not have a semantic that falls within the category of livings things, but could easily be co-referenced by a pronoun resulting in an incorrect reference.

Co-reference resolution in the approach only considers previously introduced terms. The situation can arise where a pronoun is used for a term that has not been introduced to the reader as yet, known as cataphoric reference, also a form of co-reference resolution. In this situation, and given the approach to co-reference resolution this could lead to the situation where a completely irrelevant term is referenced as the actual candidate.

Nevertheless, these analyses and rules are then utilised by automation to process the natural language specification and generate a Two-Level Grammar (TLG) representation. The TLG, a specification language defined by this technique can subsequently be translated (automatically) into VDM++ (an OO extension of the Vienna Development Method), thereby allowing model generation, identifying classes, relationships, operations and attributes. The resultant TLG/VDM transformation can also be converted into a high-level programming language such as Java or C# to allow rapid prototyping.

Overall, this approach towards automated analysis provides a robust means towards model and actual code generation, which can be considered its key strengths; disappointingly, there are no evaluations of any type for this technique. The keys to the approach are the document reformulation into a contextual representation; generation of the domain and domain specific knowledge bases and model generation rules, which guide and assist automated analysis. However, a considerable amount of effort needs to be exerted prior to any automated analysis to the extent where the manual analysis transforms the requirements into both domain and domain specific XML models, which carry such a level of detail it almost makes the automated analysis part irrelevant.

Overmyer, *et al* **[OLR01]** propose a technique for conceptual modelling through linguistic analysis of the natural language requirements specification and have developed a tool, 'LIDA', which assists the developer by automatically detecting possible features such as classes, attributes, relationships and operations from a natural language specification.

Their methodology is defined by the extraction of language lexical features such as nouns, verbs and adjectives, which are then compiled into candidate lists of classes, relationships, attributes and operations. The extraction is achieved through two custom built dictionaries: one that represents nouns and the other, verbs. The process scans the texts and pulls out matching words that are contained within either dictionary set, which results in a candidate list of potential class features.

With candidate lists generated, the human can then manually identify the actual candidates for the conceptual model where through a further automated step these features are then transformed into a UML diagram along with API descriptions. The API descriptions are a reverse engineering of the resultant model generated back into natural language statements.

Even though the overall aim is to alleviate the manual identification, the approach is entirely dependent and reliant upon the developer identifying appropriate classes, attributes, operations and relationship from the list of all possible candidates. The approach further assists manual analysis by providing frequency of occurrence information for each candidate thus aiding the manual decision making process.

Kof **[Kof05, Kof05a, Kof07 & Kof08]** proposed an approach to requirements document analysis through natural language processing as a means towards ontology extraction. The ontology can then be utilised to derive models and drive further manual analysis of the proposed software system.

The approach taken follows three key steps:

1. Individual Term Extraction
2. Term Clustering and Taxonomy Construction
3. Term Relationship Discovery

Individual term extraction is assisted through natural language analysis of the sentence and construction of a full part-of-speech parse tree using an external tool, ASIUM **[ANF98]**. The purpose of utilising the full parse tree eases the identification of sentence predicate (main verb) and both its arguments, subjects and objects (nouns). The approach initially locates the main verb, and then with a series of left and right traversals of the parse tree, it extracts both subjects and objects. In addition to extracting individual terms, compound terms are also discovered during this approach. Thus, ensuring full compound terms are extracted rather than them being extracted individually.

In addition, during the extraction of compounds there are many structures in the form of *(Property) of (Object)* e.g. *failure of water level detection unit*, therefore in this case the whole tree would be extracted as one entire concept. However, *of* does not only indicate properties of other objects, but can also indicate many other constructs such as direction, time and others, which would require disambiguation to ensure correct interpretation. The approach only considers the 'existence' property and does not undertake any disambiguation.

With all terms extracted, it is possible to start constructing the taxonomy using ASIUM. ASIUM builds clusters of nouns discovered during the initial phrase and makes use of contextual, lexical and syntactical similarities to decide whether two terms are similar and if they have a high similarity score, they are grouped together. To enhance the chances of similarity towards each of the extracted terms, they are also reduced to their base stems. A further enhancement to the cluster results requires a manual search and discovery of intersecting cluster, which can be manually inserted into the final taxonomy model.

Using another external tool, KAON **[SA97]**, it is also possible to perform association mining and identify relationships between concepts. This is achieved by means of a simple count, identifying how many times a particular concept appears within input texts. Therefore, a decision whether an association is important and should be included is undertaken by two metrics and a user defined threshold.

The metrics primarily investigate how many times a set of concepts appear together within the same sentence and within all sentences, defined respectively as *support* and *confidence*. However, the user has the final decision on whether to include the association or not. There is the possibility with this strategy for important association, which may only be mentioned a minimal number of times within the texts, to be overlooked and subsequently lost during the analysis process.

The majority of this approach is automated by means of natural language analysis and use of other external analysis tools. Other aspects are interactive such as clustering and relationship discovery that only require the user to confirm/validate the results. Nonetheless, the user still requires an understanding of the requirement texts.

In addition to user understanding a set of rules were specifically developed to enhance the process that can be applied to the specification document and are defined as follows:

- Use the same name for key concepts;
- Mark compound words with hyphens;
- Avoid compound concept conjunctions e.g. *'stop or start message'* should be *'stop message or start message'*;
- Do not use verbs in the form of *'have'* or *'be'*;

- Avoid erroneous/supplementary information aimed particularly at the reader;
- Avoid cross sentence references *e.g. 'Message X is sent by unit Y'.*

These rules effectively require a rewrite of the specification only to improve the results, which also identifies a key limitation of the approach. The claim is that their approach will effectively work without this step, but better results can be achieved by performing this transformation. This passes additional burden onto the developer/analyst to have a greater understanding of the requirements prior to the re-write taking place, and also lengthens the overall analysis phase as well.

Overall, the results produced by this interactive process leads to a well-defined ontology. This can subsequently be transformed into various model types such as Use Case Models or Message Sequence Charts to aid understanding and complement the requirements analysis phase.

The evaluation investigates the completeness of concepts extracted by the approach, but only the concepts and not the relationships between them. The approach argues that relationships are not explicitly defined in the text thus contradicting many of the related works that do extract relationships from the text.

The results of the evaluation are compared against those that have been identified by the author. They do identify this as a threat to the validity of the evaluation and state that it should be undertaken by a domain expert. Finally, there is no analysis to bolster the key findings that the extraction of concepts matches those of the concepts contained within the document. The evaluation overall does not lend itself to validating the approach.

2.3.1 Fully Automated

The review of the fully automated works is similarly split in to two key sections, controlled and uncontrolled. The following reviews are chronologically ordered.

Controlled Language:
Nanduri and Rugaber **[NR95]** proposed an approach that performs requirements validation via the creation of an object model automatically from a specification. This model is then compared against a manually developed solution, which helps to identify any missed classes, relationships and alternative design choices. Linguistic analysis of the specification is used to

identify and create the relevant classes, relationships, attributes and operations automatically from the language. This analysis is based on rules they have developed to extract the relevant model components.

Due to language complexities such as ambiguity, inconsistency, sentence structure and incompleteness, it is necessary to rewrite the specification document manually as a set of simple sentence structures.

```
                     +----Js---+
 +-Ds-+---Ss--+--MVp-+   +--Ds-+
 |    |       |      |   |     |
The cow.n jumped.v over the moon.n
```

Figure 2.3-2 Example Link Grammar [NR95]

The basis of the approach is formulated within a link grammar, where each link identifies components of the sentence that can be linked to other aspects of the sentence. For example, both the subject and the verb of the sentence would connect through one link. Another link could exist between both verb and any objects and more links may be present between all three components (subject, verb and object). This is achieved through utilisation of an external tool that automates the analysis and returns a link grammar parse tree (see Figure 2.3-2)

These links form the basis of their rules for the creation of model features and their approach is based solely on the presence of these links and their order. This then allows the decision to be made as whether they should be included within the design. There is no cognitive or syntactic analysis undertaken automatically between these components of the sentence to decide upon their inclusion.

The strength of this approach is in the creation of an alternative design choice for the developer and the identification of any missed or overlooked features of the specification. However, the developer still has to create a design for comparison manually. This is a time consuming process and is common for all manual approaches. There is also the additional effort required to rewrite the specification using a simple sentence structure, which also has the potential to introduce inconsistencies or skew the intended interpretation of the original document.

Once rewritten, the specification is analysed by a link grammar parser, which may return different linking requirements between the components of the sentence due to parser inadequacies. In addition, the parser cannot process hyphenated words, idiomatic expressions, quotation marks or undertake co-reference resolution. The resulting information serves as input for analysis by their extraction rules, where nouns serve as classes and possible attributes, verbs as relationships and operations. Unfortunately, there is no evaluation to identify the quality of the approach.

Li *et al.* **[LDP04, LDP05, LDP05a]** utilise a pattern/rule-based and interactive approach towards UML Model generation from natural language specifications. The basis of the approach requires structured specifications typically in the form of subject-verb-object (SVO) triplets to allow successful transformation and detection of candidate classes, relationships and operations.

The technique developed tags each word within the sentence with their relevant part of speech such as nouns, verbs or pronouns. This tagging is an automated process however, it is unknown if this is a tool that has been custom built for the approach or is an external part-of-speech tagger. Nevertheless, it is then necessary, after tagging is complete, to transform sentences into the SVO triplet format.

Given the tagged sentence, a pattern-based approach is utilised to create the SVO triplet structure, where the first noun is considered the subject of the sentence (S), the following verb is considered the main verb (V) and any subsequent following nouns are considered as objects (O). For example, *The baker bakes bread and cakes*, the SVO triplet would be as follows: S-V1-O1-O2, which would be translated into two separate individual triplets so that both objects are associated to their own subject as follows: S-V1-O1 and S-V1-O2. Given the example this would effectively become, *The baker bakes bread* and *The baker bakes cakes.* Though a novel technique, it is an unnecessary step splitting these into individual structures, as it is simple enough to identify the attachment of the objects contained within the sentence

With all sentences split into their respective triplets, an initial class diagram is created automatically detailing the candidate classes, attributes and operations, but none of the relationships. The approach then poses questions asking the user to confirm candidates discovered. The refinement phase is entirely user driven: one that requires a formal understanding of the proposed software system before decisions can be made concerning valid candidate features.

Overall, the approach produces useable UML and simple Use-Case diagrams as a precursor to further software development activities. There is still a reliance on the user and their requirement to have an understanding of the system prior to deciding whether a feature detected by the automated approach is correct and relevant.

Popescu *et al* **[PRM+07]** propose an approach to automated UML model generation through use of a constraining grammar, automated language analysis, transformation rules and user intervention as a means to improve the quality of requirements specifications.

The sole purpose of the constraining grammar is to allow the concise expression of the software requirements by means of simple sentence constructs and specifically stating the actual requirements. It is also the goal of this constraining grammar to reduce/remove/address ambiguities, inconsistencies and under-specifications, which may be present. This requires a manual analysis and rewrite, which on its own could potentially introduce further ambiguities/inconsistencies and loss of important information contained within the original specification.

The language analysis phase makes use of an external link grammar tool, similar to that of other approaches **[NR95]**, which detects all relevant parts of speech and identifies the connective relationships between the components contained within the sentence. These linkages are then used in conjunction with transformation rules to generate a textual description of the UML model.

In addition to the link grammar, WordNet is also utilised, but only to return words contained within the specification to their base form i.e. transforming plural nouns to their singular form. This helps to avoid the creation of duplicate classes, attributes, operations and relationships.

As previously stated, the link grammar identifies the key sentence features and connective relationships between these such as the subject of the sentence, the main verb and sentence objects and transforms the sentences into simple sentence construct in essence an SVO triplet. A key rule of the approach considers that if a link also exists between both subject and object, then it will result in the creation of two classes (represented by the nouns/subject and objects), a relationship and operation defined by the sentence verb.

During the transformation process, additional consideration is given to the verb and its type. For example, a genitive verb, indicating possession, will result in both subject and objects of the sentence being created with an aggregation relationship, rather than a standard associate.

The aggregation relation is subsequently utilised during a further processing phase, where all aggregated relationships are considered for transformation into attributes of the subject class. During this analysis, only classes that are lacking in relationships, operations and attributes will be considered for transformation into attributes and will be placed within the subject class.

With the analysis undertaken, it is then the responsibility of the analyst to investigate the diagram for ambiguities and the technique identifies areas to investigate:

1. Association relationships may indicate ambiguities and the analyst should validate were different classes communicate with the same target type is correct otherwise manual manipulation is required to ensure the model demonstrates the correct communication

2. Each class should reflect one and only one concept. Thus does *book* and *textbook* represent two different distinct concepts?

3. If a class has an attribute and it is not of primitive type, this may indicate that the attribute is not well defined within the specification. On the other hand, it may identify a genuine communication between class components.

4. If a class has no relationships with others in the model, it may indicate under-specification. This though, could be a direct result of the transformational phase undertaken prior to any automated analysis, where the original document was rewritten into simple sentence form.

Nonetheless, the approach creates a means to automate the software modelling generated from the restricted natural language constructs where the approach is validated using recall and precision adapted from information retrieval techniques. Their interpretation of recall considers information extracted by the link grammar in terms of what is contained within the source. That is, it investigates how well the actual natural language parser performs through extraction of nouns in comparison to the actual nouns contained in the specification, with a similar comparison with precision. The evaluation does not consider how well recall and precision perform in relation to a human generated model.

Strangely, the evaluation uses an *intro man page* of the *Cygwin environment* and not an actual requirements specification document. The reasoning given is that the manual page seems to be a suitable experiment source. However, the key purpose is to improve the quality of requirements specifications through auto-generated models. Therefore, a manual page does not appear to be an appropriate source of information, but for the purpose of the evaluation it should suffice.

The evaluation starts with a manual identification of every noun or compound noun by the authors, then these results are compared with the automatically extracted ones, which allow for a quantitative analysis, where the average precision rate is 89.79% with an average recall rate of 69.2%

The key strength of the approach can be considered to be within the transformational rules used to detect the key features of the model classes, relationships, attributes, operations and generalisations. However, the approach is reliant on the initial transformation process from an uncontrolled natural language to its restricted grammar and this is the crux of the approach.

Seresht, Ormandjieva & Sabra **[SOS08]** present a proof of concept that accepts a collection of textual requirements specifications as its input and outputs the resulting static and dynamic models of the captured software system. The objective of this work is to provide interactive and automated assistance throughout the process of requirements elicitation and analysis. This is achieved through three techniques: automated NLP quality assessment of the textual requirements during the elicitation phase; NLP quality assessment of the requirements during the development of a static UML model and dynamic Use Case models. The generation of a graphical visualisation extracted from the requirements is presented for user validation and feedback. There is no evaluation presented with this work.

The methodology that supports the identification of static model initially starts with a pre-processing phase that identifies a First-Cut Structural View (FSV) that is combined with Expert Compared Contextual (ECC) Models - domain data models which are a UML representation of the domain. Both the FSV and ECC models are used to generate an Improved Structural View which is subsequently transformed into a structural view.

Textual pre-processing is used to remove ambiguities and is supported by a decision-tree text classifier that applies a quality characteristic model which is not limited to just syntactic features such as passive verbs, but also word frequency and ambiguous keywords. In addition, discourse features are also considered such as words per sentence, unique words and frequency of ambiguous sentences. Once the ambiguities have been identified and removed a set of heuristics are applied that focuses on SVO constructs that considers nouns as candidate classes and verbs as relationships; no additional syntactic features are considered in the construction of the FSV. The domain models (ECC) and the FSV are compared and a variety of rules are applied that transform nouns to classes and verbs into relationships. The resulting output is an Improved Structural View (ISV) which is subsequently transformed into a static UML model.

The proof of concept controls the language by focusing on simplified language constructs and removes ambiguities during its textual pre-processing phase. It is unclear from the work whether these ambiguities can be resolved, but the authors do mention a validation and feedback phase. Overall, the initial analysis creates classes/relationships from nouns and verbs which is compared to the domain model, which is a static UML representation also detailing classes and relationships. The domain model contains extensive information ensuring that missing information is included within the final model. This calls into question the practice of why the approach is undertaking specification analysis and the creation of a first structural view when the final model created is essentially a view the domain model itself.

Deeptimahanti, Ratna & Babar **[DR08, DR09 & DB09]** propose a methodology towards automated software development based on the Rational Unified Process (RUP). Their approach, known as 'Static UML Model Generator from Analysis of Requirements (SUGAR)', is used in conjunction with natural language processing (NLP) and a supporting glossary of terms.

As with other approaches, it too requires complex sentence structures to be transformed into simple sentence constructs such as SVO triplets. It also requires transformation of all passive voice sentences, such as *Customers are transported from one location to another* into active form thus ensuring the inclusion of main subject, i.e. *Taxis transport customers from one location to another.* This ensures that the object, which is actually undertaking the action, can be discovered.

During the reconstruction of the sentence, all prepositional phrases, adjective phrases, determiners and adjectives are discarded if, and only if, they precede the subject of the sentence. The exclusion of such information, even if it is only from the subject of the sentence, can and will lead to the loss of information that could be used to identify possible relationships, attributes and multiplicity during the modelling activity.

Once the specification has been reconstructed (manually), it is then possible to undertake NLP analysis and subsequently have a parse tree returned (via the Stanford NLP Analyser **[KM03]**). The parse tree contains all the relevant parts of speech for the given sentence. An additional processing step is also undertaken by WordNet to perform morphological analysis and transformations, where any plural word is subsequently transformed into its singular form. This is only to ensure no duplicate classes are created during the automated analysis aspect of the approach.

With all the pre-processing undertaken and with the suitable parse tree available, it is then possible to undertake UML model generation. The key to the approach is a noun-phrase approach, where nouns are considered as classes and verbs are considered as operations of those classes.

To enhance the approach a glossary is constructed, which is used to ensure a common vocabulary for disambiguation purposes such as *client* and *bank* client. In addition to this disambiguation, the glossary is also utilised to remove irrelevant words that would not give any benefit to the final model. The process of generating the glossary requires manual analysis of the specification, probably undertaken during reconstruction of the specification into simple sentence form. The reliance on manual transformation of the specification prior to automated analysis and intervention still involves considerable effort on the part of the analyst to ensure that no relevant information is accidently removed during this transformation. Unfortunately, this cannot be guaranteed without further scrutinising the resultant transformation process however, the approach is not supported by an evaluation.

Uncontrolled Language:
Mich *et al* **[Mic96, MMZ02, MG02 & KZM+04]** propose a case tool that generates Object Oriented Models (OOM) from natural language requirements specification documents. Their approach is built on the premise that it should use an uncontrolled natural language (i.e. not a sub-set of natural language), that the analyst should not have to intervene for clarification during the analysis and that their engagement should only be for creative purposes.

The basis of the approach is built upon LOLITA (Large-scale Object-based Linguistic Interactor Translator Analyser **[LG94]**), which supports their automated OOM analysis and model generation approach. LOLITA pre-processes the requirements specification to correct, simplify and normalises them by transforming passive sentence into active sentences, correcting spelling mistakes and ambiguity resolution via an in built inference engine.

The information is then subsequently transformed and stored in a conceptual graph known as LOLITA's SemNet (semantic network). This SemNet is then analysed to produce an Object Oriented Model identifying classes, attributes, operations and relationships. This approach is one of the few that is fully automated, only requiring minimal developer intervention.

LOLITA pre-processes texts to discover their morphology, syntax, semantics and pragmatics, which are defined as nodes in the semantic network (SemNet). Each node contained within is defined as either an *event* or *entity* node; simple relationships are identified as connections between nodes and complex relationships are implemented using event nodes.

Every node also has a set of control variables, but only some of these are utilised within OOM and are defined as *rank, type and family*.

- Rank gives quantification information identifying whether a node is universal, individual or a named individual.
- Family is used to classify nodes into semantic groupings to which they belong such as living, human, human organisation, inanimate and manmade.
- Type are where concepts are sorted into specific groupings such as *entity, relation or event*, thus making additional information available to the node itself, which may assist the development of the OOM.

Event nodes have frame-like structure that can represent the various components of the event itself such as the subject, the action, its transitivity, and the object; all identified from the natural language texts. This information is extracted from a deep structural analysis of the texts.

The essential classifications identified by LOLITA for this approach are *event nodes*, which are categorised in four groups: *static, cyclic, dynamic and instantaneous*

- *Static* refers to unchangeable situations;
- *Cyclic* represents recurrent temporal events;

- *Dynamic* correspond to events that span over a specific period of time;
- *Instantaneous* define events that span the shortest period of time

Overall, LOLITA undertakes all language analyses and construction of the SemNet, which contains *nodes* and *arcs* that carry additional information. The actual OOM analysis algorithm utilises the resulting semantic network from LOLITA, which aims to discover the classes, their relationships, operations and attributes based on the OMT methodology **[RPB91]**.

There are two distinct phases *(context dependent* and *independent)* during the analysis process. The role of the context independent phase is to flag nodes that represent class candidates where some are deleted altogether, whilst others are marked for user investigation as their inclusion in the model is unknown.

The goal of the context independent analysis is to extract a class candidate list to pass onto context dependent analysis, but first though it must remove or mark nodes if they are within one of the four categories: *general, superficial, system dependent or meta-knowledge*:

- *General knowledge* rules use semantic information. Where nodes that represent spatial or temporal knowledge and highest level semantic hierarchies such as *groups, something and things* are eliminated
- *Superficial knowledge* represents anaphoric references, where the references have been resolved. However, the information is no longer required by the process and is subsequently removed
- *System dependent* refers to nodes that are considered as duplicates (i.e. nodes with the same name), which are also removed
- *Meta-Knowledge* can give information to guide the requirements modelling and through consideration of the node's '*status control*' variable (identified by LOLITA) a decision can be made to either delete this node or consider it as a class candidate

After analysis through these filters they leave behind a candidate class list, which is then passed to the context dependent processor, where the decisions to create attributes or relationships are based on event node classifications {*static, cyclic*} and where operations are extracted from node classifications {*dynamic, instantaneous*}.

At the root of the approach is a threshold analysis technique which can be influenced by the user and is used to aid inclusion of candidate classes. The analysis examines the number of events a node has and if they are below the user defined level they are not included within the final model. Finding the correct balance between the thresholds and the model produced is the challenging aspect, which requires additional understanding of the requirements on the side of the user as a high threshold could lead to the introduction of irrelevant information and conversely a low threshold could miss important aspects of the design.

Finally, this approach is one of the few that also carry an evaluation of its effectiveness. The main hypothesis of the evaluation was to establish whether class model generation supported by the approach would be of a higher quality than those not using the tool. The evaluation involved a small group of students from university that were split into individual groups who had varying degrees of experience; half of the group were exposed to the tool, where the other half utilised traditional manual development methodologies.

The resultant output from the experiment, UML Models, was judged by experts who are undefined as to their status. Overall, they judged the performance based on design aspects such as classes, relationships, operations detected and on average the automated tool produced higher quality models. However, with such a small set the results are relatively inconclusive, although those who did utilise the tool preferred using it.

Perez-Gonzalez *et al* **[Per02, PKS+05]** discuss a technique towards automated analysis of requirements specifications in the pursuit of generating an object model and sequence diagrams. Their methodology is based on the proposed usage of rule posets; these are partially ordered set of roles utilised to simulate the human analysis process whilst modelling a problem.

The specification is transformed (automatically) into a subset of natural language called 4WL. 4WL has been designed for the approach to identify the subject (who is performing an action), its verb (defining actions and relational aspects), an optional object (the receiver of the action) and an optional prepositional phrase, which is to identify any possible relational information towards some other word contained within the sentence.

The 4WL transformation could be viewed as being taking a sentence of a type, which contains *subject-verb-object-object* and transforming this into *subject1-verb-object1* and *subject1-verb-object2*. Subsequently, this is defined as an SVO triplet, but how this transformation is achieved is not discussed within any of their papers. Even though the 4WL language is only useful at transforming declarative sentences; sentences that have a clear statement of intent or purpose, which have clear *subject, verb and object(s)*, it does not account for other sentence types such as *interrogative*, *exclamative* or *imperative*. In addition when the approach is presented with a non-declarative sentence type, it is not clear whether the approach will attempt any transformations or not.

The key purpose of 4WL is to answer questions related to an object in the model:

- What does the subject do?
- Who receives the action?
- Which others participate?
- When does it happen?

With these questions answered, it is possible to then generate both class and sequence diagrams of the proposed software system.

Once transformation is completed, automated language analysis can take place on the 4WL statements. This process aims to identify and extract every noun and verb contained within the sentence and then for each of these language components to have a role assigned.

Their role machine, a partially ordered set, is used to identify each part of speech and assign roles (such as *doer (subject)* and *patient (object))*, which then make it possible to generate both static and dynamic views of the system.

The role machine approach is based on the linguistic concept of theta-roles, and partially ordered sets. A theta/thematic relation aims to describe the role the noun plays in terms of the verb and identifies aspects such as the doer of the action and the patient i.e. *Susan ate the apple* where *Susan* is the doer and *the apple* is the patient.

The partially ordered sets collate this information and identify the role of the noun and its position within the each sentence. This therefore allows a decision to be made with respect to classes and attributes, where relationships and operations are derived directly from the verb. Even with the language simplification and set classifications, for a noun to be considered as a class, its probability also has to be discovered by investigating its position within the sentence and its frequency of appearance across all sentences. Only through user validation can a final decision be made to create either a class or an attribute based on the accumulation of these features.

The focus of the approach is the automated transformation of declarative sentence into their 4WL language, which then leads to automated analysis utilising their role machine to identify the importance of specific terms contained within the specification and identify *class candidates, relationships* and *operations.* However, what happens when the system is presented with other sentence types such as interrogative, exclamative or imperative? It is unclear from their work whether the sentence is manually rewritten into declarative sentence form.

The final decision to include candidates and their features *(relationships, operations),* which leads to the creation of a system model that is human dependant and highlights a potential weakness/bottleneck in the approach towards Rapid Application Development. An incorrect decision at this key stage in the process could lead to important model components being excluded resulting in an incomplete model being created.

Harmain *et al* **[Har00, HG02]** propose an approach to automated software development through linguistic analysis supported by an external tool LaSIE (Large Scale Information Extraction). Their key goal is to process uncontrolled natural language texts and generate a model of the proposed software system via assistance of LaSIE. Along with the many other approaches discussed, this approach is one which attempts to minimise developer involvement by not utilising a sub-set of natural language nor does it require human involvement in the construction of domain independent or domain specific models.

The core of their system is LaSIE, which performs all lexical pre-processing such as sentence identification, part of speech analysis and morphological analysis prior to their application of their object oriented analysis rules.

Semantic analysis is also undertaken as a part of LaSIE, after syntactic analysis. The semantic analysis develops a simple predicate argument structure based around the main verb of the sentence. A parser is used during this process, which means it may only produces a partial parse tree and not necessarily a complete parse tree structure, but what it does create there is high confidence that the resultant output is correct. This is primarily because the parser may not find a full parse for all the sentence features, it will therefore return the best and most complete parse tree that it has. The key features that semantic analysis aims to discover are the subject and objects of the sentence, plurality, voice, and time which are used in construction of a discourse model (also derived from LaSIE).

The discourse model represents a *world model extract* generated from each individual sentence structure utilising the prior analysis features. This model represents a declarative knowledge base, which contains the key background information. It identifies objects (nouns), events (verbs) and attributes (nouns). The model itself can be either specific or general, thus giving rise to a trade-off between weak support (general) or strong support (specific) for textual understanding. Overall, the approach adopts the generalised model, which allows it to be more adaptable across differing domains.

In addition, it is also possible with this model to undertake co-reference resolution and presupposition expansion, where passive sentences may be expanded to include anonymous objects to aid understanding.

With all this prior analysis and world knowledge, it is now possible to undertake object-oriented analysis. Candidate class lists and relations are derived from the discourse model based upon a set of set of rules defined for the approach. In the first instance, all nouns are considered as candidate classes, non-copular verbs (verbs expressing actions) are considered as candidate relationships and attributes are discovered through simple heuristic matching based on possessive verbs (i.e. *verb forms of have)* and their following nouns.

The key decision to create an actual class is based upon a simple frequency of occurrence and a user definable threshold. Therefore, any candidate class that is below this threshold and does not participate in any relationships is subsequently discarded. This in its own right could lead to situations where potential candidates are lost from the process because they are only stated once within the specification.

Relationship identification (simple associations) is the case of discovering the verb and its logical subjects and objects and then connecting both together. On the other hand, aggregation relations are discovered through sentence patterns defined by as *something is made up of something, something is a part of something* and *something contains something.*

Finally, the approach is also able to identify the multiplicity of the candidate classes, where the determiners are utilised. Their approach identifies the multiplicities of *'one'* based on the presence of articles (*a, an, the); 'many'* on the presence of *'all, each, every, many and some'* and specific numbers on the presence of an actual number present before the noun itself.

Overall, the approach provides a robust means of automated model generation with minimal developer input which comes in the guise of threshold manipulation. Thresholds are a strategic aspect of the approach, which are used to decide whether to create a candidate class or not. A high value set by the user could miss key aspects of the design, where a setting to low could result in the introduction of additional and irrelevant information. Given that all the prior analysis that is undertaken by the external tools and rule set; to have the final decision as to whether an element should be created based solely on a user definable value is risky.

The approach contains a robust evaluation, which stems from techniques utilised within the information extraction arena. Their evaluation investigates *recall*, *precision* and a new measure, *over-specification*, defined by the authors.

- *Recall* identifies the correct and relevant information identified by the approach;
- *Precision* measures its accuracy of the approach;
- *Over-Specification* measures additional information extracted through the process;

The corpus of software requirements specifications used in the evaluation were identified and taken from Object-Oriented Analysis and Information System textbooks. Even though this corpus contained 37 software specifications ranging from 100-1500 words in length, only 8 had accompanying models. From this, five were kept aside for a blind evaluation for the final build of the system.

The results from the evaluation demonstrate that this approach achieves high-levels of both precision (66%) and recall (73%) by comparison with a human designed model. In addition, it also generated a high level of over-specification (62%) deemed important by the approach. The additional information generated by *over-specification* and the potential loss of information (through misplaced thresholds levels) will require additional analysis to ensure nothing important has been overlooked. Even so, the approach gives a means to enhance the analysis and model generation process through an unrestricted textual analysis tool and minimal developer involvement.

Zhou and Zhou **[ZZ03]** applied natural language processing techniques to understand written requirements using an entirely uncontrolled natural language input taken from functional specifications. This process is aided by a manually developed knowledge base that aims to improve the performance of class identification.

The approach aims to identify classes, relationships, attributes and multiplicities from the written natural language texts. The key heuristic of this approach is *that core classes of the domain are always semantically connected with other classes and their attributes.* This define the foundation of the domain ontology, which aids further detection of classes, but requires a manual identification of these core classes.

Core classes are ones that the authors themselves have a high confidence in actually being a class (how this is established can only be considered to be through personal experience) and will therefore be included within the knowledge base, but this is also an activity which needs to be undertaken by an analyst. As a result, understanding of the domain would be a requirement. In addition, the knowledge base also goes to the extremes of defining a description of the candidate, vertical relations identifying possible instances, part-of relationships (generalisations) or horizontal relations identifying relatedness or similarity with other concepts contained within the domain and attributes. Essentially, the knowledge base is defining a textual version of the model, which can be accessed by the automated process.

The approach argues that candidate classes are mostly concerned with major noun phrases and minor verbs. The approach defines that *nouns are not equal to noun phrases*, and only considers nouns with pre-modifiers (such as a preceding noun or adjective) as candidates of interest. This aims to simplify the construction of the knowledge base for automated analysis. However, adjectives before a noun can be indicative of an attribute. In a similar vein, not considering solitary nouns can also lead to the loss of potential candidates as when the noun phrase identified the whole is considered during class detection and not any of its constituent parts.

Initial candidates are detected by means of *part of speech (POS)* analysis and sentence parsing. These candidates are then transformed into refined candidates that have been identified as being contained within in the knowledge base and indicate key features of the specification. POS analysis is aided by an external part of speech analysis tool, i.e. Brill Tagger **[Bri94]**, where the Brill Tagger tags each individual word with its likely POS such as Nouns, Verbs, and others. There is an additional step utilising WordNet to refine the candidate's name such as removing pluralisation. Therefore, if the candidate is contained within both the knowledge base and WordNet, it is then known as a refined candidate. Although, when the candidate is contained within WordNet, but not within the user's knowledge base, it is unclear whether this would be considered as a refined candidate or not.

With the refined candidate list, it is then possible to perform relationship detection, which utilises a link parser, where the Link Grammar Parser generates linkage information in a similar vein to the approach by Nanduri and Rugaber, **[NR95]**, this is also used to detect the individual parts of speech for candidate class detection. The key to relationship detection is linkage distance; that is, given the candidate class, how many links are there between this and other potential candidates contained within the sentence. Therefore, a relationship is only considered to exist when the linkage distance is less than or equal to three. This approach is only investigating the subject of the sentence and associated objects, which connect through the main verb of the sentence. The approach does not consider whether it is appropriate for the relation relationship to exist or not.

The next step in their processing is consideration of attributes, which utilises a 7-tuple linguistic pattern to determine whether the concept (class) is a property concept (attribute), though it is unknown if this process is manual or automated. Nevertheless, the basic premise exists that if there is only one property associated with the class then it is an attribute, otherwise it should be a class. For example, the authors state if we are only interested in obtaining the *price*, then this is just an attribute. However, if we are interested in maintaining information about *price, discount, and effective date*, then *price* is a class.

Finally, parallel structures, which naturally exist in the language, aid identification of additional attributes, missed relationships and classes. The parallel structures considered are individual words such as *and or* and phrases such as *both...and..., as well as, either or...* Therefore, if one of the elements contained within the parallel constructs has a relationship with an already identified concept or property contained within the sentence, the other element within the parallel construct is also considered to have has the same relationship or is considered as an attribute but this is dependent on the concept/property under consideration.

The approach is entirely dependent on manual analysis to extract and identify candidates to reside within the knowledge base. The subsequent class detection and refinement process in conjunction with WordNet relies on the knowledge base and only considers Noun-Noun or Adjective-Noun structures during the refinement process. As a result, there is the possibility to overlook individual nouns that may identify a candidate. In addition, adjective pre-modifiers attached to the noun are more than likely to indicate an attribute of a potential class rather than representing a class itself.

However, a key strength of the approach is linkage distance utilised to discover relationships, rather than identifying a relationship because a verb is present. Furthermore, the usage of parallel structures to resolve or identify some the unknown/unresolved features contained within the language is an interesting approach, but one which could lead to dubious results if the original feature has not been correctly identified.

Finally, the approach itself does not conclude with the creation of UML models, but rather descriptions of the features and their likely types such as classes, relationships, attributes and there is no evaluation of the approach.

Ilirva & Ormandjieva **[IO05, IO06, OI06]** discuss a methodology utilising unrestricted natural language in a three-phase approach towards a partially automated analysis. The three phases consist of a linguistic component, semantic network and OO model generation.

The linguistic component processes each individual sentence by means of an external tool, MBT-Tagger **[DZB+06]** and identifies their applicable parts of speech such as, nouns and verbs. The information extracted from the first phase of linguistic analysis is utilised to determine the three function roles such as subject, object or verb contained within the sentence. The SVO triplets are also split into individual groups such as the subject predicate and object groups through a manual process and represented in a tabular form. The purpose of the tabular representation is to form a knowledge base, which details each individual triplet, its language component (noun/verb) and overall sentence type (main, conditional or conjunctive) defined as follows:

- The main sentence is likely to contain candidate classes and relationships
- The conjunctive (and/or) sentence type identifies the potential relationship between other words.

- The conditional sentence may introduce pre/post conditions for candidate features that should also be considered.

The tabular representation aids in determining the connections between the words and the construction of a semantic model, helping to discover actual relationships between each component. This model is also depicted in a graphical notation.

The key aim for the semantic network is to represent the features contained with specification and tabular representations, where nodes are nouns (subjects/objects) representing classes and relationships are represented by verbs or as arcs between the nodes. Additionally, prepositional words and words with possessive endings are also transformed into relationships with their respective counterpart.

The semantic network can then automatically be transformed into both Use Case and OO representation, where nodes represent classes/actors, arcs connecting nodes represent relationships and where verbs in active form also identify the operations of nodes.

The key strength of this approach is utilisation of unrestricted natural language (NL) and the transformations from NL to tabular representation to a semantic model and subsequent automated software models. However, their methodology requires involvement in the initial transformation process to extract the SVO groups and transformation into their tabular knowledge base. In addition, the requirement of human involvement, understanding and effort to make the correct decisions during the transformation process between tabular and semantic models is of utmost importance. These decisions require careful consideration as it is the semantic network that is transformed into both software model types. An error introduced early during transformation phase will ultimately propagate into the final models themselves.

The approach does conclude with an evaluation; however, there is no worthwhile information that can be extracted. They state that it has been evaluated against other similar systems that translate natural language in a formal model, resulting in no difference between the results. There is no indication of any imprecision or incompleteness from the analysis, but there is no result data presented. There is no formal evaluation method presented and as a result, the findings are considered unreliable.

Sampaio *et al* **[SRC+07]** introduce an approach towards automation within the domain of aspect-oriented requirements engineering. Their tool, EA-Miner, is used to support the costly manual analysis phase of requirements engineering through the creation of viewpoints which could be transformed into UML Models but is a feature that is not considered nor discussed by the authors. Furthermore, the authors also state that the goal is not to replace the requirements engineer but to help them save time and focus upon the key information.

The approach is broken down into 4 key phases: eliciting requirements from customers, identification of model concepts, structuring of the specification and validation/resolution of requirements and conflicts. Phase 2 of the approach uses a combination of NLP and rule based techniques to identify model abstractions such as viewpoints from nouns and use cases from verbs which are automatically mined from the requirement documents. Every concept identified by the approach is considered to be candidate. Phase 3 gives the ability to refine the candidates identified during the previous phase where it is possible add, remove group abstractions into identified viewpoints or use cases although it is unclear whether this is an entirely manual process. Nevertheless, the authors do state that the approach is not aimed at 100% automation implying that that phase 3 may be partially automated. In addition, the approach gives the ability to filter the results based upon thresholds, stemming and synonym lists all of which aid the engineer during the refinement phase and also offers practical guidelines and best practice as well.

NLP processing is supported by an external tool which uses a corpus based approach to language analysis which identifies all relevant parts of speech from nouns, adjectives through to verbs, but only nouns and verbs are considered during the creation of viewpoints/Use cases. In addition to NLP processing, the external tool has the ability to semantically tag words and sort them into related groups. This is achieved through analysing the context for the phrase in which the word is used and is the core feature utilised when generating viewpoints/use cases.

Overall the approach does not create a UML model of the system but presents the viewpoints and all related requirements to the user from which a model of the system could be manually created. The approaches focus to considered only nouns could lead to a loss of information that is important to other aspects of the design such as attributes, relationships and operations. However, a novel approach is the identification of non-functional requirements from keyword lists which is an important feature of any software system and is not considered by any of the related works. The works concludes with an evaluation of both a time based analysis and a measure of the precision and recall vs a human expert. Users that use EA-Miner to create views were on average 130 times faster in comparison to the manual analysis resulting in a vast time saving. When considering the accuracy of the approach and when user knowledge of the system is not taken into consideration the approach is comparative to the human in terms of recall but performs poorly in terms of precision. However, when the user expertise is utilised the results both recall and precision outperform the human analysis greatly. Nonetheless the overriding disadvantage of this technique is the lack of final model creation and the user involvement and understanding required to achieve the best results.

2.4 Conclusions

There are many approaches which differ in complexity and novelty, but all have a common goal to develop ways to simplify and reduce effort, and enhance the software development process through semi/fully-automated analysis techniques.

All of these works follow well-defined steps as they are all attempting to simulate the human analysis process, defined as follows:

1. Textual Specification Acquisition
2. Natural Language Analysis
3. Knowledge Extraction
4. Model Generation

The review of the literature explores how these steps have been semi/fully automated and presents the key strengths and weaknesses of those approaches. It would appear that they all perform exactly the same process, utilising some means of language analysis, where all the relevant parts of speech are identified such as nouns, verbs and adjectives, which are then subsequently mapped to *classes, operations, relationships* and *attributes* via some technique. However, there are differences that set the approaches apart; the most notable can be defined as follows:

1. Requirements Specification Manipulation (prior to semi/fully automated analysis)
    - Controlled Natural Languages (CNL)

        - Manual/Automated transformations
    - Language Simplifications
        - Manual
    - Specification Languages
        - Manual

2. Model Feature Detection Techniques (during analysis)
    - Domain models
        - Manually/Automatically defined
    - Threshold based analysis
        - Automatically defined, manually set
    - Rule based analysis
    - Semantic consideration

For the majority of the approaches human involvement is still a key requirement throughout the process, which is demanding and in some cases profound. Manipulation of the requirements specification can lead to a situation where a full rewrite is necessary or a transformation into some controlled/specification language prior to any automated analyses is required.

The transformation process is an error-prone task that could result in the loss/exclusion of information because there is no procedure to validate the transformed document. The potential loss of information is of no fault of the human, but due to the need to transform the requirements in order to simplify the automated analysis procedure. When considering the manual transformation process, it is also possible for a situation to arise where unconscious bias, a common characteristic, is applied unknowingly and for information to be disregarded; information that may be vital and that should be maintained within the specification. In addition, automated transformations can also lend itself to this bias, *not unconsciously*, but by design. They too can disregard information, such as compound-nouns, adjectives and prepositions all of which create greater understanding of attributes and relationships towards candidate class and their creation.

The sole purpose of specification manipulation is to reduce the original requirements into a form that is tailored to the specific automated analysis technique for *class/relationship/operation/attribute* detection process. Specification transformation is not a requirement of all approaches and only a minority use the full-unadulterated specification **[Mic96, MMZ02, MG02, KZM+04, Har00, HG02, ZZ03]**

Where domain models are used as a feature detection techniques, the domain models created come in two flavours: manual or automatic. In a manual context, the models can simply be utilised to avoid duplication of class creation or other model features, but more commonly, they are used to the extent of defining a textual representation of the resultant model **[Bry00, LB02, LB02a, LB02b, LB02c, LB03, BLC+03, ZZ03, CHK07, IO05, IO06, OI06]**. This textual representation can detail anything from candidate classes, relationships, operations, class hierarchical structures and attributes. All of which require an understanding of the actual requirements and do not consider the actual manual effort required to define these models.

Automatically constructed domain models **[Mic96, MMZ02, MG02, KZM+04, Har00, HG02]**, built from the utilisation of external tools such as LaSIE and LOLITA, aim to address the issues associated to manual interventions. The domain models generated construct semantic networks that can be used to reduce ambiguities and identify key features of the model alongside other refinement techniques. The key benefit of these techniques is the absence of human intervention and therefore does not distract the user from the task at hand. However, in one case **[Har00, HG02],** the model generated is not taken advantage of during the final decision making process, which is left to a user defined threshold analysis.

Finally, semantic analysis is a key consideration throughout the approaches as well however, the interpretation identified in related works leads to little consideration of actual semantics in the sense of the study of meanings. Semantic considerations that are utilised in some cases are typically part of external tools such as LaSIE and LOLITA. In addition, the resulting semantic analysis is typically not utilised in the final analysis step, where it would be most crucial to consider during candidate feature detection.

In the majority cases, semantic analysis takes the form of utilising WordNet either in an automatic or manual effort. In both automatic and manual efforts, WordNet is only utilised to identify duplicate words or words that have similar meanings therefore, allowing the removal of words which could result in the creation of erroneous candidate classes. However, WordNet itself contains much more valuable information and if harnessed through a novel technique it could be used to better the automated model detection process.

Despite these issues all of the approaches, in some way or another, take steps towards an automated analysis approach with the key goal of improving the models generated through some means of automated language analysis.

However, consideration of the term *automation* either fully or semi, is meant to reduce the effort required by manual analysis, enhance quality and allow for rapid application development, but in majority of cases this is not apparent. It is not apparent due a shortage in quality evaluations.

The majority of evaluations undertaken only investigate the qualitative aspects of the resultant models. This is important feature to evaluate, because producing results of a low quality serves no purpose and will hinder the overall process. However, it is only a minority of approaches **[Mic96, MMZ02, MG02, KZM+04, Har00, HG02]**, which actually undertake any meaningful form of formal evaluations. Other evaluations are limited by not having a well-defined and or stated methodology, which makes it difficult to validate those results.

Only one of the evaluations actually investigates the actual reduction in effort offered by these approaches **[GB94]**, but uses a small set of domain experts and a non-domain experts using the automated analysis tool and compares how much faster automation is. The results from this are interesting, but are limited to only finding key abstractions contained within the specification rather than actually creating a model from the specification.

Overall, the review of the literature has identified an important question: *"To what extent does analysis of an un-restricted natural language specification contribute to a 'better' first-cut design through means of a deep syntactic and semantic analysis?"*

Chapter 3
Techniques towards Automation

3.1 Introduction

Chapter 2 introduced the related techniques for automated software requirements specification (SRS) analysis and model generation. This identified that these fall within two categories: Semi or Fully Automated implementations.

The main issues identified from these approaches are as follows:

- Excessive Manual Effort
- Manual rewriting of the software requirements specification (SRS) document prior to automated analysis such as:
    - Simple Sentence Constructs (Subject Verb Object (SVO) triplets)
    - Controlled Natural Language
    - Specific Specification Languages
- User defined candidate extraction rules as a pre-requisite
- Manual Domain Model Generation
    - Excessive Detailing; relationships, classes and attributes
- Negligible Word Semantic Consideration
- Limitations of Threshold Based Analysis

The aim of this chapter is to address these key issues through both semantic and syntactic analysis of a free-form natural language requirements specification. The automated method presented here will analyse the syntactic structure of every sentence; determining its clausal structure, identifying each part of the sentence (subjects, predicates and objects) and considering every individual part of speech.

Along with a syntactic analysis, the approach also considers the semantics of each term (i.e. an individual word contained within a sentence), utilising a generalisable rule based and algorithmic approach in its decision-making process. As a result, it is possible with this technique to uncover:

- Classes
- Relationships
- Attributes
- Operations
- Parameters
- Multiplicities

The solution is not meant to replace the analyst/developer; it aims to allow them to concentrate on the important aspects of the overall software design such as flexibility, evolvability, maintainability and its implementation.

It is therefore the goal of this approach to use the techniques discussed to:

- Use unrestricted natural language requirements specifications
- Emancipate the analyst/developer from the manual analysis and model generation process
- Aid conceptualisation of the specification, as a first step towards model generation
- Reduce excessive analysis & time effort
- Introduce maintainability features through inheritance hierarchy construction

A prototype implementation of the proposed approach has been developed which currently identifies all these model features *(classes, relationships, attributes, operations, parameters & multiplicities)* from an unrestricted natural language specification. In addition, the prototype also includes additional features to assist understanding of the resulting analysis and inclusion of best design practice, which can be defined as follows:

- Traceability Links
    - Allowing discovery and tracking of model features from the resultant model design back directly into the language contained within the specification, thus enabling an understanding of from where and why the specific feature was generated.

- Integration of widely accepted best design practices such as *'programming to an interface rather than an implementation'*, introducing design flexibility

The remainder of this chapter starts with a view of the approach from a high-level and then proceeds to discuss the approach techniques that make it possible to automatically analyse a natural language requirements specification and create first-cut design. The chapter then concludes with a review of the key issues that can impede the automated creation of a UML model.

3.2 Approach Overview

The techniques discussed in the subsequent sections 3.2.1 and 3.2.2 are intertwined with one another, where the Common Semantic Model (CSM) is reliant on the Syntactic Analysis Model (SAM) and vice versa.

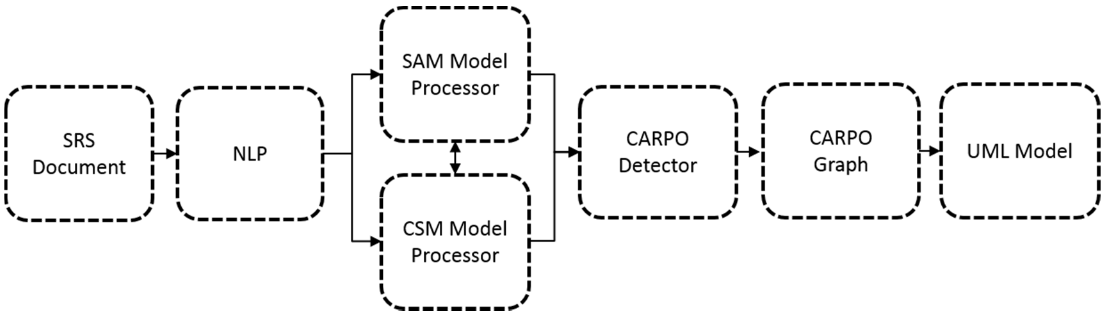


Figure 3.2-1 Automated Software Architect Automation Process

Figure 3.2-1 demonstrates a high-level view of the Automated Software Architect (ASA), which uses a Software Requirements Specification (SRS) prior to it being transformed into a formal SRS document, which is written in natural language. This is then processed by a Natural Language Processing (NLP) toolkit OpenNLP and the subsequent information obtained from this analysis is utilised by both the Common Semantic Model (CSM) and Syntactic Analysis Model (SAM) models allowing the identification of the relevant UML model features. The CSM is used instead of a user-defined knowledge base allowing identification of candidate UML features, where the SAM is used to extract candidates from the specification and is used in conjunction with the CSM Model where the CSM is reliant on the SAM and vice versa.

The goal of specification modelling is to simulate the human process through collation of the information contained within the specification on a sentence-by-sentence basis. It is similar to the human process of software modelling by means of considering both the semantic and syntactic features to identify model candidates. This is achieved through an intertwining of both semantic and syntactic analyses, where word semantic classifications and the syntactic understanding of sentence constructs/structure are utilised to extract the correct and relevant candidate features from the written specification.

Once the information has been extracted from the specification, it is presented for analysis and the relevant features decided upon by both the CSM and SAM models are then maintained in local memory for later usage until sentence processing is complete. The resulting analysis is stored within the Class, Attribute, Relationship, Parameter, Operation (*CARPO*) graph, which is subsequently processed to generate the class/UML model of the proposed software system.

The CARPO graph is simple storage structure used to track all candidate features and maintain the integrity of the approach primarily by avoiding the creation of duplicate information. At the heart of the CARPO graph lies nodes that can represent candidate classes or attributes. The linkages between each node can represent relationships, operations, and identify parameters of operations; relationship links also carry additional information such as multiplicity or the type of relationship (i.e. generalisations and associations). In addition, all nodes and linkages carry traceability links back into the specification document. This satisfies the requirement to identify where features of the model have been discovered and leads to a better understanding of the automated decision making process.

The resulting UML modelling is constructed by traversing the CARPO graph through inspection of its nodes and linkages and by utilising the UML2 plug-in for Eclipse; it is therefore possible to transform the graph into its graphical representation.

3.2.1 The Common Semantic Model

The goal of the semantic model is to extract as much information from the text contained within the specification, without the need for any user defined back-end domain knowledge. The CSM technique manifests itself in terms of Common Sense Understanding (CSU), which raises two interesting questions:

- *What is Common sense?*
- *How can common sense be utilised to identify the relevant features of a class/UML model from natural language?*

Simply put, common sense is the consideration of what the general population would commonly agree upon based upon their common understanding, similar experiences and knowledge developed over a lifetime of everyday interaction with the world.

It therefore considers whether the features contained within the specification should be included within the initial class/UML model. A feature can be a *class, relationship, attribute or operation* and its decision for inclusion is based upon the common sense semantics of a given word/phrase, identified through the syntactic analysis of the natural language texts.

The common sense model constructed for the ASA refers to information retrieved from the external semantic/lexical dictionary, WordNet **[Mil95]**. Under the direction of George A Miller, WordNet is a large lexical database of the English language, which is a dictionary of *nouns, verbs, adjectives and adverbs.* These are grouped together as sets of cognitive synonyms, each expressing a distinct concept. The resultant network of words and concepts are a direct result of a manual human consideration (i.e. WordNet has been built manually rather than automatically), where constructed synonym sets are interlinked through conceptual-semantic and lexical relations. Even though WordNet provides a plethora of useful information, only the semantics are utilised during the actual processing of a given phrase or term and is where the common sense semantics are obtained.

Within WordNet, a given word may have a list of different semantics for differing contexts, but these are ordered by their most commonly agreed upon meaning *i.e. common sense understanding*. Where the first sense of the word contained within the dictionary is considered the most commonly understood meaning. Therefore, when WordNet is queried for the semantics of a particular word, it returns a list of all the senses associated to that term in order of most to least commonly understood **[Fel98]**.

For Example: a search for '*Shelf*' yields two semantics see Table 3.2-1.

Table 3.2-1 Example Noun Sense Classification

Sense	Semantic	Term	Description(sense)
1	noun.artifact	Shelf	A support that consists of a horizontal surface for holding objects
2	noun.object	ledge#1, shelf#2	A projecting ridge on a mountain or submerged under water

It is the only highest frequency *(i.e. the most commonly understood or common sense definition of the term)* semantic, *sense#1*, which is used to make a decision regarding the creation of a model feature. In contrast, a search for *ledge (sense#2, ledge#1)*, which has a similar meaning to *shelf* but contextually different, would yield a result containing only one sense - that being *noun.object*. See Appendix A.1 for a list of all semantic definitions for both verbs and nouns.

Under no circumstances is any form of disambiguation or surrounding context taken into consideration during the lookup process. This may seem counterintuitive not to consider surrounding context and to disambiguate, but is a key issue which is addressed within section 3.5.4 in the context of the *Software Requirements Specification Issues in the Context of Automated Software Development.*

3.2.2 The Syntactic Analysis Model

In addition to WordNet, the external tool OpenNLP **[Mor07]** is the means behind the syntactic analysis of the natural language software requirements specification. It automatically parses natural language texts and returns a syntactic parse tree of the sentences contained within the software specification, which is later utilised in the automated detection process.

The OpenNLP tool is based upon Ratnaparkhi's Ph.D. dissertation **[Rat98]** that demonstrates how to apply maximum entropy models to various natural language problems in pursuit of the relevant syntactic structure. The toolkit itself has many features such as sentence identification, tokenisation, chunking, name finding, co-reference resolution and full part of speech (POS) tagging. However, the key features utilised by the ASA are sentence identification, full POS parsing and co-reference resolution. OpenNLP itself has an accuracy rate of 96% for unseen data **[Rat98]**.

Through support from OpenNLP, the sentence structure/parse tree is obtained. For example, given the sentence, *The Company operates both individual taxis and shuttles,* OpenNLP returns its analysis of the sentence in the form of a parse tree - see Figure 3.2-2.

It is then possible to traverse this tree and identify individual parts of speech (POS) such as *nouns* (NN), *verbs* (VB), *prepositions* (PP) and more (see Appendix A.2) from each individual sentence. This information is then subsequently utilised to identify candidate *classes, attributes, operations, relationships* and *multiplicities* in conjunction with the semantic analysis.

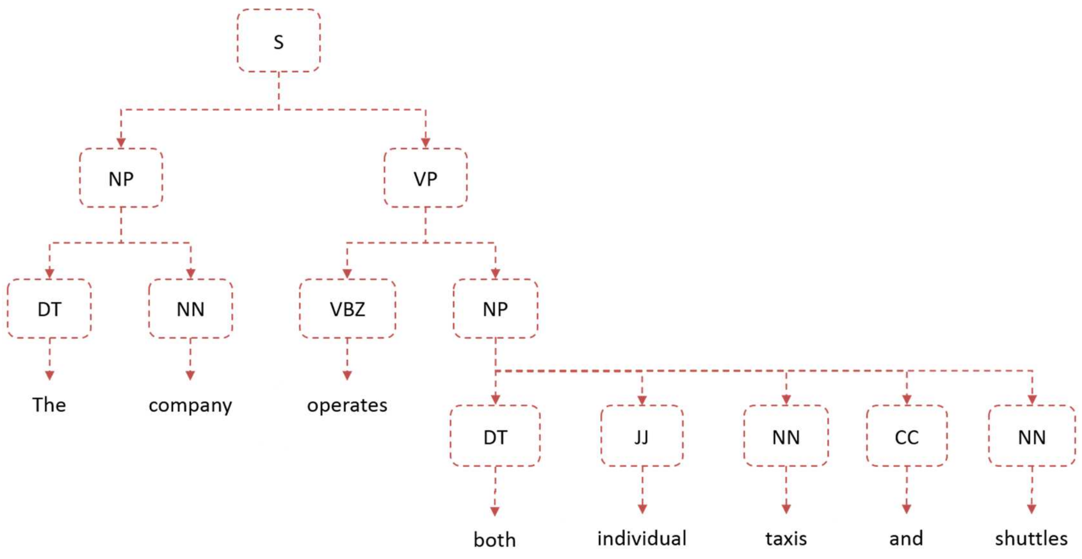


Figure 3.2-2 Part of Speech Parse Tree using OpenNLP

Along with the individual parts of speech identified each of the words within the sentence are also reduced to their base stem. This is a feature that is not supported by the OpenNLP Library and the Porter stemming algorithm **[PRR80]** is used to identify word base stems. The key reason to reduce each word to their base stem ensures that no duplicate model feature will be created for the same word that may be used with varying inflections. For example in Figure 3.2-2 the words 'taxis' and 'shuttles' will be reduced to the following base stems, 'taxi' and 'shuttle' thus ensuring any reference to either 'shuttle' or ' taxi' will not result in duplication

In addition to the individual POS components and base stems, it is also possible to extract structural information obtained from the parse tree, which aids in the discovery of additional UML features. Features such as classes, relationships, operations (including parameters and placement), multiplicities, class hierarchical structures and attributes are only discoverable through the syntactic relationships

The key to the syntactic analysis is a model that identifies the grammatical constructs that we use every day to understand and infer what is contained within written texts. This analysis subsequently allows for the identification and creation of an initial/conceptual UML model of the proposed software system, extracted just from the natural language software requirements specification (SRS).

The following sections build upon and discuss the intertwining of both semantic and syntactic models that extract, analyse, decide and deliver an initial UML model from a textual requirements specification.

3.2.3 Rule Derivation

The identification of the rules that guides the ASA through its decision making process can been considered to have its roots both within the noun phrase approach **[Abo85]** and common class patterns **[Bah99, RBP91]**. A set of specifications was utilised during the identification of the rules and are defined in Appendix B.6. These specifications where chosen for their domain diversity: payroll, aircraft, video store, music store and medical systems – they do not have any associated UML models, it is also considered that they would contain a representative range of different syntactic and semantic features.

The syntactic analysis model acts as the basis for finding the key candidates that should be considered as either a class, attribute, relationship, parameter operation or other UML feature based solely on their syntactic type: nouns, verbs, and adjectives. Whereas the semantic analysis model is charged with deciding that if the candidate exists within a specific set of semantics then the combination of both syntactic and semantics will imply some relevant UML model feature.

The semantics sets identified from WordNet **[Mil95]** which leads to UML feature discovery were considered and identified by the author from their generalised description and domain as they are defined within WordNet. For example the semantic domain of type *artefact – manmade objects* for nouns was thought to be a group of items that would of importance to a model and should therefore be created as class candidates.

The general strategy taken towards rule identification was to first process the each of the training specifications utilising just the individual parts of speech such as nouns and verbs which lead to the extraction of set of all possible candidate classes, relationships and operations. The sets of candidates were subsequently cross-referenced against their semantic sets which allowed candidates that where not contained within a defined semantic set to be omitted allowing the remaining candidates to be considered as either classes, relationships or operations. The initial set of rules were derived from this process. The more complex rules associated to the identification of multiplicities, operation placement, parameters, class hierarchies and relationships where identified from the syntactic structure and constructed from the study of grammatical constructs **[Kie09, Kie09a, Kei09b, Kei09c, QGL+].** The rules identified for these features were then run against the set of test specifications to validate their generalisability.

Overall, 28 core rules have been derived that best generalise the identification of the core UML features such as classes, attributes, relationships, parameters operations and multiplicities from their individual parts of speech, syntactic structure and semantics. The following sections of this chapter discusses the approach at a greater depth of detail.

3.3 An Interwoven approach featuring Semantic and Syntactic Analysis for Model Extraction

This section reports on the techniques used in generating a UML model from a natural language specification. This will review the core components of the sentence such as the noun and verb phrases and will undertake a top down view of each individual component contained within those phrases, reviewing their key syntactic and semantic relationships. The discussion will then conclude with a view of additional modelling features that can be extracted during the process from sentences and clausal structures.

3.3.1 Clause and Sentence Structures

The main starting point of any automated analysis is the sentence itself. Every sentence has a structure that can be classified into three distinct types; *simple, compound* or *complex*. These can be further decomposed into clausal structures being of either *dependent* or *independent* types. Both clausal components contain a *subject* and a *predicate*, which is a combination of different phrases (for instance nouns can be *subjects* or *objects)* and the predicate is formed by a combination of verbs, objects, prepositions, adjectives and/or adverbs.

In its most common form the structure of a sentence can be defined as follows: -

1. A sentence is composed of different clause types (Independent or Dependent Clauses) where:

2. A clause is composed primarily of a subject and predicate where:
3. A subject contains a noun phrase and:
4. The predicate contains both a verb phrase and an object(s) where:
5. The object(s) contains a noun phrase and can be of type: direct (affected by verb's action) or indirect (receives the direct object).

Figure 3.3-1 demonstrates the overall structure of sentences, clauses and clause components.

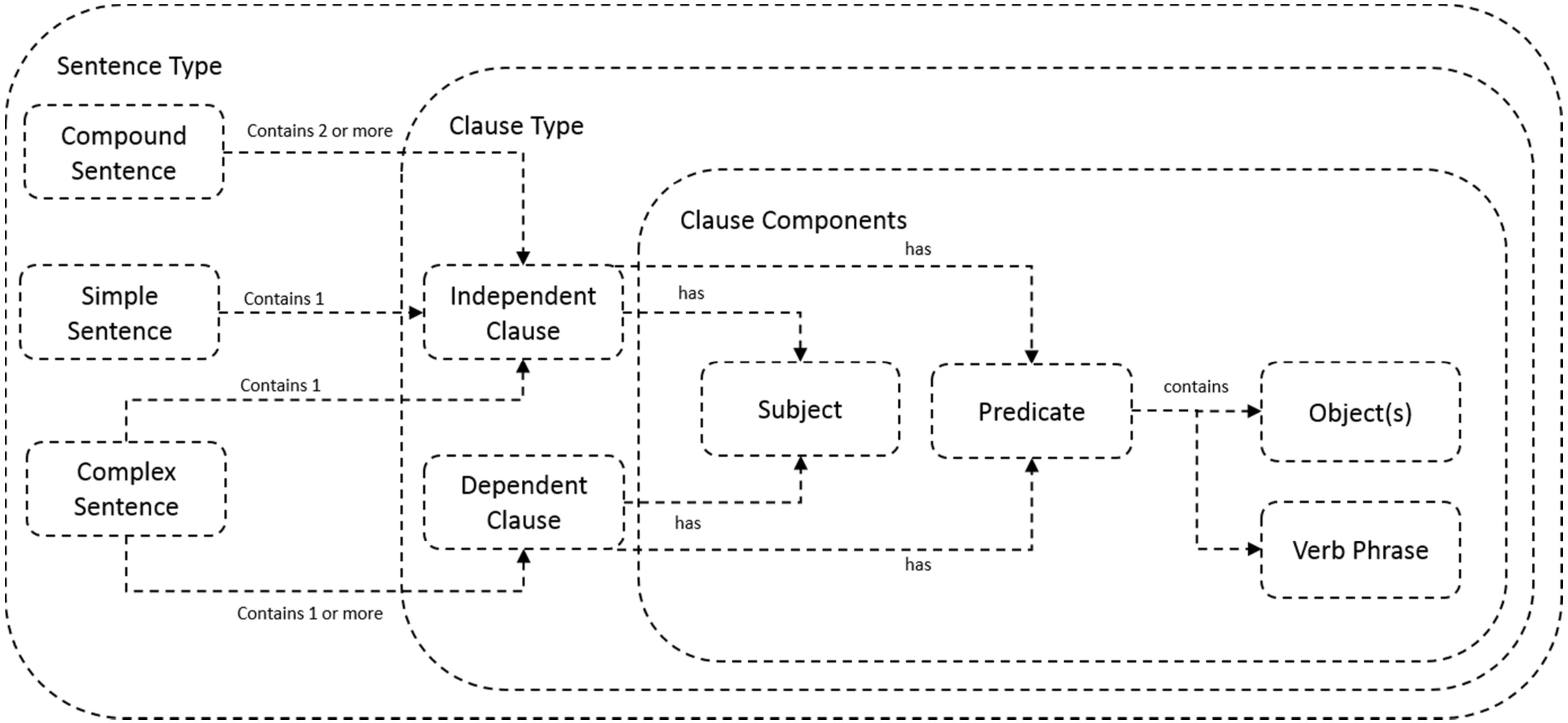


Figure 3.3-1 Sentence and Clause Structure Components

3.3.2 The Noun Phrase

The noun phrase (NP) is a phrase based on a noun, pronoun or other noun-like word (nominal). It is the most common unit in sentences and examination of the noun phrase structure itself provides an abundance of useful information that can be extracted and directly related to the features of a UML model.

The noun phrase such as, *The PhD student,* consists of a head noun - a unit of speech used to identify any class of people, places and things *(the common noun),* or to name a particular one of these *(the proper noun),* which can be optionally modified through pre- and post-modifiers.

Figure 3.3-2 defines the structural components of a noun phrase and its UML mapping describing what is possible to identify, namely:

- Multiplicity *(from determiners),*
- Class Hierarchical Structures and Attributes *(from pre-modifiers),*
- Class candidate *(from the head),*
- Relationships, Parameters, Actions or State *(from post-modifiers),*
- Attributes (see 3.4 Additional Modelling Considerations) *(from the noun head)*

All of the above features are achievable automatically from its syntactic structure and in conjunction with the semantic model.

The remainder of this section will concentrate on the following detection techniques:

1. Class detection and the Noun Phrase Head
2. Multiplicities from Determiners
3. Attributes, State and Class Hierarchical Structures from Pre-Modifiers
4. Relationships and Operation Parameter consideration from Post-Modifiers

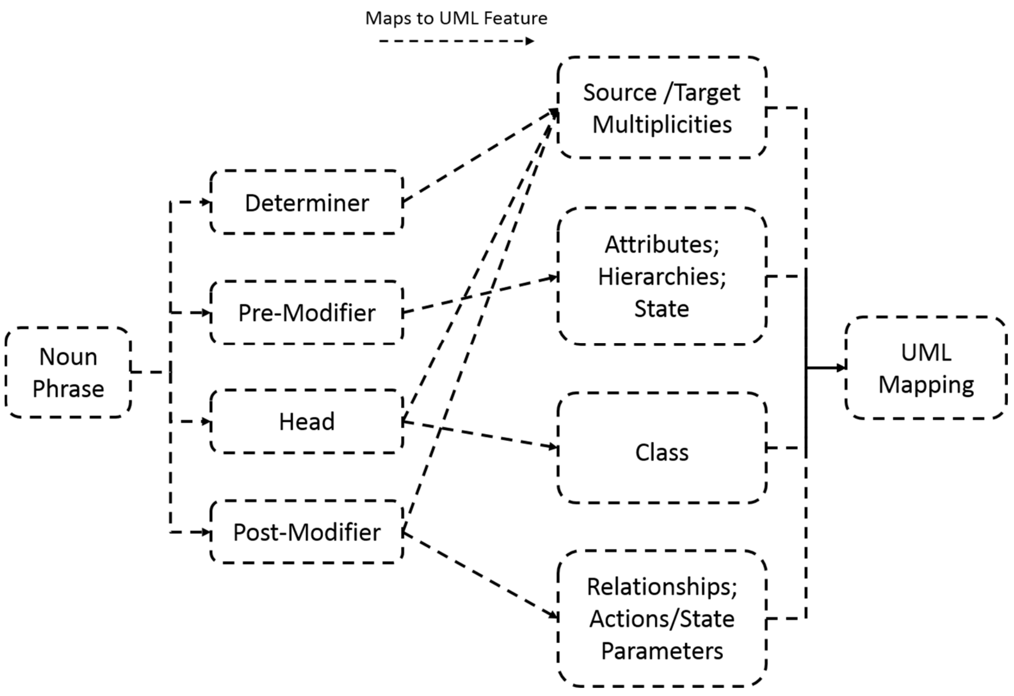

Maps to UML Feature
Noun Phrase
Determiner
Pre-Modifier
Head
Post-Modifier
Source /Target Multiplicities
Attributes; Hierarchies; State
Class
Relationships; Actions/State Parameters
UML Mapping

Figure 3.3-2 Noun Phrase Structure to UML Mapping

3.3.2.1 Class Detection and Noun Phrase Head

Syntactic Considerations

The noun's role within the sentence is variable; it can perform the function of a subject, be the object, a complement to either the object or subject based on the verb's definition, or if it is contained within a prepositional phrase, the preposition's complement. The noun is identified through OpenNLP Part-of-Speech (POS) tags *(NN (non-plural), NNS (plural form)* and *NNP (proper noun))*.

Subject Definition: The *subject* represents the actor of the sentence, i.e. the one performing the action associated with the verb. In terms of modelling, this represents a candidate class as it identifies *people, places* and *things* or other additional class model features such operation placement, relationship start point and multiplicity consideration.

Object Definitions: There are two key definitions of sentence objects; the direct object which identifies/answers *What?* in relation to the verb; the indirect object which identifies/answers the questions *To Whom?/ For Whom?* and is the recipient of the direct object.

Subject and Object Identification: The subject of the sentence is identified by the rule based on first introduction. That is, the first noun discovered within the sentence is considered to be the subject, where all other nouns are considered as objects, without differentiating their type (direct or indirect) - see Figure 3.3-3. This is a technique also used in related approaches as well **[Bry00, LB02, LB02a, LB02b, LB02c, LB03, BLC+03, LDP04, LDP05, LDP05a]**.

It is significant to note that the first noun identified in the sentence may not be the true subject of the sentence. This occurs only when a sentence is in its passive form, which is controlled by the verb of the sentence. The first noun syntactically is the subject of the sentence, but in its passive form, it receives and does not perform the action of the verb. In this case, the sentence object is performing the action. The passive construct is revisited later (Section 3.3.3) as it is related more to the function of the verb and operation placement rather than the noun itself.

The first introduction rule implies that the first noun discovered within the sentence is considered the most likely candidate to be the Subject of the sentence.

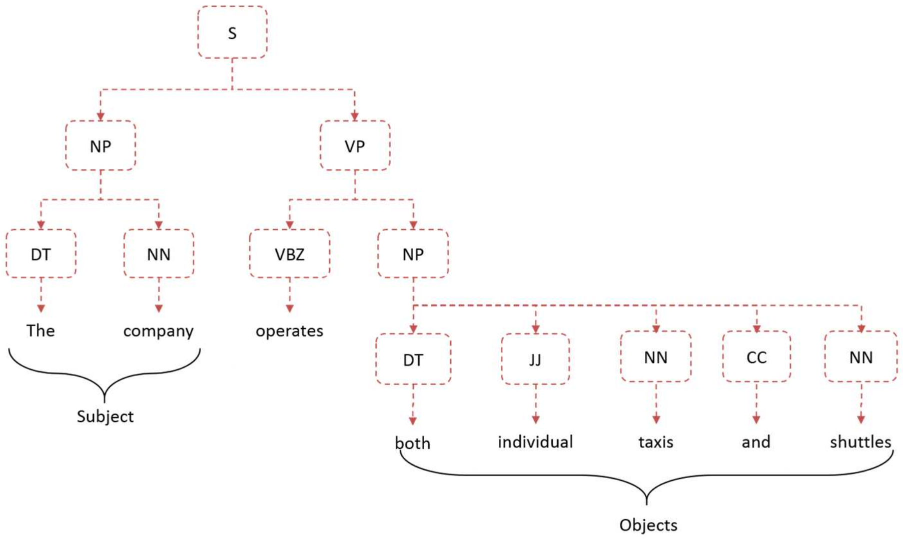


Figure 3.3-3 Example Subject, Object Identification

In the majority of cases, the sentence construct is *Subject→Verb*, where → means followed by, but in some it is possible to have *Subject-Inversion,* defined by the construct *Verb→Subject*. However, for each sentence that contains *Subject-Inversion*, it is specifically tagged by OpenNLP (see Appendix A.2), which makes it possible to easily identify this structure. In addition, subject-inversion still lends itself to the prior rule of *first introduction*, thus allowing efficient resolution of the sentence's key subject.

Semantic Considerations

With both subject and objects of the sentence identified as class candidates, the final decision as to whether a class should be created or not is based upon two key features: the presence of a *noun* detected through syntactic analysis, and the noun's set of semantics obtained from WordNet.

WordNet contains twenty-five noun semantics, which have been classified into groups by the WordNet authors that best define a noun's semantic type, given a specific context (see Appendix A.3). As previously discussed, this is where the Common Semantic Model meets the Syntactic Analysis Model and the two techniques come together to allow informed decisions to be made regarding the creation of a class from the candidate noun under evaluation.

The classification of these semantics in terms of class modelling implications has been defined by the author through a manual consideration of the individual nouns and the semantic descriptions defined by WordNet. The class modelling classifications allow automation to be completely domain independent, requiring no further manual assistance in the area of domain models or specific domain rules. This is partly due to the Common Semantic Model and the initial manual classification of the WordNet semantics, the majority of which are tangible and most likely define an aspect one would wish to model.

Table 3.3-1 defines the semantic, their description (defined by WordNet), and states the feature they most likely represent within a UML model (defined by the ASA). The class modelling classification has been achieved by reviewing the nouns contained within the semantic sets and considering their semantic description.

In addition to the 14 semantics, some of them (*animal, person, plant* and *shape),* also identify hierarchical structures where an *abstraction* construct is deemed beneficial to include within the UML model. This means that when a noun also has one of the hierarchical semantics, an *abstraction* will is also created. This is helpful to include because when the analyst is presented with the initial model, it aims to provoke their thoughts regarding the maintainability and flexibility of the overall design extracted from the requirements specification.

Table 3.3-1 Candidate Class Semantics

Noun Semantic	Description	Class Modelling Implication	
		Class	Hierarchy
Animal	Nouns denoting animals	X	X
Artefact	Nouns denoting man-made objects	X	-
Body	Nouns denoting body parts	X	-
Communication	Nouns denoting communicative processes and contents	X	-
Food	Nouns denoting foods and drinks	X	-
Group	Nouns denoting groupings of people or objects	X	-
Location	Nouns denoting spatial position	X	-
Object	Nouns denoting natural objects (not man-made)	X	-
Person	Nouns denoting people	X	X
Phenomenon	Nouns denoting natural phenomenon	X	-
Plant	Nouns denoting plants	X	X
Shape	Nouns denoting two and three dimensional shapes	X	X
Substance	Nouns denoting substances	X	-
Time	Nouns denoting time and temporal relations	X	-

The rationale for utilising WordNet semantics as a means to aid class candidate detection can be justified in terms of both the Common Class Pattern (CCP) **[Bah99]** and Noun Phrase approaches **[Mac01]**. The key aim is to simulate these manual processes (automatically) in conjunction with the knowledge from WordNet and with the assistance of syntactic analysis. The manual noun phrase approach aims to identify the nouns contained within the specification, where the candidates are sorted into classifications of *Relevant, Irrelevant* and *Fuzzy*. Both the CCP and Noun Phrase approaches provide initial guidance and are heavily reliant on the developers' understanding to identify candidate classes. Therefore, utilisation of the semantic information contained within WordNet in an automated context aims to simulate this manual knowledge extraction process towards candidate class detection. This is summarised in the following rule:

Rule 1 – Class Detection

If a noun's most common semantic belongs to the set of candidate class semantics, then that noun is a candidate class

3.3.2.2 Non-Candidate Class Semantics and Noun Phrase Head

Not all of the noun semantics imply that a class should be created – only 14 out of 25 are considered to indicate a candidate class. For the remaining noun semantics, 10 identify additional aspects of the design (see Table 3.3-2) that one may also wish to model with only one semantic, *feeling* not representing any UML modelling feature. The non-candidate class semantics have been manually classified by the author through consideration of what the semantic description implies in terms of UML Modelling features.

Table 3.3-2 Additional UML Model Features

WordNet Semantic	WordNet Description	Modelling Implications
Act	Nouns denoting acts or actions	Operation
Possession	Nouns denoting possessions and transfer of possessions	Relationship
Quantity	Nouns denoting quantities and units of measure	Multiplicity
State	Nouns denoting stable states of affairs	Object State
Process	Nouns denoting natural processes	Algorithm
Motive	Nouns denoting goals	Algorithm
Relation	Nouns denoting relations between people, things or ideas	Relationship
Attribute	Nouns denoting attributes of people and objects	Class Attribute
Event	Nouns denoting natural events	Algorithm/Operation
Cognition	Nouns denoting cognitive processes and contents	Algorithm

The modelling implications identified from these semantics are currently not utilised within the ASA, with the exception of the *Attribute* semantic, as they require further investigation on how best to manage their inclusion within automated model generation. However, during manual analysis (of the individual words) it became apparent that in some cases the semantics listed in Table 3.3-2 could still indicate a class candidate within certain constraints. The constraint considers nouns from the semantic groups listed in Table 3.3-2 , which also have an artefact semantic within their set of candidate senses. An *Artefact* is something that is a manmade item (by WordNet's definition) such as a *Car* or a *House*. Therefore, the ASA considers that an *Artefact*, a manmade object, will always be a high value item in terms of UML modelling because an entity which is manmade and is contained within the specification indicates a feature which should be considered for inclusion within the UML model.

For this reason, a *noun* that has a semantic contained within the set of non-class semantics, but also contains the semantic *Artefact* within its set of senses (obtained from WordNet), can also be considered as a class candidate. This is formally defined as follows:

Rule 2 – Class Detection from Non-Class Semantics

If a noun's most common semantic belongs to the set of non-candidate class semantics, and that noun also contains an artefact semantic, then noun is a candidate class

The inclusion of any candidate UML feature from either non-class or candidate class semantic rules will always be scrutinised once the model has been presented to the analyst for review.

The usage of this data aids the discovery of candidate classes that may be overlooked but the automated approach

Table 3.3-3 demonstrates a count of all the nouns that do not have a candidate-class semantics and also counts those nouns that also contains an *Artefact* semantic as well. The usage of this data aids the discovery of candidate class that may be overlooked by the automated approach.

Table 3.3-3 Semantic Word Count per Non-Class Candidates

Non-Class Semantic	Total Noun Count	Non-Class Count (no artefact semantic)	Candidate Class Count (contains artefact semantic)
Act	6762	6628	134
Possession	1302	1249	53
Quantity	1745	1674	71
State	5066	4982	84
Process	927	908	19
Motive	63	63	0
Relation	557	540	17
Attribute	3483	3408	75
Event	1167	1111	56
Cognition	3470	3355	115

The results indicate that a non-class semantic but also has an *artefact* semantic yields a small set of likely candidate classes or none in one cases. The inclusion of this small set of candidates is considered helpful to the automated analysis process by assisting in extracting additional candidates from the natural language requirements specification.

Further justification for the inclusion of this rule can also be demonstrated by this example: *Individually tailored programs of study must not contradict the rules governing the degree, such as the structure or prerequisite* ***courses*** *required so that the student can qualify for the degree's compulsory* ***courses****.*

The term in bold, *courses,* has *act* as its most common semantic implying some action (see

Table 3.3-4). This would be ignored by the *Rule 1* and would not be created as a *class*. However, *courses* and its context within the example is something that we would wish to model as a class within the proposed software system. However, context is not considered within the ASA therefore, consideration of both non-class semantics and the presence of an *Artefact* semantic as defined by *Rule 2* yields the creation of a candidate class (see

Table 3.3-4). This highlights a strong case for disambiguation which is addressed in section 3.5.

Table 3.3-4 'Course' Sense Definition List

Sense	Semantic	Terms	Description(sense)
1	**noun.act**	**course1#1, course of study#2**	**education imparted in a series of lessons or meetings; "he took a course in basket weaving";**
2	noun.group	course#2, line2#10	a connected series of events or actions or developments; "the government took a firm course"
3	**noun.artifact**	**course#3, course of action#1**	**facility consisting of a circumscribed area of land or water laid out for a sport; "the course had only nine holes"**
4	noun.act	course#4, path#4, track#1	a mode of action; "if you persist in that course you will surely fail"
5	noun.object	course#5	a line or route along which something travels or moves; "the hurricane demolished houses in its path"
6	noun.location	course#6, trend1#2	general line of orientation; "the river takes a southern course"
7	noun.food	course#7	part of a meal served at one time
8	noun.artifact	course#8, row#4	(construction) a layer of masonry; "a course of bricks"

3.3.2.3 Determiners, Nouns & Multiplicity Mappings

A determiner is a modifying word that comes before the noun. The determiner references the noun that it precedes and is either definite (specific), indefinite (general) or is quantitative. The determiner and its corresponding noun aid the identification of a relationship's multiplicity (i.e. the cardinality or number of elements of some collection). This is obtained through consideration of the determiner's quantification, the noun's plurality, the verb expressing the relation and both sentence subject and objects. Figure 3.3-4 demonstrates the multiplicity mappings for both nouns and determiners and the cardinality that they map to.

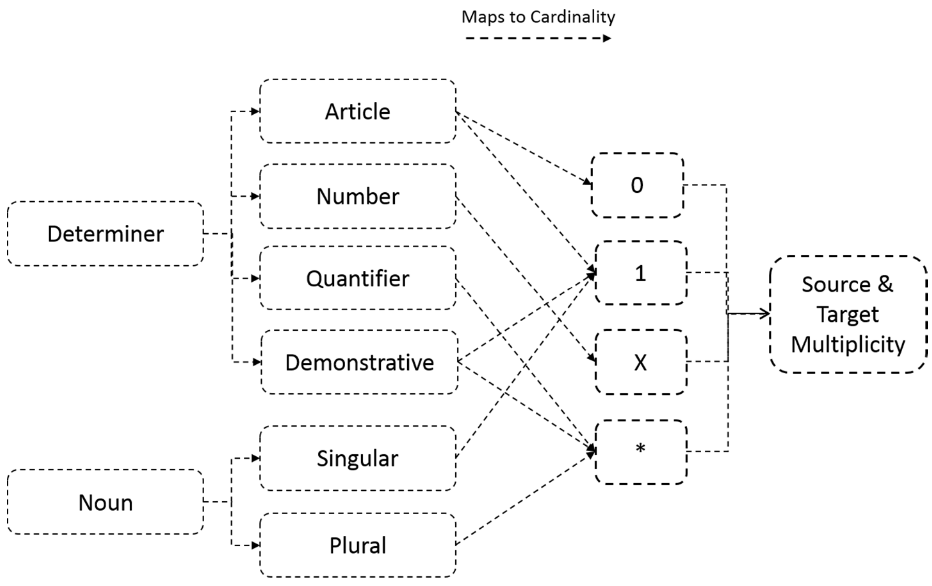


Figure 3.3-4 Determiner & Noun Multiplicity Mappings

Multiplicity - Start Range (Determiners): The determiner's quantifier demonstrates an amount and identifies the starting multiplicity range, the *X* of the *Y* in *[X..Y]*.

Determiners themselves have types: *Article* demonstrates whether it is a definite or indefinite with the most common being *a, an,* and *the,* and represents a single multiplicity; *Number* represents a cardinal number *(i.e. a minimum set of X); Quantifiers* identify a many relationship *(many, any, all, every, some* and *each);* finally, *Demonstratives,* most commonly *this, that, these* and *those,* represent either a single or many multiplicity mapping and when no determiner is present it indicates a zero mapping. Regardless of their type, the extraction of multiplicity mappings is key to identifying the relationships between classes. Table 3.3-5 summaries the determiners and their associated multiplicity mappings.

Table 3.3-5 Determiner Quantification

Determiner	Quantification
a, an, another, the, both, either, that, this	1
all, any, every, them, these, those, each, many, much, some	* (i.e. many)
No determiner present	0

Multiplicity - End Range (Noun Plurality): The head noun determines the end range of the current multiplicity under consideration, the Y of the [X..Y]. Table 3.3-6 summarises the multiplicity mappings for nouns.

Table 3.3-6 Noun Multiplicity Mapping

Noun Type	Multiplicity Reference
Non-plural	1
Plural	* (i.e. many)

Multiplicity Detection Process: The multiplicity range for a candidate class is constructed during sentence analysis and is obtained for both the source *(sentence Subjects)* and the targets *(sentence Objects)* respectively.

For example: *Each student is enrolled in many seminars.* Assumptions for this example includes that *Student and Seminar* are classes and that the verb *enrolled* indicates a relationship between *Student and Seminar*.

```
(ROOT
  (S
    (NP (DT Each) (NN student)) ← Subject (Target Multiplicity)
    (VP (VBZ is)
      (VP (VBN enrolled)
        (PP (IN in)
          (NP (DT many) (NN seminars)))))) ← Object (Source Multiplicity)
```

Figure 3.3-5 Source/Target Multiplicity Detection

The process is primarily a lookup which involves identifying the determiner (DT) contained within the noun phrase (NP) and the plurality of the noun (NN). This information is subsequently used to determine what multiplicities for a given noun or determiner through the conversion rules defined in Table 3.3-5 and Table 3.3-6, where Table 3.3-7 summarises the outcome for the example in Figure 3.3-5

Table 3.3-7 Multiplicity Mappings for Example

Multiplicity Type	Determiner/Mapping	Noun/Mapping
Source	Each / [*]	Student / [1]
Target	many / [*]	Seminars / [*]

With all multiplicities mapped, it is a simply case of applying conversion rules (see Table 3.3-8)

Table 3.3-8 Multiplicity Mapping Conversion Rules

Determiner Value	Noun Form	Range Mapping
0	Singular	[0..1]
0	Plural	[0..*]
1	Singular	[1..1]
1	Plural	[1..*]
*	Singular	[1..*]
*	Plural	[*..*]
Missing	Singular	[0..1]
Missing	Plural	[0..*]

The resulting multiplicity mapping for the example is demonstrated in Figure 3.3-6

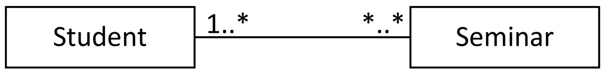


Figure 3.3-6 Final Mapping for Source/Target Multiplicity Detection

Multiplicity Preservation: The multiplicity range is then preserved in memory for later usage and used in conjunction with any potential relationship that maybe uncovered during the analysis process. Since multiplicities may change from relationship to relationship they are stored with the specific relationship discovered during its analysis rather than the class.

The rule used to detect multiplicities can be defined as follows:

Rule 3 - Start Range Multiplicity Detection (Determiners Present)

*If a determiner belongs to the set of multiplicity mappings {0, 1, *}, then the start range for multiplicity has been found*

Rule 4 – Start Range Multiplicity Detection (Missing Determiners)

If the determiner does not exist, then the start range is known as single (1)

Rule 5 - End Range Multiplicity Detection Rule (Plural Nouns)

If a noun is a candidate class, as defined by Rule 1 or Rule 2, and the noun is plural, its mapping is known as many (*)

Rule 6 – End Range Multiplicity Detection (Non-Plural Nouns)

If the noun is a candidate class, as defined by Rule 1 or Rule 2, and the noun is not plural, then its mapping is known as single (1)

3.3.2.4 Attributes, Class Hierarchies and State from Noun Pre-Modification

Noun pre-modification, where one or more words *(adjectives, nouns* or *participles)* are placed before the Noun Head and further define the noun's meaning, can express much more in terms of UML Modelling. Figure 3.3-7 demonstrates the mappings that pre-modification can represent.

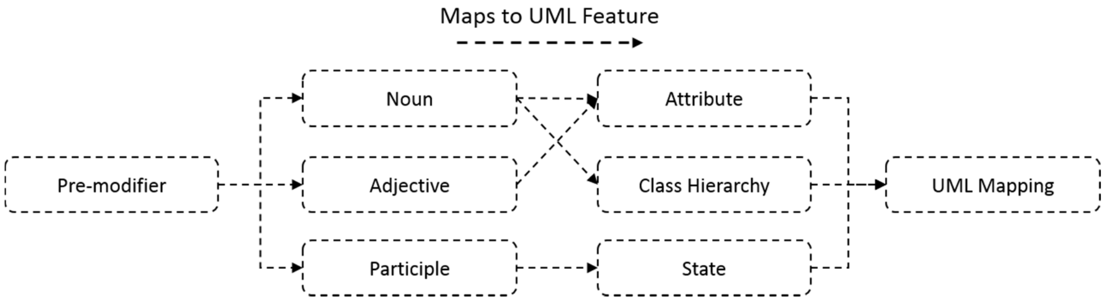


Figure 3.3-7 Pre-Modifier to UML Mapping

The presence of a pre-modifier requires a decision of what it implies and if it should be included within the model. The features of pre-modification are discussed in the following sub-sections.

Adjective Pre-Modifier: The most common pre-modifier is the adjective, which precedes the word that it modifies and typically expresses an attribute of that word. They can be grouped into categories for example: *colour, size, sound, taste, touch, shape* but other groupings do exists. However, unlike the rich semantic groupings for both noun and verbs, adjectives are only represented by the types <*adj.all*> (all adjective clusters) and <*adj.pert*> (relational adjectives) within WordNet and since there are no useful semantic groupings for adjectives a manual set of semantics would have to be defined and classified accordingly. This would allow similar decisions to be made in line with how the ASA manages decisions for both nouns and verbs. The identification and consideration of adjective semantic groupings is a path that is not currently followed by the ASA.

Noun Pre-Modifier: Based primarily on its semantics, it is possible that the *noun-modifier* can represent an attribute (see Section 3.4 Additional Modelling Considerations), or a class inheritance hierarchical structure.

For example, *"The game will display the **defence grid** and **offence grid** to each player"*

```
(ROOT
  (S
    (NP (DT The) (NN game)) ← Subject
    (VP (MD will)
      (VP (VB display)
        (NP
          (NP (DT the) (NN defence) (NN grid)) ← Object
          (CC and)
          (NP (NN offence) (NN grid))) ← Object
        (PP (TO to)
          (NP (DT each) (NN player)))))
    (. .)))
```

Figure 3.3-8 Example Syntactic Structure

Figure 3.3-8 demonstrates the syntactic structure for this example, where only the sentence objects are only considered. During syntactic analysis any determiners are ignored, the head noun is identified first and then modifiers are classified, defined by these rules:

- The Head Noun is the last noun contained within the noun phrase
- Modifiers are any nouns which precede the Head Noun

The phrases can be broken down to its constituent parts and their individual semantics can be extracted, see Figure 3.3-9 and Table 3.3-9.

$$\texttt{(NP (DT the) }\overbrace{\texttt{(NN defence)}}^{\text{Modifier}}\ \overbrace{\texttt{(NN grid)}}^{\text{Head}}\texttt{)}$$

Figure 3.3-9 Noun Modifier/Head Parts

The pre-modifiers *defence* or *offence* do not indicate an attribute because their semantics are not within the set of candidate attributes. Furthermore, their semantics are outwith the set of candidate classes Table 3.3-9 demonstrates the semantics captured from WordNet but the semantics for the head noun are contained within this set.

Table 3.3-9 Noun Phrase Semantic Analysis

Phrase	Pre Modifier Noun	Pre Modifier Semantics	Head Noun	Semantics
Defence grid	defence	<noun.process>	grid	<noun.artefact>
Offence grid	offence	<noun.act>	grid	<noun.artefact>

Therefore, their inclusion within the noun phrase indicates some form of sub-class to super-class mapping (i.e. *type/kind-of).* Even if the modifier semantics were within the set of candidate classes, it is still possible that they represent a *type/kind-of* hierarchical structure, rather than two independent classes.

The hierarchical structure rule is defined as follows:

Rule 7 – Class Hierarchy Detection Rule

Given the noun phrase, if the head noun is a candidate class as defined by Rule 1 or Rule 2 and the head noun's pre-modifier is also candidate class as defined by Rule 1 or Rule 2, then an interface and abstraction is extracted based on the head noun

In both cases the head noun, *grid,* is within the set candidate classes and neither modifier is within the set of attribute semantics which results in the creation of a class hierarchy (Figure 3.3-10).

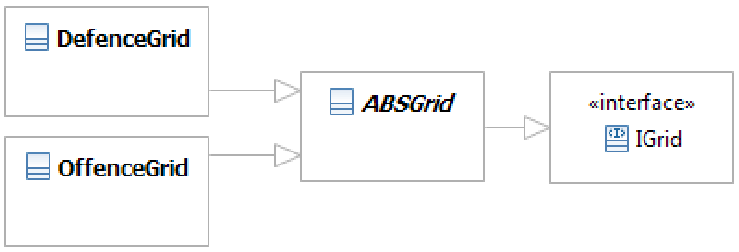


Figure 3.3-10 Automated Example Noun Phrase to UML Mapping

The creation of this inheritance hierarchy is a point of flexibility and a point of consideration for the developers within the overall architecture. It allows any concrete implementation of the abstraction (IGrid) to be utilised at run-time, and allows consideration of the system to accommodate new future *grid* types.

State and Participle Pre-Modifier: The participle indicates a characteristic feature of the noun which it modifies and is a verb that comes in two forms:

- Passive Form *(-ed)*
    - In its passive form, the participle represents a sense of completion
- Active Form *(-ing)*
    - In its active form, the participle represents a sense of incompletion

Consider the examples:

Passive Form: *'A sold car has a 3 year warranty'*

$$\texttt{(NP (DT A) }\overbrace{\texttt{(VBN sold)}}^{\text{Participle}}\ \overbrace{\texttt{(NN car)}}^{\text{Head}}\texttt{)}$$

Figure 3.3-11 Passive Participle/Head Noun Parts

Active Form: *'The approaching train is coming from London.'*

$$\text{(NP (DT The) }\overbrace{\text{(VBG approaching)}}^{\text{Participle}}\ \overbrace{\text{(NN train)}}^{\text{Head}}\text{)}$$

Figure 3.3-12 Active Participle/Head Noun Parts

During syntactic analysis, the participle is identified by its syntactic position within the noun phrase (see Figure 3.3-11 and Figure 3.3-12) and is discovered by:

Rule 8 – Object State Identifier

Given the noun phrase, if the head noun is a candidate class as defined by Rule 1 or Rule 2 and the head noun's pre-modifier is a participle, then an object state accessor is said to exist.

In the case of modelling, a participle pre-modifier implies a state attribute of Boolean type. This would manifest itself within the above examples as a public accessor to determine the state of the candidate class via the name of the participle modifier, if a candidate class exists as defined by the head noun. The key reason why it defines a Boolean is that the participle modifier indicates either a sense of completion (true) or incompletion (false). It is therefore possible with this knowledge to create said public accessor where the passive participle modifier would be defined as *has<ModifierName>* and the active modifier would be defined as *is<ModifierName>.*

3.3.2.5 Relationships, Parameters & the Post-Modifier

Post-modifiers, as with pre-modifiers, give additional information in relation to the head noun. They introduce information that can express *relationships, parameters, actions* or *state* in terms of modelling (see Figure 3.3-13). Although, for consistency, the discussion of both *clausal constructs* is pushed towards the overall view at the clausal level (see 3.4 Additional Modelling Considerations).

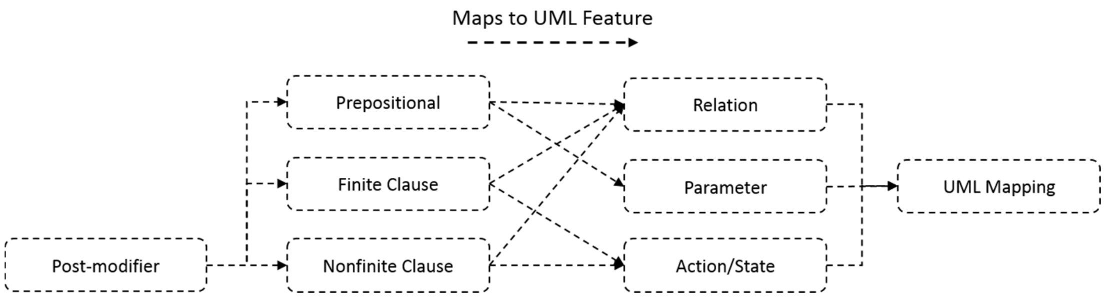


Figure 3.3-13 Post-Modifier UML Mapping

The most common form of modification for both verbs and nouns is the preposition, which expresses a relationship between either of these entities.

Figure 3.3-14 details the structure of a prepositional phrase, its attachments, to what it can be attached (either to noun or verb phrases), and what these represent when mapped to UML.

The Preposition's complement can take the form of an *adverb, noun phrase* or *clause*. Dependent on the complement's type, the processing would revert to the relevant processing technique. The most common complement is a *noun phrase* and the processing of this feature would be the same as that discussed in section 3.3.2.

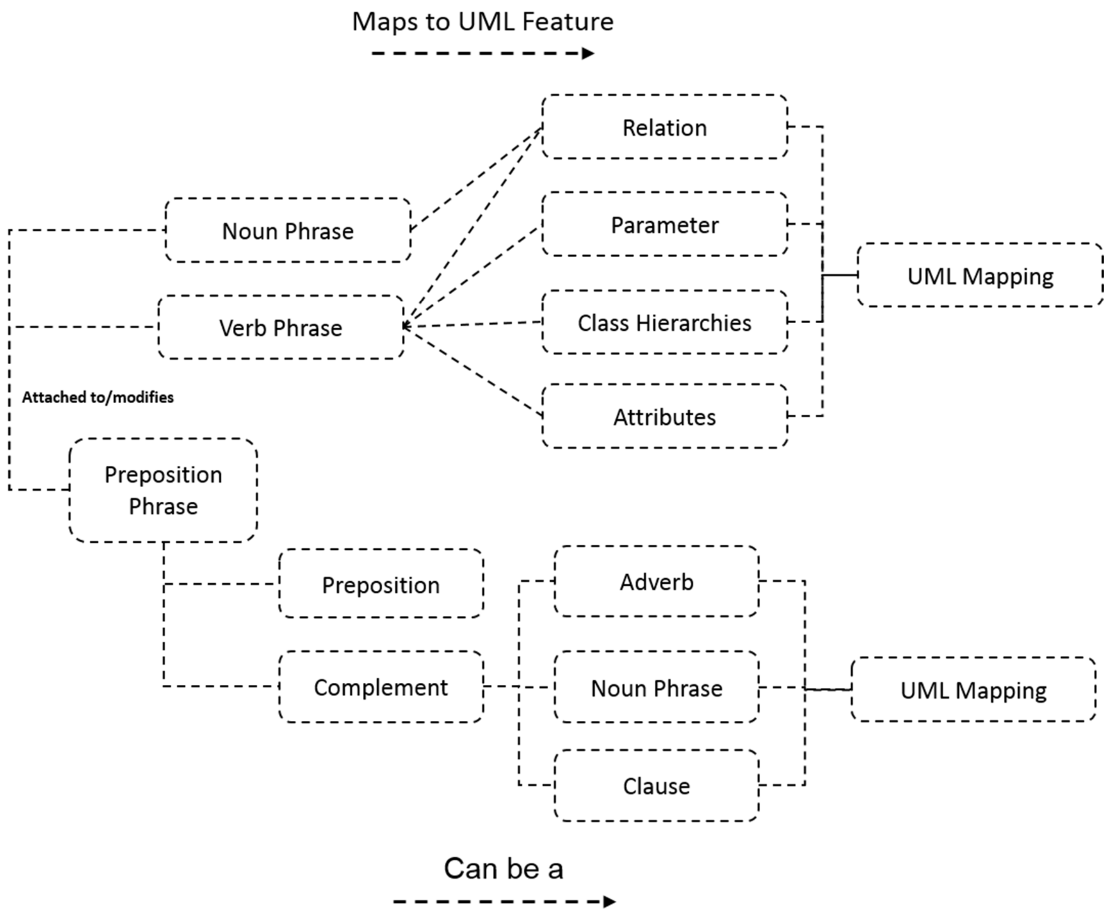


Figure 3.3-14 Preposition Phrase Syntactic Structure

Defining Prepositions Semantics: The preposition can express *spatial, temporal* or *logical relationships*. They typically take the form of words such as *at, in* and *on*. However, the semantics for propositions are not available through WordNet therefore, a manual custom generalisation of the prepositions' meaning has been established using online dictionaries **[Dic07, Wiki07, Oxf07, COL07]**, which can then be used in automated processing.

The *first sense definition* across each of the dictionary sources identifies the most common sense of that word (see Appendix A.3). Identification of prepositional semantics uses the same common semantic technique as defined for both candidate class and relationship detection (see Section 3.2.1). This is then used to define its grouping (*spatial, temporal* or *logical* relationship*)* including some sub-categories. In some cases, not all of the *first sense definitions* within these dictionaries may be the same. In that case, a *majority-rules approach* is undertaken and the *Common Semantic Model (CSM)* is then considered for the most appropriate classification for that preposition (i.e. the sense with the highest average).

For example, the preposition *as* has 2 definitions

- The first definition indicates a role,
- The second definition introduces a basis of comparison.

Table 3.3-10 details the definitions obtained from each dictionary, and in the majority of cases it defines *as* as indicating *role*.

Table 3.3-10 'as' Dictionary Definitions

Dictionary	Sense Role
Dic07	1. In the role, function, or status of: to act as leader.
Wiki07	1. Introducing a basis of comparison, with an object in the objective case. 2. In the role of.
Oxf07	1. used to refer to the function or character that someone or something has: 2. during the time of being (the thing specified):
COL07	1. in the role of, being, my task, as his physician, is to do the best that I can, 2. as for or to with reference to, 3. as if or though as it would be if, she felt as if she had been run over by a bulldozer, 4. as (it) is in the existing state of affairs,

Preposition Attachment Analysis: The main aspect of the syntactic analysis is to determine to what the preposition is attached: either noun or verb phrases. Therefore, recognising what the preposition is attached to allows the discovery of relationships that are not identifiable through the verb of the sentence. In addition, parameters, attributes and class hierarchical structures also become available for detection, since one of the key complements of the preposition is a noun phrase. All of these are discussed throughout the remainder of this section.

The syntactic rule for detecting the preposition's attachment can be defined as follows:

Rule 9 – Noun Preposition Attachment Detection

If the prepositional phrase's parent is a Noun Phrase, then the preposition is said to be attached to that the noun phrase.

Rule 10 – Verb Preposition Attachment Detection

If the prepositional phrase's parent is a Verb Phrase, then the preposition is said to be attached to that verb phrase.

Both noun and verb attachments are treated differently. The preposition attached to a noun indicates a relational aspect to what it is attached. Treatment of the verb is very different and requires further consideration, since the preposition can express relationships, parameters, attributes and class hierarchical constructs. Therefore, in the context of verb attachment, a decision matrix is utilised, which is a combination of the verb and preposition semantics, and the intersection of both semantics determines the relevant action to take be taken.

Noun Phrase Complement: Identifying relationships between the subject of the sentence and the complement of the preposition is a technique not considered by other related works. Relationships are typically discovered through the presence of a verb, which is preceded by a noun. This technique overlooks candidate relationships between the subject and the preposition complement as no verb exists between them except for the semantic connection. In addition, this neglect is primarily due to sentence simplifications, which eliminates the preposition sentence feature.

A preposition which is attached directly to a noun phrase can indicate an association relationship without the presence of a verb, which is the key reason to consider potential relationships from this syntactic structure. If the preposition attachment were not considered, then the relationship between both the preposition's noun complement and the head noun, contained within the noun phrase that the preposition is attached to would be lost resulting in the generation of an incomplete model.

A relationship is established between both noun phrase and preposition complement through recognition of the NP→PP attachment pattern. This is discovered through a reverse search of the syntactic structure to obtain the preposition attachment pattern.

For Example: *'A robot with an arm will collect items from the assembly line.'*

```
(ROOT
  (S
    (NP ← Identifies preposition is attached to this noun phrase
      (NP (DT A) (NN robot))
      (PP (IN with)← Preposition
        (NP (DT an) (NN arm))))← Complement
    (VP (MD will)
      (VP (VB collect)← both preposition & noun attached to this verb
        (NP (NNS items))
        (PP (IN from)← Preposition
          (NP (DT the) (NN assembly) (NN line)))))← Complement
    (. .)))
```

Figure 3.3-15 Noun Phrase Attachment Analysis

Figure 3.3-15 details the syntactic structure for this statement. The first noun phrase of the sentence *'A robot with an arm'* also contains a preposition with a noun phrase complement. There is no verb to indicate that a relationship exists. However a relationship does, and is identified through the attachment analysis pattern recognition NP→PP and the definition of the preposition itself. Note that in the second half of the statement, *'will collect items from the assembly line'*, both the noun and preposition phrases are attached to the verb of the sentence. Figure 3.3-16 demonstrates the UML model generated from this example.

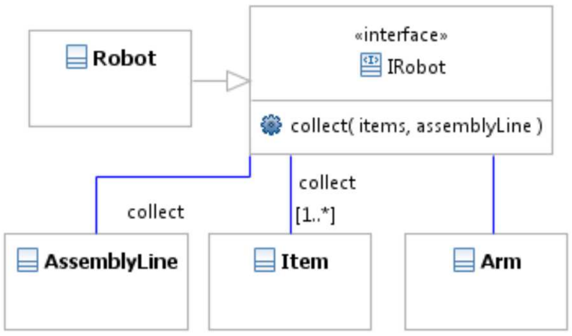


Figure 3.3-16 Resulting UML Model from Preposition Analysis

The rules applied to create this model in no particular order are as follows:

- Rule 1 – Class Detection or Rule 2 – Class Detection from Non-Class Semantics
- Multiplicity Detection: ((Rule 3 *or* Rule 4) *and* (Rule 5 *or* Rule 6))
- Rule 9 – Noun Preposition Attachment Detection
- Rule 10 – Verb Preposition Attachment Detection
- Rule 11 – Operation Detection *or* Rule 12 - Relationship Detection
- Rule 13 – Subject Operation Placement *or* Rule 14 – Object Operation Placement
- Rule 15 – Active Voice Parameter Creation *or* Rule 16 – Passive Voice Parameter Creation

3.3.3 Verb Phrase

The verb phrase is the main source for detecting candidate relationships and operations within the UML model, but it is also possible to detect a variety of other UML-related features.

The approach towards operation and relationship detection is a by-product of the class detection process. Verbs contained within the sentence help the analyst make decisions regarding both operations and relationships. In addition, the analyst's own knowledge of the situation may also assist the detection process.

Figure 3.3-17 defines the structural components of a verb phrase and its UML mapping, describing what is possible to identify such as:

- Relationships
- Parameters
- Attributes
- Operations and Boolean Operations
- Class Hierarchical Structures

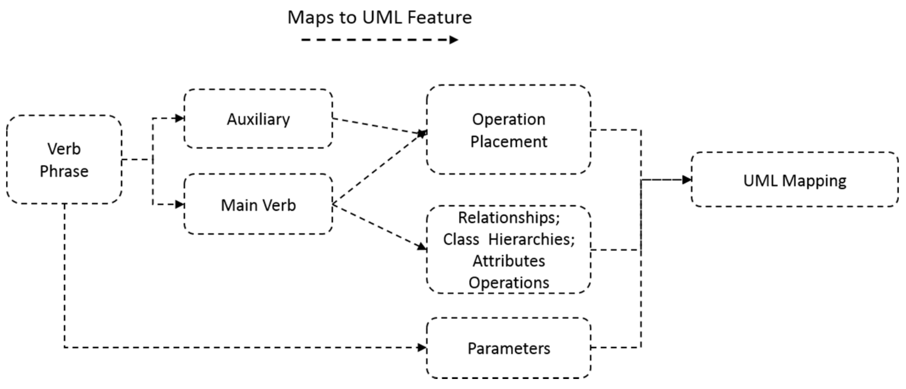


Figure 3.3-17 Verb Phrase to UML Mapping

Syntactic Considerations

The verb itself comes in two key parts, the auxiliary, which holds information about *mood, modality (modals), aspect* and *voice* and the main verb. The main verb defines the main action of the subject contained within the sentence. The auxiliary comes before the main verb, but does not necessarily need to be included with the main verb; *voice* is the only auxiliary considered by the ASA. *Voice* assists in placing operations and extracting parameters but this function is dependent on its form (either passive or active) and is considered later in this section. The *modal* verb is also of interest as it can define the possibility/necessity of an action. *Aspect* is also of interest to the ASA and has two forms: progressive and perfect. The *progressive* expresses an incomplete action in progress at a specific time whereas *perfect*, typically retrospective, indicates a completed action. Both aspect and the modal require additional consideration of what it can be used for within modelling and how it can be incorporated within the ASA. Finally, m*ood* is not considered as it demonstrates the manner in which a thought is expressed.

Detection of the verb is discovered via OpenNLP, which identifies the verb phrase as *VP*, and all following verbs are tagged starting with *VB* followed by a *[D, G, N, P* or *Z]* that identifies their type. Modal verbs are tagged as *MD* and are auxiliary verbs that give more information about the main verb's function and define the likelihood (*shall*) through to essential (*should*) (see Table 3.3-11, for a full description of verb tags definitions).

Table 3.3-11 OpenNLP Tags for Verbs

Tag	Description
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
VP	Verb Phrase
MD	Modal

In addition, all verbs, once their type has been identified *(i.e. past* and *present tense, singular),* are reduced to the base form so that the creation of duplicate operations or relations is avoided. For example, the verbs *transports* and *transported* will be reduced to their base verb *transport*.

Semantic Considerations

WordNet offers a set of fifteen verb semantics that can imply a relationship and/or an operation between the classes detected in each sentence. Table 3.3-12 details and describes the semantics obtained from the WordNet dictionary and their modelling implications.

Table 3.3-12 Candidate Relationship/Operation Semantics

WordNet Semantic	WordNet Description	Relationship/Operation Modelling Implications	
		Relationship	Operation
Body	Verbs of grooming, dressing and bodily care		X
Change	Verbs of size, temperature change, intensifying, etc		X
Cognition	Verbs of thinking, judging, analysis, doubting, etc	X	X
Communication	Verbs of telling, asking, ordering, singing	X	X
Competition	Verbs of fighting, athletic activities	X	X
Consumption	Verbs of eating and drinking		X
Contact	Verbs of touching, hitting, tying, digging	X	X
Creation	Verbs of sewing, baking, painting, performing		X
Emotion	Verbs of feeling		X
Motion	Verbs of walking, flying, swimming	X	X
Perception	Verbs of seeing, hearing, feeling		X
Possession	Verbs of buying, selling, owning	X	X
Social	Verbs of political and social activities and events	X	X
Stative	Verbs of being, having, spatial relations		X
Weather	Verbs of snowing, raining, thawing, thundering		X

The key rationale for choosing these semantic classifications of a verb is based upon the common semantic model (see Section 3.2.1). For example, the verb *transport* has five different senses and Table 3.3-13 details this common sense understanding.

Table 3.3-13 *Transport* Verb Sense Classifications

Sense	Semantic	Term	Description
1	verb.motion	transport#1	move something or somebody around; usually over long distances
2	verb.contact	transport#2, carry2#1	move while supporting, either in a vehicle or in one's hands or on one's body; "You must carry your camping gear"; "carry the suitcases to the car"
3	verb.emotion	enchant#1, enrapture#1, transport#3, enthrall#1, ravish#2, enthral#1, delight2#3	hold spellbound
4	verb.motion	transport1#4, send#4, ship#1	transport commercially
5	verb.contact	transmit#4, transfer#7, transport1#5, channel#3,	send from one person or place to another; "transmit a message"

It is the only most commonly understood semantic (in the example*, sense #1*) that is used to make a decision regarding the inclusion of a relationship and/or operation. The semantic *motion,* for example, implies movement from one location to another; that there are typically two entities involved in the situation, *i.e.* the *mover* and *something else* evaluated on a sentence by sentence basis. Therefore both a relationship and an operation will be created for this verb. Under no circumstances is any form of disambiguation or surrounding context taken into consideration during the lookup process.

Relationship/Operation Detection Rules: As with class detection, the technique of relationship/operation detection is based upon the verb's semantic classification and its syntactic structure of connected sentence subjects and/or objects to the verb itself. These rules can be defined as follows:

Rule 11 – Operation Detection

If a verb's most common semantic belongs to the set of candidate operation semantics and the verb's semantic does not belong to the set of candidate relationship semantics, then that verb is a candidate operation

Rule 12 - Relationship Detection

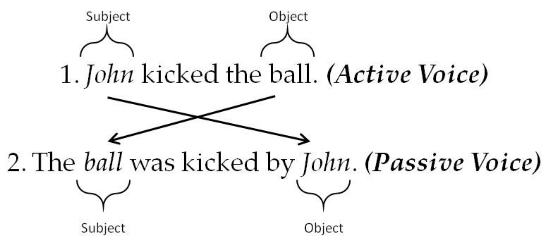
If a verb's most common semantic belongs to the set of candidate relationship semantics then that verb is a candidate relationship

The additional features of the verb such as, *Parameters, Operation Class Location, Class Hierarchies* and *Attributes* are discussed in the following sub-sections.

3.3.3.1 Operation Class Location

The operation will be placed with the subject of the sentence. The subject of sentence identifies the *actor*, the one performing the verb's action, whereas the object of the sentence is the one who receives this action. This clearly has implications in terms of operation class location. The placement of an operation within a class is decided upon through recognition of sentence voice *(Passive* or *Active).*

For example:


Subject
Object
1. John kicked the ball. (Active Voice)
2. The ball was kicked by John. (Passive Voice)
Subject
Object

In syntactic terms, the key rule that identifies passive voice sentences is

1. A verb in the form of *be* followed by an *-ed* participle

Given this structure, it is possible to differentiate between passive and active voice forms.

The first example is in active voice and allows correct identification of the subject *John*, who is the actor of the verb's action and where the operation would be placed. *Ball* is the receiver of the verb's action. However, in example two *ball* is the subject, but it is in the passive voice, meaning it receives the action of the verb. In this case, *John* is still the actor (demoted-subject) of the action, where the operation would be placed.

Upon reflection and in the context of UML modelling, the placement of the operation with the actor (sentence subject, depending on voice) may not be best suited as the receiver may have their state changed as a result of the action. Therefore, it may be more appropriate to place the operation with the receiver instead, as it could be considered that the actor is calling the action associated with the receiver. Nonetheless, the current approach is to place the operation within the actor depending on sentence voice as defined by these rules:

Rule 13 – Subject Operation Placement

If the verb is in an active form and as defined by Rule 9 is an operation and by Rule 1 or Rule 2 the sentence subject is a class candidate, then the operation will be placed with the subject of the sentence

Rule 14 – Object Operation Placement

If the verb is in its passive form and as defined by Rule 9 is an operation and by Rule 1 or Rule 2 the sentence object is a candidate class, then the operation will be placed with the object of the sentence

3.3.3.2 Parameter Detection from Operational Verbs and Sentence Voice

Parameters are identified from sentence objects or the prepositional complement, if they are nouns, through a syntactic attachment analysis. The reason why sentence objects are likely candidate parameters is founded in the function of the sentence, where objects are acted upon via the verb's describing action.

For example, *'The game will display a grid."*

```
(ROOT
 (S
   (NP (DT The) (NN game))
   (VP (MD will)
     (VP (VB display) ← Operation
       (NP (DT a) (NN defence) (NN grid)) ← Candidate Parameter
   (. .)))
```

Figure 3.3-18 Candidate Parameters Parse Tree (active voice)

Figure 3.3-18 details the syntactic structure of the example sentence. From a bottom-up search of the parse tree it is discovered that the object *(gird)* is attached to the verb *(display)*, which will then be created as a parameter of the function *display* for the class *game.* Thus, the operation signature will then be defined as follows:

Game.display:(Grid grid)

This can be defined by the following rule:

Rule 15 – Active Voice Parameter Creation

If the sentence is in active voice and by Rule 9 an operation exists and by Rule 1 or Rule 2 a class candidate exists for both sentence subjects and objects and by Rule 12 the operation is placed with the subject of the sentence, then the object of sentence is considered as a parameter of that operation

In a similar vein, as illustrated in Figure 3.3-19, when it is discovered that the same construct is in passive voice, the subject of the sentence is then used as the parameter of the operation, which is then placed with the demoted subject.

```
(ROOT
 (S
   (NP (DT A) (NN defence) (NN grid)) ← Candidate Parameter
   (VP (VBD was)
     (VP (VBN displayed) ← Operation
       (PP (IN by)
         (NP (DT the) (NN game)))))
   (. .)))
```

Figure 3.3-19 Candidate Parameters Parse Tree (passive voice)

Passive voice parameter creation is defined by the following rule:

Rule 16 – Passive Voice Parameter Creation

If the sentence is in passive voice and by Rule 9 an operation exists and by Rule 1 or Rule 2 a class candidate exists for both sentence subjects and objects and by Rule 13 the operation is placed with the object of the sentence, then the subject of sentence is considered as a parameter of that operation

3.3.3.3 Attributes from the Verb form *have*

Analysis of the verb can allow the identification of class attributes which is only apparent through the presence of a particular verb type and not achievable through additional attribute detection methods based on noun semantics (see Section 3.3.2.1). The verb in the form of *have* and its associate forms such as *has* and *had* permits identification of attributes since the semantics of the verb form *have* indicate possession/ownership. It is only sentence objects which are considered as attributes. The subject of the sentence is considered as the class that receives these as its attributes.

The basis of the rule that allows identification of attributes from the verb is defined as follows:

Rule 17 – Verb Derived Attribute Detection

If the verb of the sentence belongs to the set of verb forms {has, had, have} and the noun following the verb is a class candidate as defined by Rule 1 or Rule 2, then that class is transformed into an attribute

For example, *'The room* ***has*** *a balcony and a bathroom"*

In Figure 3.3-20 *has* is the main verb of the sentence and has no preceding or following verbs. As a result, both *balcony and bathroom* are therefore considered as attributes, which will be attached to the subject of the sentence *room.* Note that all of these words are also candidate classes based on their semantics.

```
(ROOT
  (S
    (NP (DT The) (NN room)) ← Candidate Class
    (VP (VBZ has) ← Verb in form of ‘have’
      (NP
        (NP (DT a) (NN balcony)) ← Candidate Attribute & Candidate Class
        (CC and)
        (NP (DT a) (NN bathroom)))) ← Candidate Attribute & Candidate Class
    (. .)))
```

Figure 3.3-20 Attributes based on *have* Verb forms

In some cases, it may not be appropriate to model these as an attributes when they are better suited to being an actual class within the model.

Given the example: *A book* **has** *a title and author*

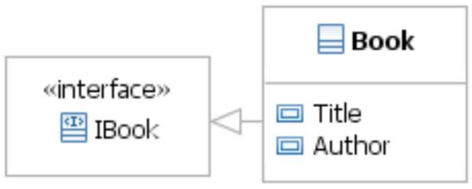


Figure 3.3-21 Automated Model Analysis - Attribute from *has* rule

The current rule for attribute detection would result in both *author and title* being created as attributes within the class *book* (see Figure 3.3-21). This is entirely justifiable, it is appropriate to say that a *book* has both an *author and title*, although it is also possible for *author* to be represented as a class which may be more appropriate (see Figure 3.3-22).

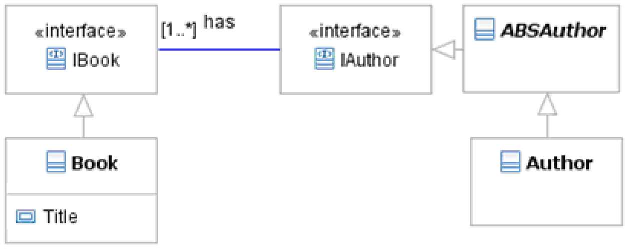


Figure 3.3-22 Alternative UML Model

3.3.3.4 Class to Attributes Transformation by means of Term Frequency Analysis

The decision to transform a class into an attribute is achieved through utilisation of both semantics and identification of the importance of a word, defined by its term frequency. Term frequency is used to determine how important a specific word is given the average frequency of terms within the specification, where an above average frequency indicates that it should remain a class and below average frequency indicates an attribute. In addition, an important factor is that the sentence must only contain one main verb which is not modified, as if it contains a verb that is modified it could potentially alter its meaning.

The approach can dynamically switch to create either an attribute or a class depending on the sentence construction and the importance of the relevant sentence features and is defined by the following rules:

Rule 18 – Dynamic Verb derived Attribute Detection

If the sentence contains a noun that is class candidate and is preceded by a verb, where the frequency count of that noun is less than the average noun frequency for the document and there exists only one verb within the sentence and the semantics for that verb belongs to the set of {has, had, have} forms, then the noun is said to be an attribute

Rule 19 – Class & Relationship Detection

If both the subject and objects of the sentence are class candidates and their semantics are not within the set of attribute semantics and their term frequencies are greater than the term frequencies for the document average and the verb belongs to the set of {has, had, have} forms and there is only one verb, then it is said that both subjects and objects are class candidates and an association relationship should exists between the class candidates

3.3.3.5 Class Type/Inheritance Hierarchies from Verb

This relies on the understanding and utilisation of specific forms of *be* verbs, which can indicate class type/inheritance hierarchies. The key to identifying a class type/inheritance hierarchy can be defined as follows:

Rule 20 – Inheritance Hierarchy Detection

If the verb belongs to the set of 'be' forms and that verb is the only verb in the sentence and both the subject and object of the sentence are candidate classes as defined by Rule 1 or Rule 2, then an inheritance hierarchy is said to exist between both class candidates

Consider the example, *A car is a vehicle.*

In Figure 3.3-23 *vehicle* would be the super-type for the class hierarchy. Not only is it possible to extract class hierarchies from noun pre-modifiers (see Section 3.3.2.4), it is also possible through verb consideration as well.

```
(ROOT
  (S
    (NP (DT A) (NN car))
    (VP (VBZ is)
      (NP (DT a) (NN vehicle))) ← Candidate Super-Type
    (. .)))
```

Figure 3.3-23 Class Hierarchies based on *'be'* Verb forms.

It is the presence of the verb in the form of *be*, and its syntactic structure *(subject-verb-object)* SVO ordering, which defines a modelling *is_a* relationship.

3.3.3.6 Preposition Verb Phrase Attachment

Previously discussed in the context of the noun (see Section 3.3.2.5), the preposition's complement can take the form of an *adverb, noun phrase* or *clause*. However, in the context of the verb phrase, it allows for the discovery of relationships, operations and class hierarchical structures.

This is achieved by two key methods

a. Identification of syntactic attachment via the pattern PP→VP

b. A Verb - Preposition Semantic Decision Matrix defines the action defined by the semantics of both the preposition and the verb attachment (see Appendix A.4)

The syntactic attachment is discovered through identification of phrase markers such as PP (preposition phrase). Once the phrase marker is found (PP), a bottom up search of the parse tree is performed to discover its parental attachment, in this case a VP (verb phrase), which may also be a NP (noun phrase). The item of interest is the PP → VP attachment, which is used to look up the matrix that contains the semantics of both verbs and prepositions. Therefore, to discover the most relevant model feature to be created, the semantics of both the verb and preposition have to be discovered. These are available via WordNet for verbs, utilising the most common semantic strategy, and a custom semantic model for the preposition semantics (see Section 3.3.2.5). Therefore, through an intersection of these semantics within the matrix, it is possible to decide whether to create a *relationship, parameter, attribute* or a *class hierarchical structure*. Once this information has been identified, the relevant feature can then be modelled defined by the following rule:

Rule 21 – Matrix Relationship Detection

If the noun is a class candidate as defined by Rule 1 or Rule 2 and the intersection of both the verb's semantic and the semantic of the preposition belongs to the set of relationship semantics, then a relationship is said to exist between the noun and the object of the preposition

Rule 22 - Matrix Parameter Detection

If the noun is a class candidate as defined by Rule 1 or Rule 2 and the verb is an operation as defined by Rule 11 and the intersection of both the verb's semantic and the semantic of the preposition also belong to the set of parameter semantics, then the object of the preposition is said to be the parameter of the operation

Rule 23 - Matrix Class Hierarchy Detection

If the noun is a class candidate as defined by Rule 1 or Rule 2 and the object of preposition is also a class candidate as defined by Rule 1 or Rule 2 and the intersection of both the verb's semantic and the semantic of the preposition also belong to the set of class hierarchical semantics, then it is said there exists a class hierarchical relationship between the class candidate and the object of the preposition

Rule 24 - Matrix Attribute Detection

If the noun is a class candidate as defined by Rule 1 or Rule 2 and the intersection of both the verb's semantic and the semantic of the preposition also belong to the set of attribute semantics, then the object of the preposition is said to be an attribute of the class candidate

The key rationale for utilising the decision matrix is founded upon phrasal verbs, which is a combination of both prepositions and the main verb. This combination typically changes the understanding. For example, *look after,* means *to keep under careful scrutiny,* but in separate contexts *look* and *after* produce a different understanding.

To exemplify further, *The customer is known as a member*

```
(ROOT
  (S
    (NP (DT The) (NN customer))
    (VP (VBZ is)
      (VP (VBN known)← attachment considered here
        (PP (IN as)
          (NP (DT a) (NN member)))))
    (. .)))
```

Figure 3.3-24 Verb-Preposition Example

In Figure 3.3-24 it is important to identify the correct verb and preposition attachment. As previously discussed, it is achieved by first finding the preposition (PP) and then a bottom up search to its parental attachment, the second verb phrase (VP) in this case. From the attachment pattern PP → VP, we discover the verb *'known'* and the preposition *'as'*. The semantics of the verb and preposition are as follows *('cognition', 'logical.role')*. Through matrix analysis, the intersection of both verb and preposition semantics identifies the decision contained within the matrix i.e. *'type-of'*, indicating that an inheritance hierarchy should be created. The justification for this specific example is that if something has a role, then it is possible to consider that it is a *type-of* implying a class inheritance structure, but more importantly the ASA aims to introduce points of flexibility/maintainability through the introduction of these hierarchical constructs. On the other hand 'member' could be considered to be synonymous with 'customer' as it is identifying a form that a customer can represent. However, there is no synonymous link between both 'customer' and 'member' that can be discovered. As a result the creation of the class hierarchy serves two purposes: it allows identification of a customer through an interface/abstraction hierarchy, but more importantly it also introduces a key point of flexibility and reusability.

In the case of Figure 3.3-24, *customer* would naturally have its own class hierarchical structure (see 3.4 Additional Modelling Considerations), based on its prior semantic analysis and would result in Figure 3.3-25. The ASA recognises the existence of this originally identified inheritance structure and subsequently modifies it and introduces a new *member* inheritance structure, which is deemed more appropriate from the syntactic and semantic analysis (see Figure 3.3-26).

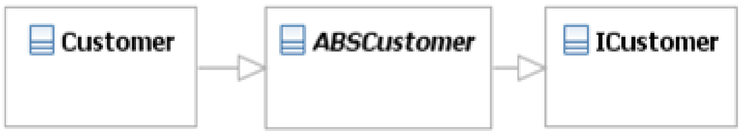


Figure 3.3-25 Original Customer Class Hierarchy

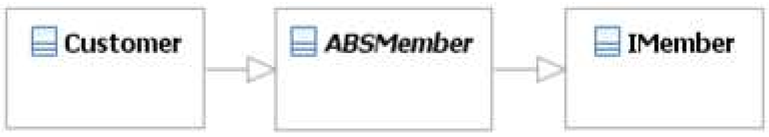


Figure 3.3-26 Verb-Preposition Replacement Class Hierarchy

3.4 Additional Modelling Considerations

3.4.1 Dependent and Independent Clauses - Relationship Detection

In the clausal constructs *(dependent/independent)* which form the sentence types *(simple, compound* and *complex)* introduced at the start of this chapter, it is the classification of clause types that are of more interest to automated modelling. The independent clause demonstrates a complete line of thought whereas a dependent clause exhibits an incomplete thought which is reliant on the independent clause for its understanding. This in its own right signifies a potential relationship between both of the subjects contained within each clause that is not discoverable through the verb of the sentence.

Identification of clause type is assisted through OpenNLP's syntactic analysis, where the tag *'SBAR'* marks the introduction of a subordinate/dependent clausal structure, and where a simple/independent clausal structure is marked by the introduction of an *'S'* tag. Therefore identification of the patterns S→SBAR, indicating a dependent clause, and the pattern S→NP, indicating an independent clause, allows the identification of both dependent/independent clausal structures.

Consider the example (Figure 3.4-1) *When a vehicle arrives at a destination,* ***the driver notifies the company***. The independent clause is in bold and upon detection of a dependent clause via its relevant pattern the ASA has to consider either a forward or a backward search for the independent clause also matched by its relevant pattern.

```
(ROOT
  (S ← Independent Clause Marker
    (SBAR ← Dependent Clause Marker
      (WHADVP (WRB When))
      (S ← Simple Clause Structure
        (NP (DT a) (NN vehicle))
        (VP (VBZ arrives)
          (PP (IN at)
            (NP (DT a) (NN destination))))))
    (, ,)
    (NP (DT the) (NN driver))
    (VP (VBZ notifies)
      (NP (DT the) (NN company)))
    (. .)))
```

Figure 3.4-1 Dependent/Independent Clause Detection

Once the independent clause is sourced, the relationship between both subjects can be identified and is constructed via this rule:

Rule 25 - Clausal Relationship Detection

If an independent clause exists and by Rule 1 or Rule 2 a class candidate exists, and if a dependent clause exists and by Rule 1 or Rule 2 a class candidate exists, then it is said an association shall also exist between both independent and dependent clause class candidates

This is only identifiable through this syntactic relationship and not through the verb of the sentence, which could result in a loss of important information. The relation will be modelled as an association relationship within UML, see Figure 3.4-2.

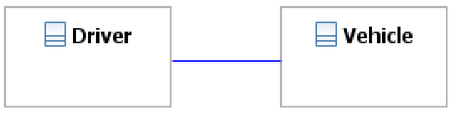


Figure 3.4-2 Dependent and Independent UML Relationship Mapping

The prime justification for creating an association relationship between the subjects of both clauses is resolved through the grammatical understanding that a dependent clause cannot stand alone and is reliant upon the independent one for its understanding. When the subjects of both clauses are the same no relationship to itself is created.

In the example: *When the company receives a call from a passenger, **the company tries to schedule a vehicle to pick up the fare.*** The independent clause is in bold and the subject of both clauses is *company* so a relationship is not required. Furthermore, neither of the *objects* contained within each clause are considered within this type of high-level relationship because they are directly affected by the verb contained within each clause and will only have a relationship with their direct subject in each separate clause. The detection and consideration of how these modelling features would be detected has been addressed in Section 3.3.3, *Verb Phrase.*

3.4.2 Additional Modelling Considerations from the Noun

3.4.2.1 Attribute Detection

Attribute detection from nouns is based upon two differing techniques based on the presence of an attribute semantic within the set of candidate semantics for that particular noun and a frequency analysis of the noun's importance to the specification.

Term Frequency (*$tf_{i,j}$*) is a technique used within information retrieval to identify the importance of a given word within a document defined by the formula in Equation 1; where *(i)* represents the term contained within the document *(j).*

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

Equation 1. Term Frequency Formula

$n_{i,j}$ is the number of occurrences of the term *(i)* in the document *(j)* and the denominator is the sum *(k)* of the occurrences of all terms contained within the document. This technique is utilised within related works as a decision making tool for class detection [**MHH89, GB94, Per02, PKS+05, Har00, HG02, VAD09]**, where in some cases the threshold can be manipulated to identify more or fewer candidate classes.

The frequencies are fixed to the document average frequency of occurrence defined by Equation 2. This is only used to make decisions in relation to attribute detection that has a below average term frequency within the context of the average term frequency for the document; where *(i)* represents term frequencies that are contained within the document *(j).*

$$dfa_{i,j} = \frac{\sum_k tf_{i,j}}{n}$$

Equation 2 Document Frequency Average Formula

The numerator is the sum *(k)* of all the individual term frequencies *(i)* for the document *(j)* and *n* is a count representing the total number of individual nouns or compound nouns contained within the document.

The term frequency technique also considers phrases (compound nouns) as a whole rather than individual terms/words. The key rationale is not to duplicate registration of both individual nouns and compound nouns within the frequency analysis and therefore not give an artificial weighting to an individual to or introduce new individual terms into the frequency analysis, which may have knock-on consequences.

The specific rules for attribute detection are defined as follows:

- Most Common Semantic Approach: (Rule 26)

Rule 26 - Attribute Detection based on Semantics

If a noun's semantic belongs to the set of attribute semantics, then that noun is considered as an attribute

- Low Term Frequency Approach (Rule 27)

Rule 27 - Attribute Detection based on Semantics, Class Candidates & Term Frequencies

If a class candidate exists as defined by Rule 1 or Rule 2, and within that noun's semantic set it also belongs to the set of attribute semantics, and the frequency count of that noun is less than the average noun frequency count for class candidates for that document, then the noun is said to be an attribute.

3.4.2.2 Class Hierarchy Detection

In addition to the set of class candidate semantics, a subset of four semantics also indicate that a class hierarchical structure should be created (see Table 3.4-1). This is primarily undertaken in an effort to give options that provide the best possibility of creating maintainable software architectures for future development.

Table 3.4-1 Class Hierarchy Subset Semantics

Noun Semantic	Description	Class Modelling Implication	
		Class	Hierarchy
Animal	Nouns denoting animals	X	X
Person	Nouns denoting people	X	X
Plant	Nouns denoting plants	X	X
Shape	Nouns denoting two and three dimensional shapes	X	X

All of the semantics listed in Table 3.4-1 typically have common features, where a *class/abstraction/interface* hierarchy would be considered constructive to include within the initial model.

For example, nouns that represent people have common features, such as a name, date of birth and address, although some of these features may not be needed for all cases. The inclusion of hierarchies allows these commonalities to be modelled within the abstraction, and consideration to be given to the use of a polymorphic approach giving rise to a more flexible and maintainable design.

Figure 3.4-3 demonstrates the type of hierarchy that would be created by automation. In the example *The company will arrange a taxi, when it receives a call from a passenger* This has the concrete class *Passenger*, abstract class *ABSPassenger* and an interface *IPassenger.*

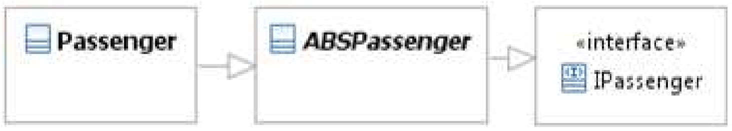


Figure 3.4-3 Automated Class Hierarchy Creation

The key role in the creation of this hierarchical structure is defined as follows:

Rule 28 - Semantic Class Hierarchical Detection

If a class candidate exits as defined by Rule 1 or Rule 2 and the semantics of the class candidate are also contained within the set of candidate class hierarchical semantics, then an interface will also be extracted for that class candidate

This approach is not saying that there are no class hierarchical structures within the remaining ten semantics from

Table 3.3-1. However, those semantic descriptions do not generically lend themselves to this interpretation and it would not be appropriate to model everything with this type of hierarchical structure. For example, the word *call* has the most common semantic type *communication* and modelling this with a *concrete/abstract/interface* hierarchy may not be appropriate.

Hierarchies that are identified during the approach will be either an interface or an interface/abstraction hierarchy. These hierarchies allow the implementation to decouple from the design allowing the implementation vary. In addition, abstractions can also cater for new functionality without impacting implementations.

3.5 Software Requirements Specification Issues in the Context of Automated Software Development.

The concept of an entirely developer-less automated analysis process is an ambitious one. However, there are some aspects of the SRS document that automation cannot manage and similarly that the human may also have some issues with, which are now further discussed.

3.5.1 The SRS Document

An SRS document contains all of the customer requirements and those requirements are elicited through various means. However, the majority start life as a narrative discussion of the problem domain. It is this document which is transformed by an analyst into a formal SRS document detailing both functional and non-functional requirements of the proposed software system. However, the formal SRS document is an expression of information in a manner that is not entirely suitable for the automated process, and the approach (discussed previously) has been geared towards the narrative specification.

The key issues associated with SRS documents for both automation and the human can be defined as follows:

- Ambiguity
- Missing Requirements
- Domain Knowledge
- Intralinguistic Variations (automation only)

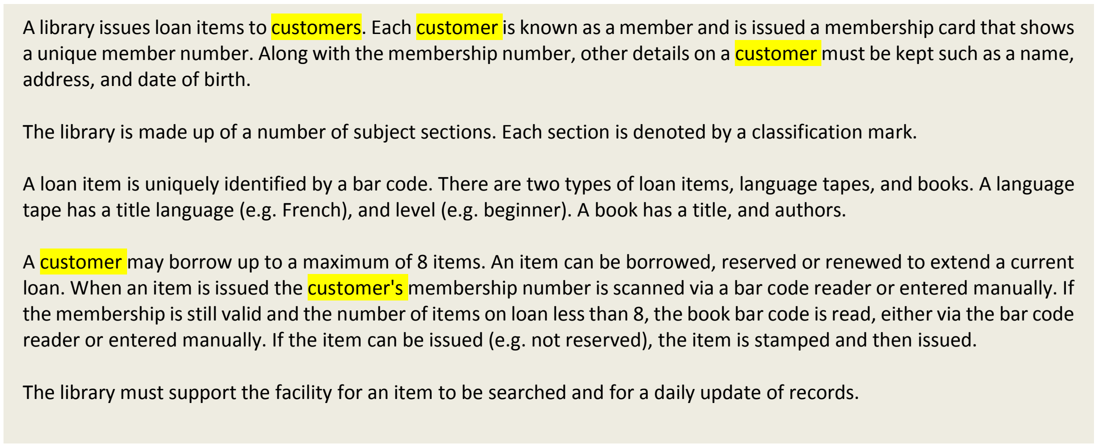
Furthermore, during the transformation from a specification into its model representation, there is a need to manage and trace the requirements not only for understanding, but to ensure every requirement has also been satisfied. The approach taken, which starts from an informal specification, makes the process of tracing requirements much more difficult due to the loss of a formal structure. However, as a by-product of the automated analysis process, it has been possible to track and tag both the requirement document and the model features generated during analysis phase, thus allowing tracing between both the specification and the model.

All of the key issues and the feature of traceability are further discussed in the following sub-sections.

3.5.2 Traceability

When the final UML model is created by automation, each of the model components identified during processing *(classes, relationships, operations, parameters and attributes)* are tagged with an identification number. This allows efficient retracing of the model feature back directly to the sentence(s) from where the component was discovered. In addition, this process gives the ability to establish the surrounding context and allows a decision to be made as to whether the model feature should be included within the final design or not.

For example, the component *Customer* (see Figure 3.5-1) has five sentence references. The traceability link allows the relevant section, paragraph and sentence to be identified. Thus, allowing an efficient and effective means of identification and understanding of the proposed model component. This subsequently allows the analyst/developer to make the final decision upon the inclusion of a component generated by automation.



A library issues loan items to customers. Each customer is known as a member and is issued a membership card that shows a unique member number. Along with the membership number, other details on a customer must be kept such as a name, address, and date of birth.

The library is made up of a number of subject sections. Each section is denoted by a classification mark.

A loan item is uniquely identified by a bar code. There are two types of loan items, language tapes, and books. A language tape has a title language (e.g. French), and level (e.g. beginner). A book has a title, and authors.

A customer may borrow up to a maximum of 8 items. An item can be borrowed, reserved or renewed to extend a current loan. When an item is issued the customer's membership number is scanned via a bar code reader or entered manually. If the membership is still valid and the number of items on loan less than 8, the book bar code is read, either via the bar code reader or entered manually. If the item can be issued (e.g. not reserved), the item is stamped and then issued.

The library must support the facility for an item to be searched and for a daily update of records.

Figure 3.5-1 Library System Requirements Specification [Cal94]

3.5.3 Intralinguistic Variations

Prior to its transformation into a formal requirements specification, decomposition or other means of transformation, software requirements are typically defined in narrative natural language form, which is the document used by the ASA. This document contains all the requirements of the proposed software system. However, where a word/noun, for instance *Customer,* is introduced, it may then be further discussed throughout the narrative specification, but by a different word such as *member*. This is a specific problem for automation where these nouns may not necessarily be synonymously related to the original word. This issue is defined as an intralinguistic variation which can result in the creation of additional/erroneous classes. This is because the syntactic analysis tool is unable to identify a link between the language variations.

An earlier technical evaluation (see Appendix C.1 – Language Inconsistency) undertaken by the author compared the models generated by the ASA against the human model (ideal solution) using four specifications that have been published in an Object Oriented text books (these models are consider as the ideal solution). The performance evaluation uncovered that each requirements specification on average contains 1.9 words that are later referenced to by different words that are not synonymously related to the original, an intralinguistic variations). If left unaddressed, this issue could easily propagate when considering other design features extracted from the language such as *attributes, operations* and *relationships.*

Intralinguistic variations can refer to many concepts, but relate particularly to referencing situations such as an endophoric reference. An endophoric reference is divided into two distinct groupings: anaphoric or cataphoric. Anaphoric refers to something in the previous text introduced and typically makes use of pronouns such as *it, they, them, me, she* or *he* for its references, whereas the cataphoric refers to something within the text, which may have not yet been identified.

For Example:

1. The library issues *loan items* to customers. A maximum of 8 *items* can be borrowed. *(cataphora)*
2. The *driver* will notify the company when *he* arrives at the destination. *(anaphora)*

Anaphoric references are resolved automatically through OpenNLP and its co-reference resolution toolkit. However, it is unable to resolve cataphoric references.

In some cases though, the referencing maybe a little more obscure. For example a word may be introduced such as *individual*, but could then be referenced throughout by six different terms *{passenger, group, fare, them, their, they}.* Even though, the term *individual* is obscure, it is possible for the human to infer from the problem domain what is meant through manual inspection, but difficult for automation as it cannot directly associate *passenger*, *group* or *fare* as an *individual*. The words *them*, *they* and *their* are resolvable through co-reference resolution supported by OpenNLP.

Therefore, the most effective way to resolve the issue of multiple intralinguistic variations is to involve the developer and create a model that allows the correct referencing of multiple words to one generic concept. Unfortunately, this is not an efficient means of minimising initial developer involvement.

The Intralinguistic Variation Model (IVM) is designed specifically to manage the cataphoric references problem and is entirely developer driven. In this approach, it is the responsibility of the developer to identify the cataphoric references contained within the software specification. The construction of the IVM model is a simple manually defined data dictionary contained within an XML file (see Figure 3.5-2). The *referent* is the word to find and to be replaced by the generalised concept *with term*. With this technique, compound nouns are also permissible which may contain more than one modifier, which can also be replaced by a compound noun. There is also a requirement to put the relevant part of speech tag with the replacement term, because it can also manage verbs that may require consideration of additional design features.

```
<referent name="membership number">
        <withTerm term="member number"/>
        <posType pos="noun"/>
</referent>
```

Figure 3.5-2 IVM Resolution

Therefore, during processing where a concept exists in the model, its replacement is easily identified, replaced and resolved through this dictionary look-up.

A manual intervention to resolve cataphoric references seems like the only logical step forward and one which would need to be created prior to automated analysis. This is because there is no other efficient means to resolve a cataphoric reference, due to the natural usage of language, where different terms are used to refer to some other term, which may or may not be synonymously related to one another. This is a major departure from the original idea: *automation should identify all relevant model features itself, without manual intervention*.

On the other hand, the only other viable solutions would be to ensure that the specification does not contain this type of reference and that a consistent language is used throughout beforehand. For example, where a feature such as *customer* is introduced, the remainder of the document must only contain references to *customer*. However, this is restrictive and adds additional effort to the process by forcing a consistent way of writing or through a rewrite of the specification.

The solution presented to resolve the issue of intralinguistic variations is not prescriptive, but the results produced from automated analysis will be affected in the ways discussed.

3.5.4 Ambiguity

Ambiguity is a difficult problem not just for automation to resolve but also for the human as well. Even though the ASA does not intend to deal with or manage ambiguity, it is worthwhile to discuss the key issues that arise from the introduction of ambiguity within a Software Requirements Specification (SRS) document.

Ambiguity typically arises when a phrase, sentence or word can be interpreted in many different ways or is taken out of context. There are two key types of ambiguity: syntactic and lexical.

3.5.4.1 Syntactic Ambiguity

Syntactic ambiguity arises when the meaning of a sentence or phrase can be interpreted in many differing ways, which affects our overall understanding of what was originally implied. This typically occurs from the varying structures implied by the sentence and the relationship between the components of the sentence such as clauses, phrases and words.

For example:

'I saw the girl with the telescope'

The example can imply that either the girl had the telescope or I used the telescope to see the girl.

The key issue associated with syntactic analysis is that there is the possibility to create erroneous model features such as, *relationships, classes, parameters, operations* and *attributes*. Given the syntactic structure for the example (see Figure 3.5-3), it can be easily seen how this can be done. The example syntactic analysis implies that it is *I* who has the telescope, but it could be the girl and here lies the specific problem.

```
(ROOT
  (S
    (NP (PRP I))
    (VP (VBD saw)
      (NP (DT the) (NN girl))
      (PP (IN with)
        (NP (DT the) (NN telescope))))
    (. .)))
```

Figure 3.5-3 Example Sentence Parse Tree

Resolution of this type of ambiguity either has to be detected by some means, which could be possible through the logarithmic probability given during sentence analysis by OpenNLP or removed explicitly and by clearly stating the actual requirements. During sentence processing, it is possible to obtain from OpenNLP the logarithmic probability indicating an opportunity for resolution. Therefore, it would be possible with this information to define a threshold which could be used to query the user as to the intended meaning of the requirement, but requires further consideration.

3.5.4.2 Lexical Ambiguity

Lexical ambiguity is the case where a single word or phrase may have multiple meanings known as polysemy. For example, the word *bear* has a polysemy count of 2 for nouns and 13 for verbs. This does present a problem when considering its true meaning and is the primary reason for choosing the common semantic model for the ASA discussed. For example:

'Bear left at the end of the road'

In this example *bear* has two very different meanings; either the animal *'bear'* has been left at the end of the road, or it is an instruction to turn left at the end of the road. The ASA in this case would create a class for *bear*, but it is difficult surmise if this is correct or not without additional contextual information. Hence, it might be possible through the surrounding context in which the word appears to find its true meaning. However, this is something this ASA does not undertake.

Current techniques utilised to resolve the ambiguity issue is known as Word Sense Disambiguation (WSD) **[IV98]**, which is an open problem in natural language processing that attempts to resolve polysemous words, but has varying degrees of accuracy.

The solution to syntactic ambiguity could be simpler compared to lexical ambiguity by being vigilant in the creation of the software requirements specification and avoiding the introduction of this issue or by means of automated detection. However, through consideration of the model developed by automation and reviewed by the developer, concerns of ambiguity could be identified within the model and corrected either by the developer or through the development of a revised specification.

3.5.5 Missing Requirements & Domain Knowledge

A problem for both human and automation is the notion of both missing requirements and domain knowledge, both of which can be interlinked. The ASA itself is entirely domain independent, being reliant on the semantic and syntactic models for the identification of design features as a starting point in the development process. Therefore, if there are requirements which are missing from the specification, then the ASA will never be able to identify these features, which is also a similar issue for a manual approach.

During the manual approach, it may or may not become apparent that there are missing requirements, which can be rectified at a later point. If not, then it is likely that this will lead to an incomplete software model that may require further iterations to resolve the situation. In a similar vein, if the requirements are missing and then later discovered after automated analysis has taken place, these can easily be included and the analysis re-run.

Personal/common knowledge of the domain can also be of benefit that can aid discovery of additional aspects related to some feature contained within the specification, which this approach does not benefit from. This information may not be included within the actual specification for a variety of reasons as it may be assumed knowledge. For example, a taxi has features that may not be discussed, but may also require modelling such as its passenger capacity. Resolution of this specific issue can be addressed by ensuring that all the relevant information is contained within the specification, or information discovered later can be included and analysis re-run, but this is easier said than done which is why domain experts are employed.

Another specific issue regarding domain knowledge are acronyms contained within specifications, such as ATM that identifies an automatic teller machine. WordNet does provide some information for some acronyms, although it is unlikely that it includes information about industry specific acronyms (for example UML). This can be similarly resolved through full definition of the actual acronym, but is again an expensive process.

A practical solution of these issues for both acronyms and specific domain knowledge would require the development of a domain dictionary or ontology that would allow the ASA to query for acronyms and knowledge that are not within the semantic model.

Even though both missing requirements and domain knowledge are not addressed during the approach, they are worthy enough of discussion in relation to the problems that they present and possible routes for resolution. Since the ASA is highly efficient (time-wise) in its analysis, it is likely that these deficiencies of both domain and missing requirements would quickly become apparent from the model produced, thus allowing manual investigation and resolution through iterative analysis and re-processing.

3.6 Implementation

This section presents a general overview of the ASA's software architecture (Figure 3.6-1). The ASA is standalone application that is implemented in Java and is operated from the command line. The ASA accepts an English natural language specification document as input and returns a UML model in XMI format according to the OMG MOF guidelines that can then be imported into a variety of CASE tools for review, validation and manipulation.

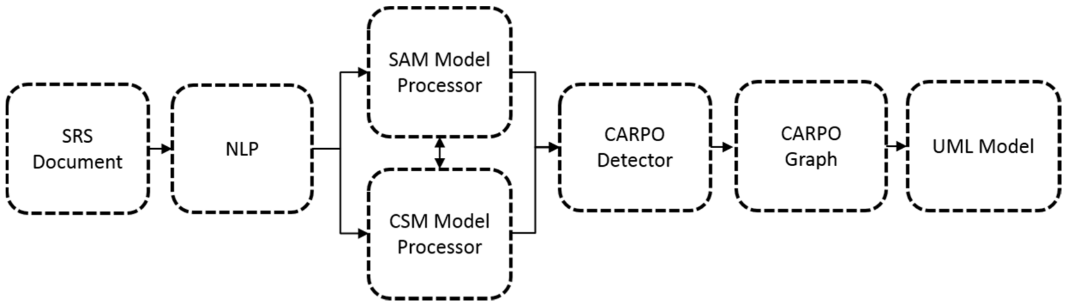


Figure 3.6-1 Automated Software Architect Automation Process

The ASA has two key external dependencies to ensure correct operation: the natural language processing toolkit OpenNLP **[Mor07]** that aids syntactic analysis and WordNet **[Mil95]** a data dictionary that assists with semantic analysis

Essentially the process is to take the original data and either perform some transformation and or identify additional information for use within the Class, Attribute, Relationship, Parameter and Operation detector (CARPOD) processor. The processes defined within CARPOD can identify the relevant model features which are then subsequently stored in a graph data structure which is later used by a UML processor to construct the UML model.

The implementation is split into two key stages: data gathering and detection phases; the goal of the data gathering phase is to ensure that all relevant information is extracted and is available for the CARPO detector.

The data gathering process can be defined as follows:

- Document Structural Analysis and Transformation
- Word Data Collection

Document structural analysis seeks to understand how the requirements specification is structured in terms of paragraphs and sentences. This understanding allows the specification to be transformed into an XML document which can be used to trace decisions made by the ASA back into the requirements. The transformation process from an SRS document into a XML document is automatic and Figure 3.6-2 defines the corresponding document type definition (DTD).

```
<!ELEMENT document       (paragraph+)>
<!ELEMENT paragraph      (#PCDATA|s)*>
<!ELEMENT s              (#PCDATA)>

<!ATTLIST document       id CDATA  #REQUIRED>
<!ATTLIST paragraph      id CDATA  #REQUIRED>
<!ATTLIST s              id CDATA  #REQUIRED>
```

Figure 3.6-2 SRS to XML Document Type Definition

The Document Type Definition (DTD) preserves the integrity of the document by allowing sentences to be associated to their relevant paragraph. Each of the paragraphs and sentences are tagged with an identifier that aids traceability. Paragraphs are identified by pattern matching blocks of text followed by a blank line, where the natural language parser will identify all of the individual sentences. A separate process rewrites the SRS into an XML document. Figure 3.6-3 exemplifies an SRS that has been transformed into XML format.

```
<document id='0'>
        <paragraph id='1'>
                <s id='p1.0'>A library issues loan items to customers.</s>
                <s id='p1.1'>Each customer is known as a member and is issued a
                membership card that shows a unique member number.</s>
        </paragraph>

        <paragraph id='2'>
                <s id='p2.0'>The library is made up of a number of subject
                sections.</s>

        </paragraph>
</document>
```

Figure 3.6-3 Sentence and Traceability Link Identification

With the structure of the document identified and the parse tree constructed by OpenNLP (see Figure 3.2-2) the approach can then start to construct the relevant syntactic and semantic information that is needed for use within the CARPO process. Starting from the perspective of the individual word, Figure 3.6-4 identifies the relevant information that needs to be known prior to CARPO processing.

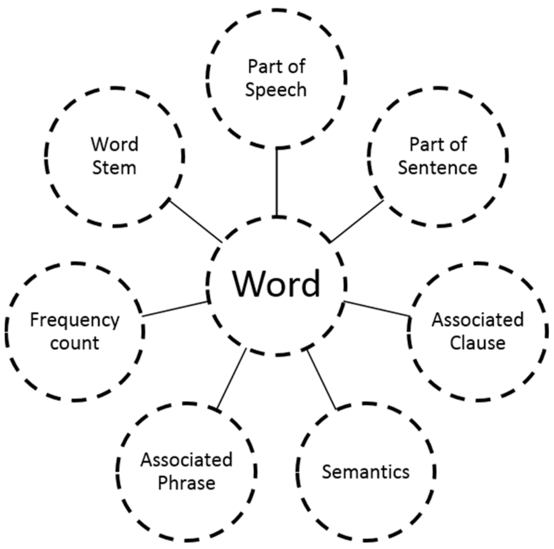


Figure 3.6-4 Word Data Collection

In essence, the OpenNLP parse tree is reconstructed along with the additional information identified during word data collection, which is then made available to the CARPO detector for processing.

The CARPO process then performs an in-order traversal of the newly constructed parse tree and using the Common Semantic and Syntactic models, discussed throughout this chapter, the relevant candidate features of the model can be identified and stored within a graph structure for transformation into XMI data format.

Figure 3.6-5 demonstrates the graph that is used to store candidate model features once they have been identified during the detection phase. A graph node includes a trace identifier, can have multiple edges and is used to manage candidate class, attribute or operation information; edges represent additional information such as the relationship, operation and parameters, multiplicity and traceability information which can be used to connect to other nodes or the node itself.

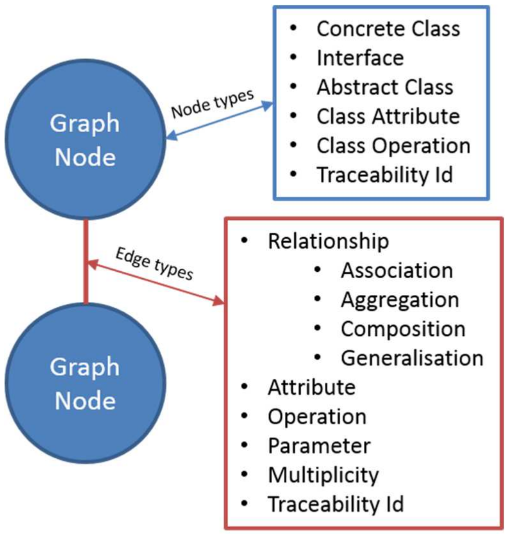


Figure 3.6-5 CARPO Graph Data Construct

Finally, every node and edge contained within the graph maintains its own traceability information so that decisions undertaken by the approach can be traced directly to the sentence from where it came from. Once the graph is fully constructed it can then be traversed and an XMI representation of the UML model can be constructed and used within a variety of case tools for review.

3.6.1 Worked Example

This section will work through the transformation process from textual requirements specification through to UML Model of the ASA's approach to automation. The following specification, Figure 3.6-6, originally identified in the related works **[Har00]** the specification contains 3 paragraphs, 7 sentences and 100 words and is in domain of hospital **[Duf95].**

A local hospital consists of many wards, each of which is assigned many patients.

Each patient is assigned to one doctor, who has overall responsibility for the patients in his or her care. Other doctors are assigned on an advisory basis. Each patient is prescribed drugs by the doctor responsible for that patient.

Each nurse is assigned to a ward and nurses all patients on the ward, though is given special responsibility for some patients. Each patient is assigned one nurse in this position of responsibility. One of the doctors is attached to each ward as an overall medical advisor.

Figure 3.6-6 Local Hospital Requirements Specification [Duf95]

This specification is processed by the approach and transformed into the following XML document representation, Figure 3.6-7:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE document SYSTEM "RQSDTD.dtd">
<document id='0'>
   <paragraph id='1'>
       <s id='p1.1.0'>A local hospital consists of many wards, each of
which is assigned many patients.</s>
   </paragraph>
   <paragraph id='2'>
       <s id='p2.1.0'>Each patient is assigned to one doctor, who has
overall responsibility for the patients in his or her care.</s>
       <s id='p2.1.1'>Other doctors are assigned on an advisory basis.</s>
       <s id='p2.1.2'>Each patient is prescribed drugs by the doctor
responsible for that patient.</s>
   </paragraph>
   <paragraph id='3'>
       <s id='p3.1.0'>Each nurse is assigned to a ward and nurses all
patients on the ward, though is given special responsibility for some
patients.</s>
       <s id='p3.1.1'>Each patient is assigned one nurse in this position
of responsibility.</s>
<s id='p3.1.2'>One of the doctors is attached to each ward as an overall
medical advisor.</s>
   </paragraph>
</document>
```

Figure 3.6-7 XML representation of Requirements Specification

Once the transformation into the XML document is complete, each of the sentences contained within the specification are processed individually to identify their relevant syntactic structure as demonstrated within Figure 3.6-8. It is this syntactic structure that makes it possible to extract relevant candidates and apply the rules that have been identified and discussed throughout this thesis.

```
(ROOT
  (S
    (NP (DT A) (JJ local) (NN hospital))
    (VP (VBZ consists)
      (PP (IN of)
        (NP
          (NP (JJ many) (NNS wards))
          (, ,)
          (SBAR
            (WHNP
              (NP (DT each))
              (WHPP (IN of)
                (WHNP (WDT which))))
            (S
              (VP (VBZ is)
                (VP (VBN assigned)
                  (NP (JJ many) (NNS patients))))))))))
    (. .)))
```

Figure 3.6-8 Syntactic Parse Tree Example

At the very lowest processing level by the ASA, it is the parse tree which is at the core which is traversed to identify the sentence structure and apply the rules discussed throughout this thesis where appropriately.

Both Table 3.6-1 and Table 3.6-2 identify all the candidate nouns, and verbs from the traversal of the parse tress, their semantics and candidate UML features that they may represent from the resulting semantic model

Table 3.6-1 Noun Class Candidates

Identified Nouns	Semantic 1	Semantic 2	UML Feature(s)
Position	Location	Attribute	Class
Patient	Person	-	Class/Hierarchy
Basis	Relation	-	N/A
Ward	Person	Artefact	Class/Hierarchy
Nurse	Person	-	Class/Hierarchy
Advisor	Person	-	Class/Hierarchy
Care	Act	-	N/A
Hospital	Artefact	-	Class
Drug	Artefact	-	Class
Responsibility	Act	Attribute	N/A
Doctor	Person	-	Class/Hierarchy

Table 3.6-2 Verb Relationship/Operation Candidates

Identified Verb	Semantic	UML Feature(s)
Are	Stative	Operation
Is	Stative	Operation
Assigned	Social	Relationship/Operation
Given	Possession	Relationship/Operation
Attached	Contact	Relationship/Operation
Prescribed	Communication	Relationship/Operation
Consist	Stative	Operation

The candidates contained within both tables and that are confirmed by the rule set discussed in this thesis are subsequently added to the CARPO graph structure. Figure 3.6-9 demonstrates a textual representation of that graph for the *Patient* class candidate that was identified by the ASA. It demonstrates the key class nodes and any relevant in our out edges for that note and what those edges describe, which may be relationships, attributes, parameters or more.

Figure 3.6-9 CARPO Graph Extract

```
INTERFACE: IPatient :MAIN_NODE: TRACE_ID: '[p1.1.0, p2.1.0, p2.1.2, p3.1.0, p3.1.1]
INTERFACE: IPatient :RELATION:
 INTERFACE: IPatient :RELATION_IN_EDGES_FROM:
  INTERFACE: IPatient :IN_NODE: 'IDoctor' TYPE: 'interface' EDGE_TYPE: 'Association'
  INTERFACE: IPatient :IN_NODE: 'Drug' TYPE: 'class' EDGE_TYPE: 'Association'
  INTERFACE: IPatient :IN_NODE: 'INurse' TYPE: 'interface' EDGE_TYPE: 'Association'
 INTERFACE: IPatient :RELATION_OUT_EDGES_TO:
  INTERFACE: IPatient :OUT_NODE: 'ABSPatient' TYPE: 'abstract' EDGE_TYPE: 'Generalisation'

ABSTRACT: ABSPatient :MAIN_NODE: TRACE_ID: '[p1.1.0, p2.1.0, p2.1.2, p3.1.0, p3.1.1]
ABSTRACT: ABSPatient :RELATION:
 ABSTRACT: ABSPatient :RELATION_IN_EDGES_FROM:
  ABSTRACT: ABSPatient :IN_NODE: 'Responsibility' TYPE: 'attribute' EDGE_TYPE: 'attribute'
  ABSTRACT: ABSPatient :IN_NODE: 'assigned' TYPE: 'operation' EDGE_TYPE: 'operation'
 ABSTRACT: ABSPatient :RELATION_OUT_EDGES_TO:
  ABSTRACT: ABSPatient :OUT_NODE: 'Patient' TYPE: 'class' EDGE_TYPE: 'Extends'

CLASS: Patient :MAIN_NODE: TRACE_ID: '[p1.1.0, p2.1.0, p2.1.2, p3.1.0, p3.1.1]
CLASS: Patient :RELATION:
        CLASS: Patient :RELATION_<IN-NULL>:
        CLASS: Patient :RELATION_<OUT-NULL>:
```

With the candidates confirmed and the CARPO graph loaded with all the relevant information at this point the automation process has essentially completed. All that remains now is to transform the CARPO graph data into a UML Model, Figure 3.6-10.

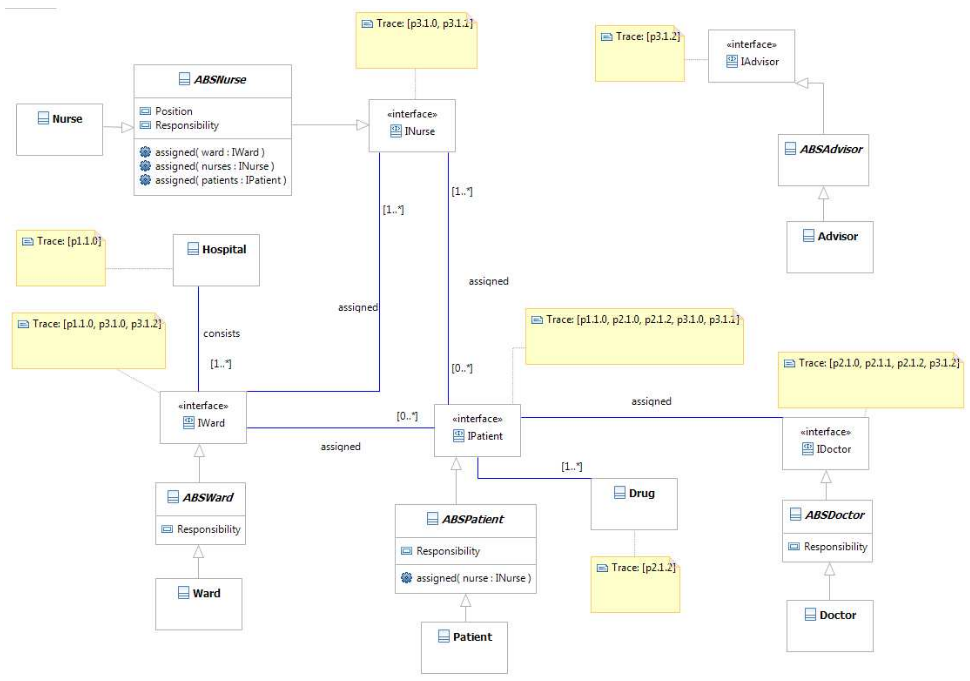


Figure 3.6-10 ASA version of Local Hospital Problem

The creation of the UML model in Figure 3.6-10 is built using the Eclipse MDT UML2 API **[MDT14]** and is constructed by traversing the CARPO graph where the information contained within each node and edge is subsequently mapped to the corresponding UML element as assisted by the UML. Finally, the model that is constructed is fully editable by the analyst

3.6.2 System Architecture

There is no user interface associated with Automated Software Architect and the ASA is started from the command line with the user passing the file location of the document to be process. The ASA will then process this document that will finally results in the creation of a UML model. Figure 3.6-11 demonstrates a high-level view of the Automated Software Architect (ASA) as previously discussed at the start of this chapter.

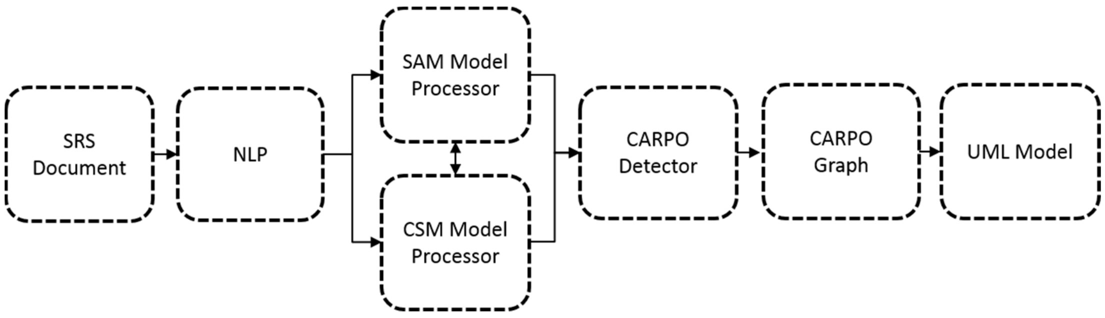


Figure 3.6-11 Automated Software Architect Automation Process

Figure 3.6-12 identifies each of the top level packages that also have a corresponding UML Model contained within Appendix A.5. Each of the UML models details the ASAs architecture at a greater level of detail. Table 3.6-3 maps the top level packages to the overall ASA Process as defined previously.

Table 3.6-3 ASA Process to Package Mapping

Architecture Component	Package Name	Purpose
SRS Document	uk.ac.strath.sd.xml uk.ac.strath.sd.tree	Transform document to suitable format and mange traceability links
NLP	opennlp.tools.lang.english	OpenNLP Toolkit Interface
SAM Model Processor	uk.ac.strath.sd.nlp	Detection of POS and language process
CSM Model Processor	uk.ac.strath.sd.nlu uk.ac.strath.sd.wordnet	Application of semantic models WordNet Interface
CARPO Detector	uk.ac.strath.sd.model	Application of static/dynamic processing rules
CARPO Graph	uk.ac.strath.sd.jccg	Data structure managing detected features
UML Model	uk.ac.strath.sd.uml	Generation of UML editable diagram

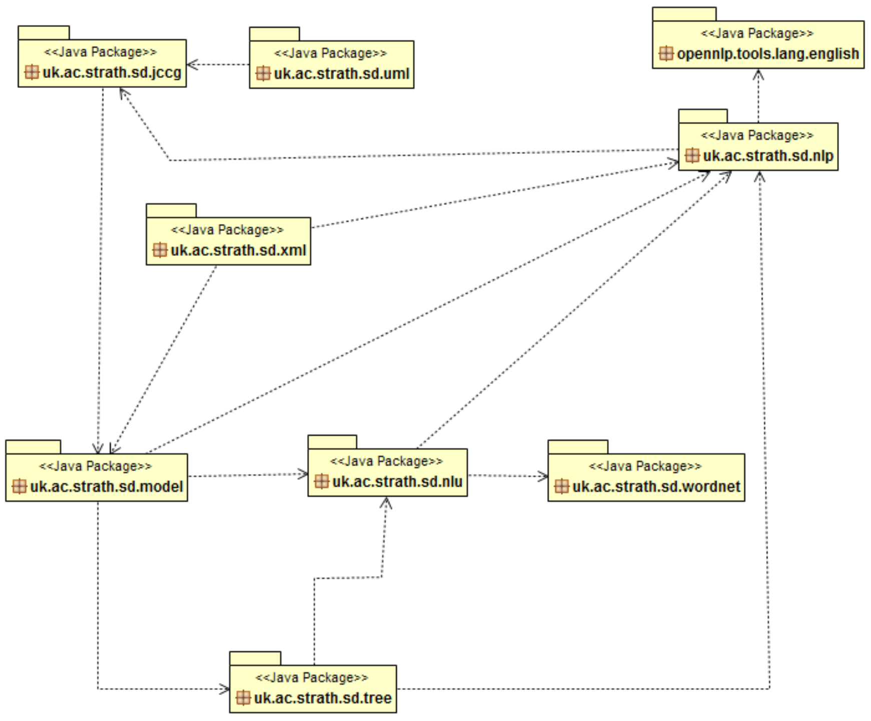


Figure 3.6-12 Top Level Package Diagram

Along with the UML and package models, a sequence diagram detailed in Figure 3.6-13 demonstrates the flow of processing within the ASA towards the creation of a UML model.

The process is started from the command line and is done so by passing the location of the textual representation of the requirements specification document. This then goes through a series of processing steps that transforms the document into a XML representation containing its relevant paragraphs and individual sentence(s) along with their associated traceability links.

The document is then processed and transformed at the sentence level where a tree representing how the sentence is constructed in terms of its individual parts of speech and semantic meanings is maintained. The information contained within the tree structure can then be processed and the appropriate candidate model features are detected by means of the static/dynamic rules applied by the CARPO detector and added to the CARPO graph. The CARPO graph is subsequently traversed and the information contained within is transformed into the final UML model for the system under analysis along with the relevant traceability information that allows the identification from which sentence the feature was detected.

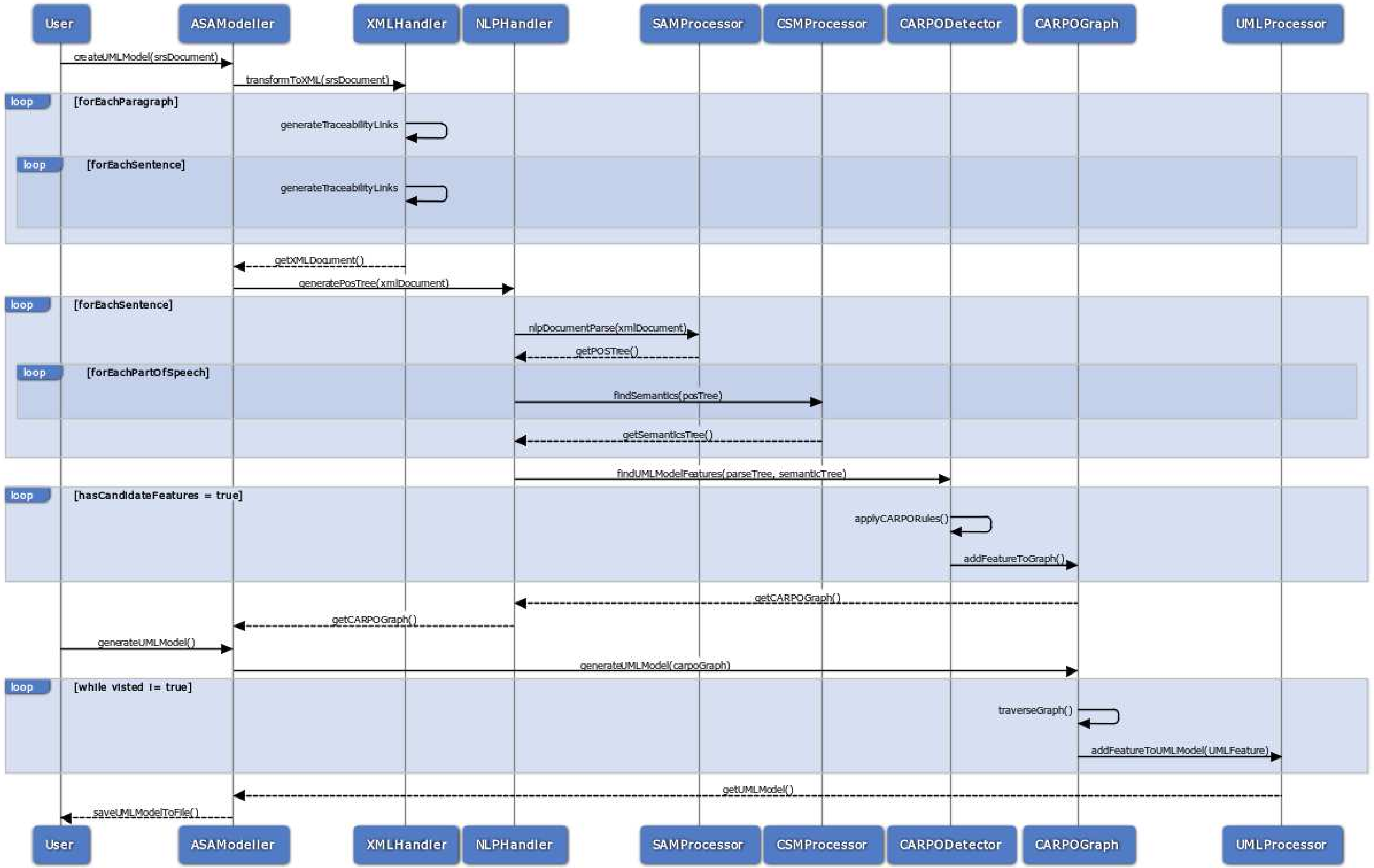


Figure 3.6-13 ASA Sequence of Events

3.7 Conclusion

This chapter has presented both the semantic and syntactic models used to analyse a narrative natural language software specification on a sentence-by-sentence basis. This is in pursuit of creating an initial/conceptual UML model automatically from the natural language in an efficient and effective manner, alleviating the burdensome approach of manual analysis.

The semantic model is used to detect relationships, attributes, operations, class inheritance hierarchies and classes and is based upon common semantic model that best imply specific design features assisted by WordNet Classifications.

The syntactic model, in combination with the semantic model, utilises OpenNLP to retrieve the syntactic structure of each sentence. The ASA itself has base rules used to identify classes, attributes, relationships, operations, multiplicities, class inheritance hierarchies and parameters from the syntactic structures. Furthermore, through analysis of the high-level syntactic structures, it is possible to identify relationships at the clause level without the need of a main verb.

This chapter concludes with a discussion of the key issues affecting both manual and automated analysis in relation to the software specification document. These issues specifically relate to ambiguity, missing requirements, domain knowledge, intralinguistic variations and tracing requirements from specification to model and vice versa. Finally, the chapter gives an overview of the implementation from data gathering, detection through to final UML XMI model and a detailed view of the applications overall architecture.

Chapter 4
Evaluation

4.1 Introduction

The techniques introduced and discussed in the previous chapter endeavour to identify the relevant model components from all the information contained within a natural language requirements specification document. The features of the initial model, whether it be classes, relationships, operations, attributes or parameters, require different considerations in the context of automated syntactic and semantic analysis when being evaluated for inclusion within the initial model. This chapter evaluates the implementation of the proposed technique towards automated analysis and initial model creation in the context of the key model features: classes and relationships. It does not consider operations, attributes or parameters in this evaluation, the reason for which are discussed later in this chapter. Finally, the chapter concludes with a comparative evaluation of its most closely related works.

One of the key issues that became apparent with related work was the lack of evaluation within related approaches **(see Chapter 2).** Some of the related works' evaluations used techniques taken from the domain of Information Retrieval (IR) and employed metrics such as recall and precision **[Mic96, MMZ02, MG02, KZM+04, Har00, HG02].** Others utilised simple evaluation techniques such as a comparative evaluation of what is versus what is not created with no concrete metrics to validate the results **[GB94, IO05, IO06, OI06, BSC06, BCA06, BSM09, PRM+07, VAD09]**. This evaluation will also exploit IR evaluation techniques. The evaluations utilising these techniques only consider the class detection process and do not evaluate other aspects of the design such as relationships, attributes, or operations. This evaluation will only investigate class and relationship detection and will not consider operation, parameter or attribute detection. This is because the model created by automation is defined only as an initial model, whereas the actual (ideal) model used in the comparison has likely been refined and where features such as operations, parameters and attributes may have been identified by other means such as domain knowledge, personal experience that automation does not have access to. The approach taken towards automated software modelling has no automated refinement phase and is entirely dependent on what information is made available at time of processing implying that if it is not in the textual specification it will not be considered by the automated approach.

The evaluation undertaken herein is based upon the work by Harmain **[Har00]**, where the performance evaluation is broken into three distinct parts:

- Firstly, a performance evaluation utilising Precision, Recall and Over-Specification; Over-specification, defined by Harmain, is a measurement of additional class candidates or relationships that are not contained within the ideal model, but are deemed useful by their context within the text and the author's own judgement.
- Secondly, results from the performance evaluation are used to investigate and identify any key issues that may arise from the requirements specification itself and that may impact the automated approach towards successful identification of all relevant model features.
- Thirdly, the evaluation will also undertake a comparative evaluation of the overall technique in the context of the most closely related works *CM-Builder* developed by Harmain **[Har00]** and *NL-OOPS* developed by Mich **[Mic94].**

4.2 Evaluation Background

The performance evaluation undertaken here was originally used by Harmain and Gaizauskas **[Har00, HG02]** for the evaluation of their *CM-Builder* implementation. This utilised a corpus of requirements specifications and also introduced a comparison between *NL-OOPS* **[Mic94]** and *CM-Builder.* Furthermore, Harmain's comparative evaluation does not measure the performance of both approaches through the measures of recall and precision, but is a high level review of the comparative models created by both approaches.

The evaluation undertaken here uses the same requirements specification corpus of both Harmain and Mich. In addition, additional requirements specifications have been identified and added to the corpus.

Harmain's evaluation is similarly based upon the work of Hirschman and Thompson **[HT95]** who identify that the evaluation of natural language processing is crucial for both system developers as well as users. They broadly define three kinds of evaluation; Adequacy, Diagnostic and Performance evaluations. Harmain only utilises the performance evaluation methodology that has a well-defined structure amenable to quantitative analysis and which sets out what will be evaluated, how this will be measured and how the values for the measure will be identified

The method step within the performance evaluation asks *How do we determine the appropriate value for a given measure and a given system?* To address this Harmain uses the metrics precision and recall which are both widely accepted within the IR community and defined in Girshman and Sundheim **[GS96]**.

There are differences between both IR and automated requirements analysis and model creation which Harmain and Gaizauskas **[Har00, HG02]** identifies as follows: An IR system extracts specific information that is based upon a search criterion such as named entities, partial parses and text simplification. In contrast, the approaches of CM-Builder, NL-OOPS and the ASA do not have a pre-defined search criteria.

The key answers in IR represent correctly extracted information by a human analyst, whereas the key answer (i.e. the original class model) represents a model of the problem domain, but there is no way to know for sure if an accurate model of the problem domain exists.

These differences (text simplification, named entities, and key answers) identify why it is difficult to evaluate NLP CASE tools and can possibly explain why other related works have not attempted any form of evaluation.

4.3 Evaluation Methodology

The performance evaluation methodology (below) defined by Hirschman and Thompson **[HT95]** addresses the three important aspects that must be considered by any evaluation. The overall aim is to evaluate the performance of a system in some area of interest and the steps are defined as follows:

- **Criterion:** What are we interested in evaluating?
- **Measure:** What specific property of the system performance will we report in an attempt to get to the chosen criterion?
- **Method:** How do we determine the appropriate value for a given measure and a given system?

Utilising these techniques the evaluation methodology is defined as follows:

- **Criterion:** The criterion applied to evaluate the ASA is to evaluate how close the models produced by automation are in comparison to that of the human developed model (i.e. *The Ideal Solution*). However, there is no ideal solution in existence for a given problem domain as two different developers could, and can, create radically different models based upon their own domain knowledge and expertise. Neither can these models be considered as correct or incorrect, but they can be categorised as good or bad. Harmain addresses this issue by simply stating that a good model is one that has been published within an Object-Oriented text book and is therefore considered as the ideal solution. Additionally, for the evaluation it has also been assumed online works can be included within the definition of an ideal solution.

- **Measures:** The measures of Recall, Precision, F-measure and a further metric defined by Harmain and Gaizauskas **[Har00, HG02]** Over-Specification are used to evaluate the performance of both class and relationship detection.

    - **Over-Specification:** Is a measure that addresses the inclusion of additional class/relationships that are not contained within the *Ideal Solution*. The extra elements are identified during the detection process by automation, and after consideration of the element within the context of their passage, their inclusion within the initial model maybe considered warranted because they represent features of the system that a developer may wish to model. The concept of the ideal solution used in this evaluation represents a model that may have already been through several design iterations to realise the final design. Therefore, it is more than probable that the *Ideal Solution* has already been through this kind of over-specification phase and through design iterations much of the additional information classified as 'extra' could have already been considered and removed through this iterative design process.

- **Method:** All classes and relationships in the manually created model (known as the *Ideal Solution*) and the automation model (known as the ASA) are compared with each other manually. The following categories are used to manually classify each element in the respective models.

- **Correct/True Positive (TP):**
  - **Class Detection:** A class found by the ASA is considered *correct/true positive,* if it exactly matches an element in the *Ideal Solution*. If no exact match exists but by the author's judgement an element exists in the ASA that is synonymously related to an element in the ideal solution, it is also therefore considered correct.
  - **Relationship Detection:** An association relationship found by the ASA is also considered *correct/true positive* if both classes are *TP* and the relationship exists within the ideal model.
- **Incorrect/False Positive (FP):**
  - **Class Detection:** An element found by the ASA is said to be *incorrect/false positive,* if it is not in the *Ideal Solution* and both the problem statement and the author's judgement confirm that it is wrong.
  - **Relationship Detection:** An association relationship found by the ASA is said to be *incorrect/false positive,* if either of the classes involved within the relationship have also been identified as *incorrect/false positive* or the relationship does not exist within the *Ideal Solution*.
- **Missing/False Negative (FN):** an element is said to be *missing/false negative,* if it is contained within the *Ideal Solution,* but has not been identified by the ASA algorithm. This stands true for both class and relationship detection.
- **True Negatives (TN):** represents class candidates that have been considered and deemed irrelevant by the ASA and are not included within the *Ideal Solution*.
- **Extra (E):** An element is said to be an *extra* if it is not contained within the *Ideal Solution,* but by the author's own judgement and aided by the context of the problem statement and identified by the ASA it demonstrates a useful concept for consideration. This stands true for both class and relationship detection.

- **Synonymous:** An element in the ASA is said to be synonymously related to another *correct/true positive* answer if, from its position within the context of the specification, it is used as a synonym for a *correct/true positive* answer. This only stands for class detection (see 3.5.3 Intralinguistic Variations). In the context of the *Ideal Solution* this is assumed to be automatically considered either consciously or unconscious bias by the domain analyst

With these method definitions defined in terms of the *Ideal Solution*, it is now possible to define the following measures:

Recall is a measure related to a positive outcome, which identifies the fraction of relevant instances that are retrieved and is defined as follows:

$$recall = \frac{TP}{TP + FN}$$

Precision identifies the accuracy of the system in terms of the fraction of the retrieved instances that are relevant and is defined as follows:

$$precision = \frac{TP}{TP + FP}$$

F-measure a further measure is also utilised during this evaluation known as the *F-Measure* **[Rij79]**. This measure is based on the harmonic mean of Precision (P) and Recall (R) and is defined as follows:

$$F - measure = \frac{2 \cdot P \cdot R}{P + R}$$

When interpreting these results the closer the score is to 1 the better the result; conversely the closer the result is to 0, the worse the system performs in terms of Precision, Recall and its overall effectiveness.

Over-Specification identifies the fraction of elements that are included over and above the elements that are correctly identified by the approach, but are not contained within the ideal model and is defined as follows:

$$\textit{Over-Specification} = \frac{E}{TP + FN}$$

4.3.1 Threats to Validity

4.3.1.1 External Threats

External threats to validity aims to address how the approach relates to the real world commonly known as its generalisability. This evaluation has based itself upon the notion of using the ideal solution where there exists a model and English requirements specification that is published either in a reputable online source or in a book. One of the key reasons for choosing the ideal solution is that no industrial specifications were/are typically available for public research. In addition, if a sample of industrial specifications were obtained – with model and specification text, it would likely not be as representative as the systems that have been identified and utilised during this evaluation.

The specifications used within the evaluation are from a variety of differing domains such as banking, hospital management, ticketing systems, library management, other management systems and computer games all chosen to try and test the generalisability of the approach. The average specification size is 273 words and therefore not representative of large scale software systems. The largest specification is 1508 words and smallest is 65 words (see Section 4.4 for additional corpus information). Nevertheless, the evaluation remains disadvantaged as a result of not having any industrial specifications which is a threat to the approaches overall validity.

4.3.1.2 Internal Threats

The approach taken has two key threats to internal validity, firstly, the manual classification of the categories *Extra* and *Synonymous* by the author. This threat does not affect the classification of *Correct,* as an answer key exists to guide the classification process and anything that is not in the answer key is hence *Incorrect*. Secondly, the corpus itself is of limited size, limited domains and are text book examples.

The manual classification threat to validity concerns the notion of unconscious bias when classifying Extra and Synonymous from those deemed as Incorrect. Unconscious bias relates to a situation where the classification of an answer say as *Incorrect,* which should be classified as *Extra,* can affect the outcome of the results. Therefore, both the manual classifications and the possibility of unconscious bias could lead to a situation that may skew the overall results, in a positive or negative manner. This issue can only be resolved through an impartial classification of the results. Even though the author has a vested interest in presenting the results in their best light, impartial as the author has tried to be, this threat persists. Therefore, all the classifications of the method categories have been undertaken by the author in an impartial manner as possible.

The specification corpus main threat to validity concerns the author's selection bias of the specifications to ones that best suit the approach. However, the majority of this bias can be negated as 13 out of the 17 specifications have been taken from related works so that comparative analysis of the approaches performance could be considered. The remaining specifications that have been selected by the author but have been identified from course work material: Gizmo Ball **[MIT05]** because of its size, complexity and additional non relevant information, Cinema System **[CIS08]** because of its size and completeness, KWIC **[Par72]** because of its small size and the ambiguities it introduces and finally Taxi **[BAR12]** because of its simplicity and conciseness. Additional threats such as the size of the specifications utilised within the evaluation (see Table 4.4-1) and the limited domain. The average number of sentences per specification is 17 with an average word count of 273 (which for industrial specifications is likely to be perceived as trivially small) and they cross nine different domains. However, to allow an effective comparison to be made between the approach and the most closely related works, utilisation of related works' corpus specifications is a necessity. Mitigation of these threats would be to obtain industrial specifications, but in the majority of cases industrial specifications are unattainable due to them being covered by Non-Disclosure Agreements.

4.4 Evaluation Corpus

The corpus utilised in this evaluation consists of a total of seventeen specifications; eight used by CM-Builder, two used by NL-OOPS and seven identified by the author. All specifications fall within the concept of the *Ideal Solution* and each has a UML model with the exception of the NL-OOPS, which is missing one model. However, NL-OOPS does provide a key for this model and the UML models from the remaining specifications can be used to generate the key and evaluate the results for both Class and Relationship detection.

The specifications are all from different domains and range from 65 to 1500 words and from 3 to 94 sentences. Each specification takes the form of a narrative English description of the system and its associated functionality. In addition, each of the specifications also has an associated UML model with the exception of specification number 10. The specifications are detailed in Appendix B.1 and Table 4.4-1 covers key information relating to the specifications. Appendix B.2 details the individual classification results and Appendix B.3 details the UML models created by automated analysis.

Table 4.4-1 Evaluation Corpus Details

No	Specification Name	Domain	Word Count	Sentence Count	Used by Approach	Reference
1	Filing Problem	Electronic Filing Management	231	12	ASA/CM-Builder	Der95
2	Library Problem 3	Book Management	154	9	ASA/CM-Builder	Cur95
3	Journal Registration Problem	Journal/User Management	153	7	ASA/CM-Builder	Duf95
4	Hospital Problem 2	Room Management	283	23	ASA/CM-Builder	Duf95
5	Hospital Problem 1	Patient Management	100	7	ASA/CM-Builder	Cur95
6	Library Problem 2	Loan Management	217	15	ASA/CM-Builder	Cur95
7	Organisation Problem 2	Staff Management	100	6	ASA/CM-Builder	Cur95
8	ATM Problem	Banking System	136	8	ASA/CM-Builder/NL-OOPS	RBP+91
9	Library Problem 1	Library Loan System	193	16	ASA/NL-OOPS	EP98
10	Meeting Problem	Sports League Management	179	17	ASA/NL-OOPS	RBP+91
11	Cinema Problem	Seating & Ticketing	561	25	ASA	CIS08
12	Gizmo Ball	Computer Game	1508	94	ASA	MIT05
13	Organisation Problem 1	Course Administration	130	12	ASA	
14	Exam Problem	Exam Marking	354	22	ASA	
15	Keyword in Context	Document Indexing	65	3	ASA	Par72
16	Lift Problem	Lift Control	181	11	ASA	PRM+07
17	Taxi Problem	Taxi Management	106	7	ASA	BAR12

4.5 Results

This section presents the results obtained for the system under evaluation in terms of Precision (P), Recall (R), Over-Specification (OvS) and F-Measure (FM). It is broken into two separate sections - one for class detection and the other for relationship detection - both of which have a comparative evaluation with related works.

Table 4.5-1 demonstrates the average performance from both class and relationship detection evaluations.

Table 4.5-1 Class and Relationship Detection Performance Summary

Classification	Recall	Precision	Over-Specification	F-Measure
Class Detection Average Performance	**0.73**	**0.60**	**0.47**	**0.64**
Relationship Detection Average Performance	**0.26**	**0.33**	**0.68**	**0.28**

The results for class detection demonstrate a positive step towards automated analysis of natural language requirements specifications and even though class detection achieves a high performance level in terms of recall and precision there is still room for improvement. The results achieved for both recall and precision in the context of relationship detection are not so encouraging. Both result sets for over specification also produce some interesting results, especially in the context of relationship detection, which is very high. It is the key intention of this evaluation to identify, discuss and highlight the primary reasons that may be prohibiting the ASA approach from achieving better results in the context of both class and relationship detection.

4.5.1 Class Detection Performance Results

Figure 4.5-1 boxplots the data contained within Table 4.5-2 and demonstrates the performance of the automated approach.

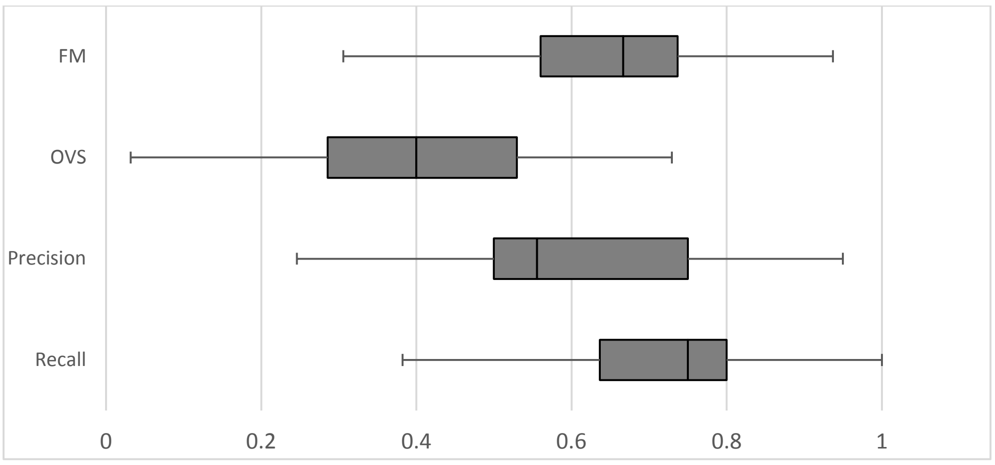


Figure 4.5-1 ASA Performance Results

The data within Table 4.5-2 details the raw performance figures for the candidate class detection process of the automated approach. The figures also considers the standard deviation which is used to understand the spread of the data. In the case of the ASA's approach the standard deviation value should be as small as possible as this demonstrates that the techniques employed for candidate class are not volatile and can produce consistent and reliable results. Overall, the approach performs rather will with an average recall rate of 73% and precision of 60% but does some have key lows where specification 12 the largest specification in terms of word and sentence count, 5 times bigger than the average specification, used in this evaluation but also has the lowest performing result. Whereas specification 13 (not the smallest specification overall) is 40% smaller in size when compared to the average sentence and word count but performs the best in terms of recall, but not so in terms of precision. The following sections investigates the results a little deeper.

Table 4.5-2 Individual SRS Performance Results

Specification No.	TP[1]	FN[1]	FP[1]	E[1]	S[1]	Recall	Precision	OVS	FM
1	6	4	5	4	2	0.60	0.55	0.40	0.57
2	4	4	4	7	0	0.50	0.50	0.88	0.50
3	4	1	5	6	1	0.80	0.44	1.20	0.57
4	8	1	7	3	2	0.89	0.53	0.33	0.67
5	4	1	0	3	0	0.80	1.00	0.60	0.89
6	7	1	4	2	2	0.88	0.64	0.25	0.74
7	3	1	1	1	1	0.75	0.75	0.25	0.75
8	9	3	6	2	3	0.75	0.60	0.17	0.67
9	5	2	4	3	1	0.71	0.56	0.43	0.63
10	7	4	7	3	0	0.64	0.50	0.27	0.56
11	8	5	16	12	3	0.62	0.33	0.92	0.43
12	13	21	28	18	8	0.38	0.32	0.53	0.35
13	5	0	10	2	0	1.00	0.33	0.40	0.50
14	16	5	5	7	2	0.76	0.76	0.33	0.76
15	4	2	1	3	1	0.67	0.80	0.50	0.73
16	8	2	5	3	1	0.80	0.62	0.30	0.70
17	6	1	0	2	5	0.86	1.00	0.29	0.92
Average Performance						0.73	0.60	0.47	0.64
Standard Deviation						0.15	0.20	0.27	0.15

4.5.2 Class Candidate Results Investigation

Although the results themselves are encouraging, further investigation into the root causes of not being able to achieve high levels of both recall and precision are sought. Given that a software requirements specification (SRS) may contain all relevant and correct information, it should be possible to achieve high levels for both recall and precision. Figure 4.5-2 details the total raw data for all SRS documents processed during this evaluation. The key areas that will be investigated further are False Negatives and False Positives as these are the key to increasing both recall and precision.

[1] TP –True Positive, FN – False Negative, FP – False Positive, E – Extra, S - Synonymous

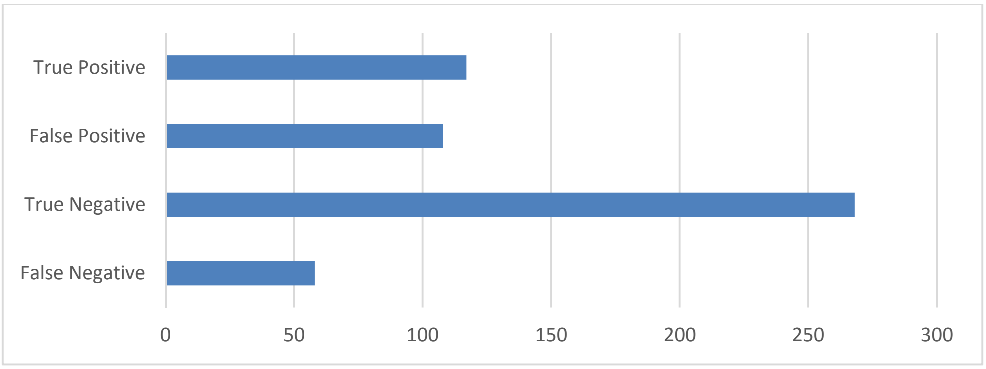


Figure 4.5-2 All SRS Raw Classification Data

Before discussing the key issues analysis, it is important to address an interesting finding associated with false negatives. The classification of a false negatives states that: *an element is considered a false negative if it is not discovered by the approach and is contained within the ideal model's answer key*. However, in some cases the answer key in the original model contains class candidates that are not actually defined within the requirements specification text at all. For example, Hospital Problem 2 **[Duf95]** the original model has a *Staff* class which is not defined in the specification texts (see Appendix B.1**)**. This raises the question as to how this class candidate has been discovered in the original model and it is only through deeper investigation of false negative classifications (see Appendix B.2**)** has this issue come to the forefront.

Figure 4.5-3 classifies false negatives (FN) in to two groups, FN(P) and FN(NP) where FN(P) identifies false negatives that are present within the requirements specification text, and FN(NP) that represents false negatives that are not present within the requirements specification text.

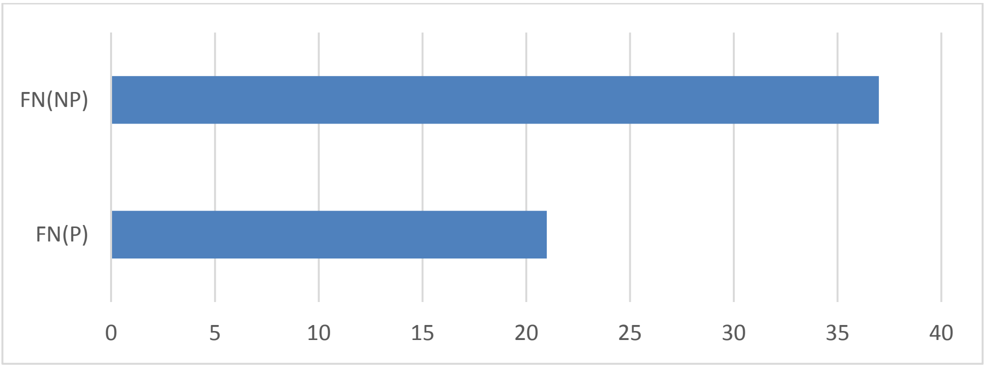


Figure 4.5-3 Present/Not Present False Negative Classifications

The analysis has identified that 64% of all false negatives, FN(NP), are not contained or defined in the requirements specification text by any means and how they are being identified within the ideal model is an unknown. Without this information contained within the textual specification it will be impossible for automation to successfully identify all relevant class candidates and as a result recall and precision will be negatively impacted. Since FN(NP) are not contained in the specification text this has identified an opportunity to accurately reflect the ASA performance by discounting FN(NP) from the performance investigation however, the results and findings of this are discussed later in this section.

4.5.2.1 False Negative and Positives Issue Analysis

Both false positive and false negative issues are very closely related as a result they are discussed under the one section as both groups have such similar issues in the areas of *semantics*, *Rule 27* and *NLP*. Figure 4.5-4 quantifies and identifies the root causes for false negatives (that are present within the specification) and false positives (See Appendix B.4 for the raw data).

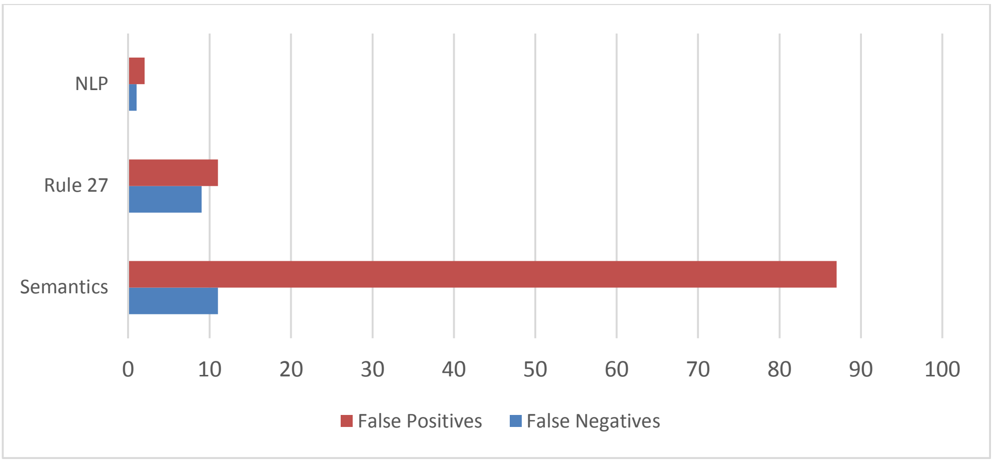


Figure 4.5-4 Key Issues Analysis of False Positives and Negatives

Semantic Issues:

In the case of false negatives there are semantics that are out with the set of candidate classes (Table 3.3-2), and for false positives the opposite - semantics that are within the set of class candidates (

Table 3.3-1). The semantics issue for both false positives and negatives represents the majority of their issues and in the case of false positives this is a key problem. Semantics is a core feature of the approach within the ASA and is used to decide what should and should not be created as a class candidate.

Table 4.5-3 demonstrates the semantic issue for both false positives and negatives, it details the issue type, the word used in the specification and its most common semantic used to identify whether it's a class candidate or not.

Table 4.5-3 False Negative and Positive Semantic Issues

Issue Type	Word	Most Common Semantic
False Negative	transaction	Action
False Negative	loan	Possession
False Negative	competition	Action
False positive	description	Communication
False positive	platform	Artefact
False positive	hypotenuse	Shape

In the case of false positives a class candidate is created when it should not have been and for false negatives a candidate class should be created but is not. In both cases, the issue is related directly to the semantics that are used to determine whether a class candidate should be created or not.

Each word also has additional semantics for differing contexts and Table 4.5-4 details the additional semantics that are available and whether those semantic could be used to create a candidate class, if the approach were able to disambiguate.

Table 4.5-4 False Negative and Positive Associated Semantics

Issue Type	Word	Associated Semantics	Associated Semantics indicate Class Candidate
False Negative	transaction	None	-
False Negative	loan	communication	Yes
False Negative	competition	Event, Person	Yes
False positive	description	Communication, Cognition	Yes
False positive	platform	Communication	Yes
False positive	hypotenuse	None	-

The majority of false negatives have a semantic that would allow the creation of a candidate class and disambiguation could lead to a possible solution, but the same can also be said of false positives. As a result, resolution via disambiguation may not be an optimal solution in all cases and an alternative solution needs to be sought to address the core issue of false positives.

Rule 27 Issue:

Rule 27 is a rule that is used to transform a class candidate into an attribute during the automated analysis and model generation process which is defined as follows:

If a class candidate exists as defined by Rule 1 or Rule 2, and within that noun's semantic set it also belongs to the set of attribute semantics, and the frequency count of that noun is less than the average noun frequency count for class candidates for that document, then the noun is said to be an attribute.

In the case of false negatives, the candidate classes should remain as a candidate class, but are being transformed into attributes because they meet the criteria defined by Rule 27. Where in the case of false positives they have a class candidate semantic and also have a high term frequency and remain as a class candidate when they are actually better suited to being attributes. Therefore, the frequency analysis for the Rule 27 is called into question. This process of transforming a class candidate into an attribute may not be appropriate and employing a different strategy, such as identifying/marking potential attributes rather than actually transforming them into attributes, may be a better approach.

NLP Issues:

The NLP toolkit, OpenNLP, identifies all the relevant parts of speech (POS) for each word however, in some cases the identification goes awry and the incorrect POS is identified. Table 4.5-5 gives examples of the issue, the word from the specification, the correct part of speech and the POS identified by the NLP toolkit.

Table 4.5-5 NLP Issues

Issue Type	Word	Correct POS	NLP Identified POS
False Positive	interact	Verb	Noun
False Positive	coarser	Adjective	Noun
False Negative	output	Verb	Verb

The issue with false positives is that they are being created as candidate classes when they should not be because WordNet identifies the incorrect part of speech. The false negative issue only impacts one specific word, *output*, where in the ideal model this is identified as candidate class, but since the word is identified as a verb it is never considered as a potential class candidate by the approach, which is correct and the NLP toolkit is correct in its POS identification. Further investigation reveals that the approach does process *output*, but creates an action rather than a class. Whether this is correct or not is open to interpretation and the problem domain. It is important to remember that the ideal models may have been already been through many design iterations and the operation *output* may have been extracted as a candidate class, which better suits the problem domain. In this case very little can be done unless some form of domain knowledge is to be created and used along with candidate processing.

4.5.2.2 Impact of discounting missing class candidates

The investigation of False Negatives has identified that 64% of all false negatives that are missing are not actually defined within the specification texts. Since they are not defined within the specification texts it is impossible for automation to create class candidates. This has led to considering what the impact on overall performance would be if false negatives not present within the specification deducted from the total false negative counts and whether this give a more accurate picture of actual performance.

Table 4.5-6 details the raw analysis data, where FN(P) represents false negatives that are actually stated within the specification document, but missed by automation. Where FN (NP) represents false negatives that are not stated within the specification document. The combination of both FN(P) and FN(NP) will give the original false negative count see Table 4.5-2

Table 4.5-6 Individual SRS Performance Results (with FN present only)

Specification No	TP	FN(P)	FN(NP)	FP	E	S	Recall	Precision	OVS	FM
1	6	2	2	5	4	2	0.75	0.55	0.50	0.63
2	4	2	2	4	7	0	0.67	0.50	1.17	0.57
3	4	0	1	5	6	1	1.00	0.44	1.50	0.62
4	8	0	1	7	3	2	1.00	0.53	0.38	0.70
5	4	0	1	0	3	0	1.00	1.00	0.75	1.00
6	7	0	1	4	2	2	1.00	0.64	0.29	0.78
7	3	1	0	1	1	1	0.75	0.75	0.25	0.75
8	9	3	0	6	2	3	0.75	0.60	0.17	0.67
9	5	2	0	4	3	1	0.71	0.56	0.43	0.63
10	7	4	0	7	3	0	0.64	0.50	0.27	0.56
11	8	4	1	16	12	3	0.67	0.33	1.00	0.44
12	13	4	17	28	18	8	0.76	0.32	1.06	0.45
13	5	0	0	10	2	0	1.00	0.33	0.40	0.50
14	16	2	3	5	7	2	0.89	0.76	0.39	0.82
15	4	1	1	1	3	1	0.80	0.80	0.60	0.80
16	8	0	2	5	3	1	1.00	0.62	0.38	0.76
17	6	0	1	0	2	5	1.00	1.00	0.33	1.00
Average Performance							0.85	0.60	0.58	0.69
STDEV							0.14	0.20	0.37	0.16

What is uncovered is an average performance increase across the key metrics with no improvement in precision, an 18% improvement in recall, and a 3% increase in over-specification is realised. Figure 4.5-5 demonstrates an analysis that compares the original results against those that has class candidates not defined in the specification text removed from the false negatives.

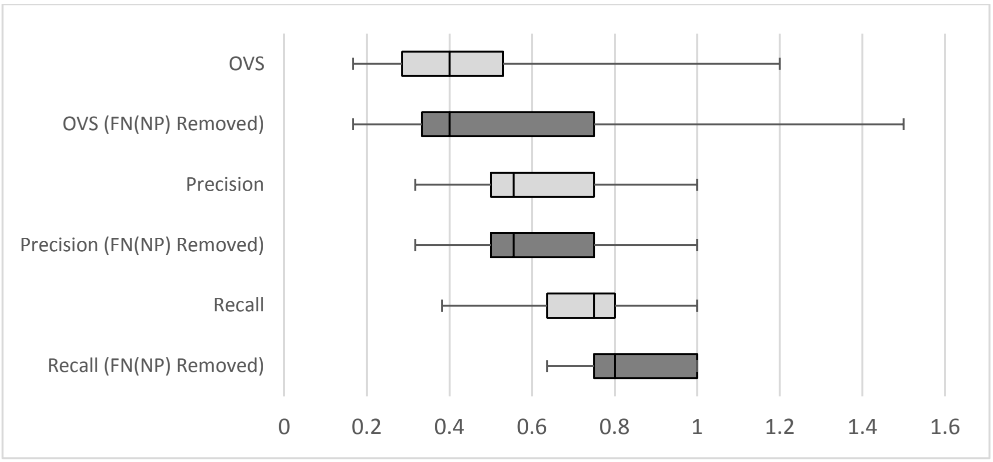


Figure 4.5-5 ASA Performance Analysis (FN (NP) Removed)

Table 4.5-7 details the raw analysis numbers for the box plots and it is clear that discounting false negatives that are not contained within the specification FN(NP) improves standard deviation and the average for recall overall.

Table 4.5-7 ASA vs. ASA (FN(NP) Removed) Class Detection Raw Data

Measure	Minimum	Quartile 1	Median	Quartile 3	Maximum	STDEV	Average
Recall	0.38	0.63	0.75	0.80	1	0.15	0.73
Recall (FN(NP) Removed)	0.64	0.75	0.80	1	1	0.14	0.85
Precision	0.32	0.50	0.56	0.75	1	0.20	0.60
Precision (FN(NP) Removed)	0.32	0.50	0.56	0.80	1	0.20	0.60
OVS	0.17	0.28	0.40	0.52	1.2	0.27	0.47
OVS (FN(NP) Removed)	0.17	0.30	0.40	0.80	1.50	0.37	0.58

4.5.2.3 Class Detection Conclusion

Overall the ASA's approach towards class detection is a positive step forward out of 551 potential class candidates analysed by the approach: ~21% can be directly associated with true positives, false negatives represents ~11%, ~19% are associated with false positives, and ~49% are associated with true negatives.

In addition to the evaluation and issues analysis, the investigation of the results has identified that 22% of all class candidates contained within ideal model are being identified by some other means as those missing classes are not detailed within the actual textual requirements specification. Furthermore, the issue of missing classes that are stated within the textual requirements specification and contained within the ideal model represents only ~12%.

Resolution of the *false negatives* is key to increasing the overall recall of the approach and this has been demonstrated by a further performance investigation that does not consider false negatives that are not present within the specification and removed these from the calculation process. This demonstrated a positive relative change in recall performance by 16%, but to actually achieve this gain the missing information has to be included within the specification document at the time of processing. Additional gains can be achieved by addressing the key issues also identified during the evaluation. Similarly there was a small increase in precision of 3% but to increase precision substantially false positive issues will have to be addressed as well.

4.5.3 Relationship Detection Performance Results

The performance evaluation, in the context of the ideal model, also considers the detection results of ASA in terms of the relationships discovered and are also evaluated through usage of the same metrics. In addition there is also a direct comparison with CM-Builder & NL-OOPS, even though these related works do not evaluate relationship detection themselves citing that the identification of relationships from the specification is too variable. CM-Builder and NL-OOPs do provide some initial system models generated by their systems, therefore it has been possible to obtain the results from these models using exactly the same methodology as applied to obtain the ASA results.

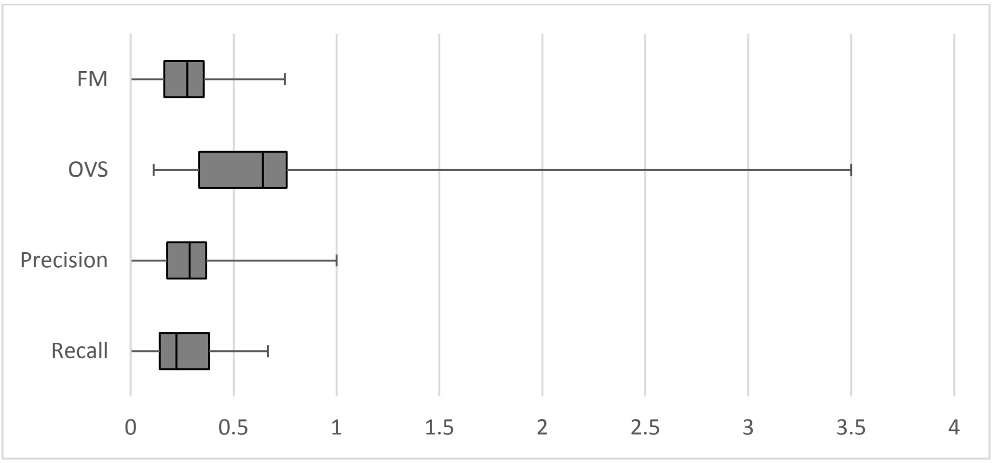


Figure 4.5-6 ASA Relationship Performance Analysis

Table 4.5-8 identifies the overall performance for relationship detection as identified by the approach; where Figure 4.5-6 boxplots the performance figures that have been automatically analysed by the approach. Overall the performance is poor with an average recall of 28% and precision of 36% and only in two cases does it achieve a recall rate of 60% or more.

Table 4.5-8 Individual Relationship Detection Performance Results

Specification No	TP	M	FP	E	Recall	Precision	OVS	F-Measure
1	2	7	3	3	0.22	0.40	0.33	0.29
2	2	5	4	7	0.29	0.33	1.00	0.31
3	1	4	4	4	0.20	0.20	0.80	0.20
4	3	4	2	5	0.43	0.60	0.71	0.50
5	3	2	0	4	0.60	1.00	0.80	0.75
6	1	2	3	1	0.33	0.25	0.33	0.29
7	2	7	6	1	0.22	0.25	0.11	0.24
8	4	13	8	4	0.24	0.33	0.24	0.28
11	1	13	16	9	0.07	0.06	0.64	0.06
12	4	34	22	27	0.11	0.15	0.71	0.13
13	2	22	12	5	0.08	0.14	0.21	0.11
14	4	2	10	2	0.67	0.29	0.33	0.40
15	0	2	1	7	0.00	0.00	3.50	0.00
16	3	14	6	9	0.18	0.33	0.53	0.23
17	5	4	0	6	0.56	1.00	0.67	0.71
9	0	7	0	0	0.00	0.00	0.00	0.00
10	0	0	0	0	No Model Available			
Average Performance					**0.26**	**0.33**	**0.68**	**0.28**
Standard Deviation					**0.19**	**0.29**	**0.78**	**0.21**

The main focus of the relationship evaluation is to consider why the majority of relationships are missing and the root cause of why there are missing relationships.

4.5.4 Relationship Results Investigation

This section of the evaluation aims to identify the root causes as to why the performance is so bad by looking to answer the questions, *Are these relationships defined within the specification?* and *Does the fault lie with the technique employed in identifying relationships?*

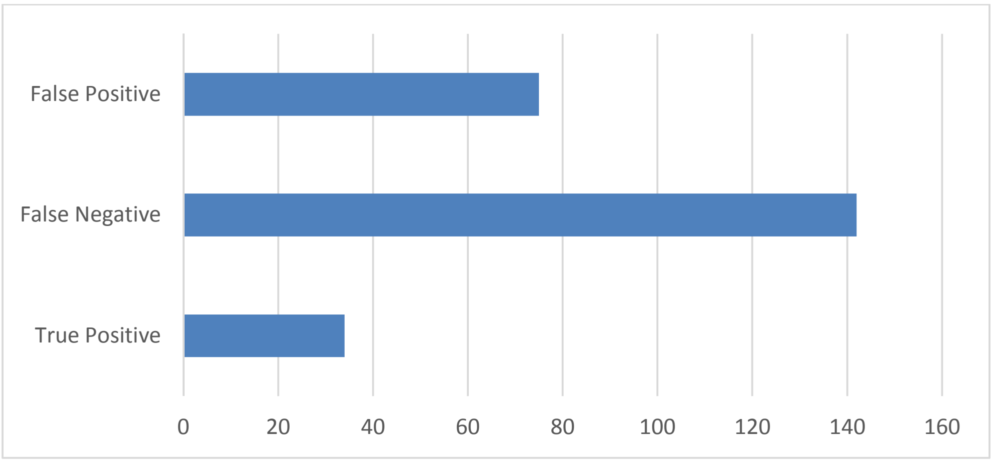


Figure 4.5-7 ASA Raw Classifications for Relationship Detection Analysis

Figure 4.5-7 illustrates the total raw data for all SRS documents processed during this evaluation. It identifies the correct identifications and a count of which relationships are missing. False positives are present in this view, but are associated directly with the detection of false positive class candidates, and consequently the relationship identified as a result of a false positive class candidate is also deemed incorrect. Therefore, if the detection of false positive class candidates can be eliminated, then the identification of the false positive relationships will also be eliminated as well. The key area that will be investigated further will be false negatives.

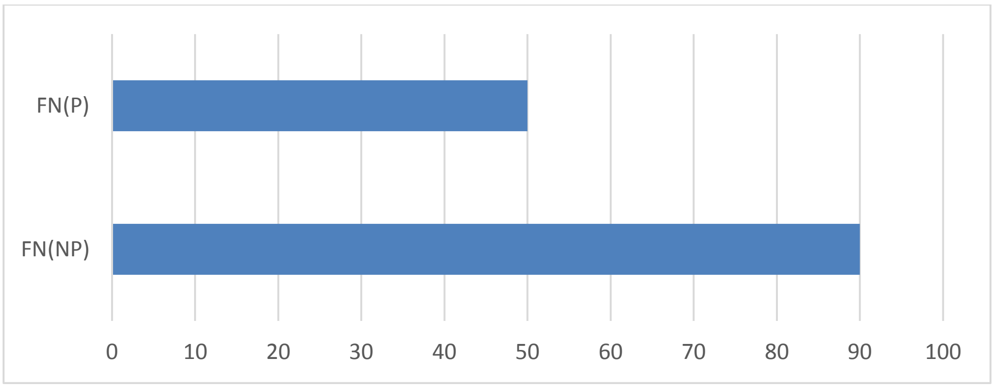


Figure 4.5-8 False Negative Classifications for Relationship Analysis

Figure 4.5-8 decomposes false negatives into relationships that are detailed within the specification FN(P) and those that are not FN(NP), as previously defined. The detection of false negatives that are not present within the specification also correlates to class candidates that are not present within the specification as they are required to complete the discovery of the actual relationship. Therefore, the investigation of false negative relationships themselves identified as FN(NP) will not be investigated any further as it is impossible for automation to detect something that is not present. However, as with candidate class detection, an investigation into the actual performance that discounts false negatives will be undertaken.

Figure 4.5-9 further classifies each false negative that are present within the specification (FN(P) that has been identified during the evaluation process into the following categories: *ASA Model* and *Domain Understanding* (see Appendix B.5 Raw Classification Data).

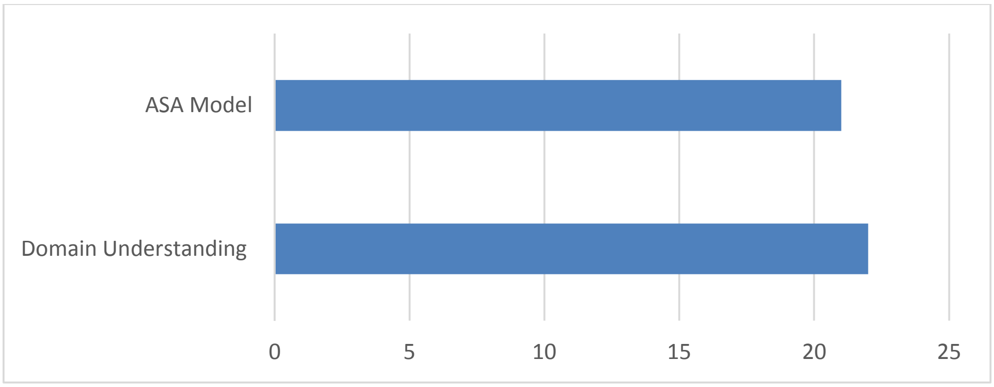


Figure 4.5-9 False Negative (P) Classification Analysis for Relationship Analysis

These classifications have been obtained by using the original UML models created by their respective authors, and the relationships identified in those models have been used to compare against the relationships identified by the ASA and are discussed in the remaining sections of this chapter

Domain Understanding

Figure 4.5-10 classifies each of the domain understanding issues these are typically hierarchical, such as interfaces and concrete implementations, missing relationships where classes are present in the specification, but are spread across paragraphs or different sentences, and bidirectional relationships.

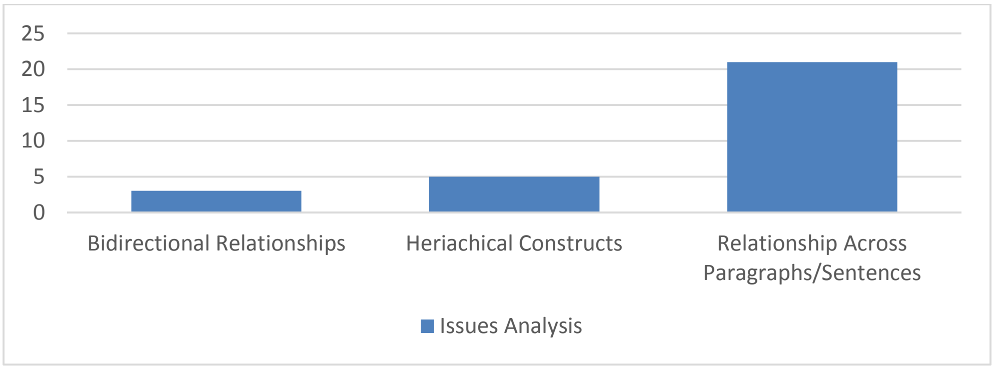


Figure 4.5-10 Domain Understanding Issues Analysis

Hierarchical relationships are only associated with one specific specification. This raises the question of whether it is an issue to be concerned about. The following examples demonstrate the key point.

1) *There are two types of* ***loan items***, ***language tapes***, *and* ***books***. **[Cal94]**

2) *A passenger on m-th floor calls a lift by pressing the **up** or **down button*** **[PRM+07]**

The approach the ASA takes to extract and identify relationships is by means of the main verb and/or syntactic constructs, and the hierarchical relationships in the examples, *language tape* and *books are type of loan item* and *up* and *down buttons are a type of button* are not identifiable by this technique because there is no verb or syntactic construct that identifies the relationship. The relationship is identified by an understanding of the problem domain which therefore makes it difficult to identify and extract the hierarchical relationship given the current approach taken by the ASA, but in both cases there is a pattern that emerges: *x and y are a type of z*. Further investigation of these patterns could aid the discovery of additional relationships.

The final issue identified through the evaluation is associated with relationships across paragraph and sentences. Since the ASA operates on a sentence by sentence basis, it can only identify relationships are within that spectrum,. Given the situation where candidate relationships are spread across sentences and/or paragraphs there is no connecting verb or syntactic structure that is local (i.e. within the sentence) that indicates this relationship is present. Only through understanding of the domain, context and processing larger chunks of domain text would it possible for these relationships to be identified and extracted.

ASA Model

Figure 4.5-11 breaks the model issues into their respective issue groups. These issues relate directly to the actual sentence that is being analysed for candidate relationships as both classes are present, but the approach fails to identify the candidate relationship.

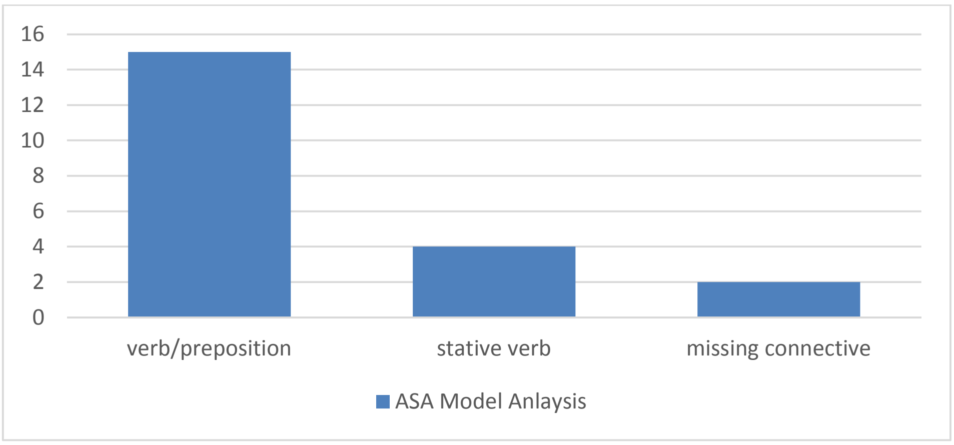


Figure 4.5-11 ASA Model Issue Classifications for Candidate Relationship Detection

The crux of problem is associated directly with verb/preposition syntactic construct combinations and the connection to their most closely related noun. In others, the type of verb contained within the sentence fails to identify the relationship or the key noun has not been discovered.

For Example:

1) *Each **instructor** works for one **department** and each **department** has at least one **instructor**.* **[Organisational Problem 1]**

2) *One of the **doctors** is attached to each **ward** as an overall medical advisor.* **[Cur95]**

3) ***Documents** may be filed along with **keywords**, authors, and/or a document description or abstract describing the document.* **[Der95]**

Examples *1, 2* and *3* demonstrate that the verb/preposition combinations, *works for, attached to* and *filed along with* fail to identify the relationships between both candidate classes within the sentence. Even though there is a mapping, Rule 21, that has been derived to aid decision making of verb/preposition combinations, those decisions result in the creation of an action rather than a relationship. It therefore raises the question as to whether those mappings are correct, or that in some situations dual interpretations are possible which need to be identified and managed.

Finally, a minority of issues surround some edge cases such as stative verbs and syntactic constructs that have not been considered in these contexts. So this raises a question as to whether these are actually issues as they are only present within one or two specifications.

For example:

1) ***Questions** may have multiple **parts**, and partial credit may be awarded for parts correctly answered.* [**Exam Problem**]

2) *Furthermore, it should be possible for readers to find **articles** which deal with **topics** that they are interested in.* **[Cur95]**

In the examples above, the elements in bold are identified as classes and have association relationships between them. The first example contains the stative verb *have,* which is used by the ASA to identify attributes and not relationships, but in this case it created a class which is incorrect and is an impact of Rule 27 application (see Section 4.5.2). The second example has no connecting verb between them because the second part *which deal with topics* is contained within a subordinate clause attached to the word *articles*, which the ASA misses altogether.

4.5.4.1 Impact of discounting missing relationship candidates

The investigation of false negatives has identified that 63% of relationships defined within the ideal model are missing as a result of a missing candidate classes as these candidate classes are not actually defined within the requirements specification texts, as previously discussed. Would excluding the relationships that are associated to class candidates that are not present within the specification or model this demonstrate actual relationship detection performance?

Table 4.5-9 SRS Relationship Results (with FN present only)

Specification No	TP	FN(P)	FN(NP)	FP	E	Recall	Precision	OVS	F-Measure
1	2	1	6	3	3	0.67	0.40	1.00	0.50
2	2	0	5	4	7	1.00	0.33	3.50	0.50
3	1	2	2	4	4	0.33	0.20	1.33	0.25
4	3	2	2	2	5	0.60	0.60	1.00	0.60
5	3	1	1	0	4	0.75	1.00	1.00	0.86
6	1	0	2	3	1	1.00	0.25	1.00	0.40
7	2	5	2	6	1	0.29	0.25	0.14	0.27
8	4	6	7	8	4	0.40	0.33	0.40	0.36
11	1	3	10	16	9	0.25	0.06	2.25	0.10
12	4	7	27	22	27	0.36	0.15	2.45	0.22
13	2	10	12	12	5	0.17	0.14	0.42	0.15
14	4	2	0	10	2	0.67	0.29	0.33	0.40
15	0	2	0	1	7	0.00	0.00	3.50	0.00
16	3	7	7	6	9	0.30	0.33	0.90	0.32
17	5	3	1	0	6	0.63	1.00	0.75	0.77
9	0	1	6	0	0	0.00	0.00	0.00	0.00
10	0	0	0	0	0	No Model Available			
Average Performance						0.46	0.33	1.25	0.36
Standard Deviation						0.28	0.29	1.05	0.23

Table 4.5-9 details the raw analysis data, where FN (P) represents false negatives that are actually stated within the specification document, but are missed by automation. Where FN (NP) represents false negatives that are not stated within the specification document. The combination of both FN(P) and FN(NP) will give the original false negative count see Table 4.5-8.

Missing class candidates that are not defined within the specification do have a negative impact on performance, and when considering what is only defined within the specification it becomes clear that the performance of the approach is better. Recall performance increase ~179%, precision sees no performance improvement at all and over-specification increases greatly. Figure 4.5-12 ASA Relationship Performance (FN (NP) Discounted) compares the performance of recall, precision and over-specification when

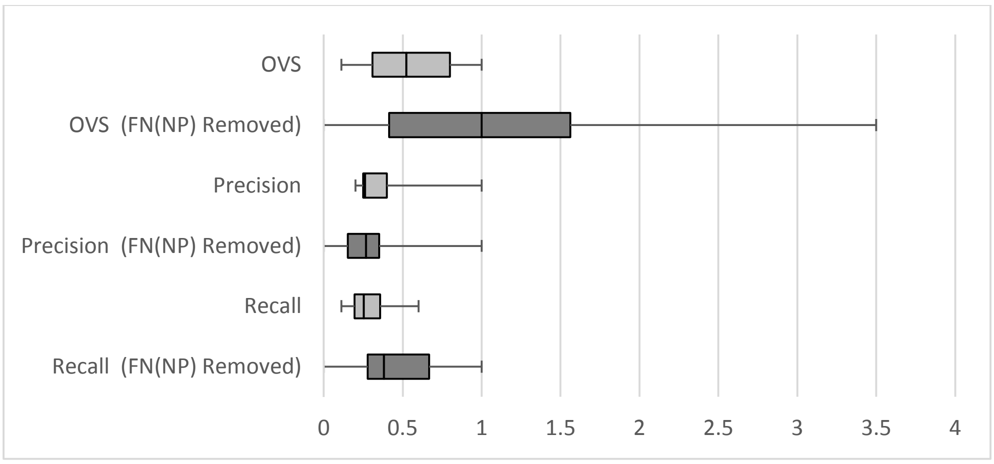


Figure 4.5-12 ASA Relationship Performance (FN (NP) Discounted)

Table 4.5-10 demonstrates the raw data for the previous box plots.

Table 4.5-10 ASA vs. ASA (FN(P) only) Relationship Detection Raw Data

Measure	Minimum	Quartile 1	Median	Quartile 3	Maximum	STDEV	Average
Recall (ASA)	0.11	0.19	0.25	0.35	0.6	0.19	0.26
Recall (ASA – FN(P))	0	0.27	0.38	0.66	1	0.28	0.46
Precision (ASA)	0.2	0.25	0.26	0.4	1	0.29	0.33
Precision (ASA – FN(P))	0	0.15	0.27	0.35	1	0.29	0.33
OVS (ASA)	0.11	0.30	0.52	0.80	1	0.78	0.68
OVS (ASA – FN(P))	0	0.41	1	1.56	3.5	1.05	1.25

As with candidate class detection an increase in recall is realised however, a steep increase in standard deviation by almost ~47% is also noted when ignoring relationships that are not defined within the specification as a result of missing candidate classes. It does demonstrate that overall performance is being masked by the inclusion of false negatives that are not present within the specification for both recall and over-specification, but what is not identifiable is whether the approach would actually find the relationship if the candidate classes were to be included.

4.5.4.2 Relationship Detection Conclusion

The relationship detection technique only identifies a very small number of relationships, and in some of those cases the relationships are not discovered due to the class candidate being a false negative. Even if the false negative class candidate had been correctly identified in the first place, it does not actually demonstrate that the relationship would have been identified. It is shown that the majority of missing relationships are not actually defined within the specification text and are being identified by some other unknown means.

The next key finding is domain understanding where the class candidates are present. However, since the class candidates are spread across sentences there is no connecting verb to relate both candidates together and as a result the relationship is not discovered by the approach. This is related to the way the ASA the specification, on a sentence only basis for UML model feature discovery.

The ASA Model also demonstrated some failings associated with the verb/preposition decision matrix, which defines what the verb/preposition combination means in terms of *actions, relationships, hierarchical constructs.* This is an area that requires further research and better definition of the verb/preposition matrix design decisions.

Related works stated *"the discovery of relationships by automation would be unreliable"* **[Har00, HG02]**, but the evaluation has demonstrated that 49% of all relationships are identifiable from the requirements specification where 51% of all other relationships are being defined by candidate classes that are not actually contained within the requirements specification texts. Even though a minority of relationships were actually discovered by the ASA, 21% actually being found by the technique, the issues identified highlight the root causes associated with those failures and provide starting points for future investigations. However, the ASA's approach of utilising the main verb, semantics and syntactic constructs has led to the identification of some model relationships.

In addition a performance review was undertaken that considered the impact of discounting missing relationships not defined within the specification. This analysis gave a filtered view of performance and demonstrated an overall increase in recall performance, with precision showing no such improvement. The filtered view does not address the key issues that have been previously uncovered but does allow better understanding of the performance of the actual technique and if the missing candidate classes were available within the specification, it does not imply that the relationships would have been uncovered.

4.5.5 ASA Comparative Class Detection Performance Evaluation

The evaluation undertaken here is a comparative evaluation of the ASAs in comparison to the key related works CM-Builder **[Har00, HG02]** and NL-OOPS **[Mic96]**.

4.5.5.1 ASA vs. NL-OOPS

NL-OOPS uses a linkage threshold analysis for the detection of its class candidates. This is a count of the number of links a candidate has with other candidates contained within its model. Therefore, given a word contained within the specification the user can define a threshold that the candidate linkage must satisfy which will result in the creation of a class. This threshold is user defined and differs per specification as it is modified to obtain the best results in terms of recall and precision. Both Figure 4.5-13 and Figure 4.5-14 show the performance results in terms of recall, precision and over specification (OVS) for the various thresholds utilised by NL-OOPS, defined by *specification_name(threshold_value)*. A higher threshold returns better recall results, but in most cases results in a lower levels of precision, a typical trade off.

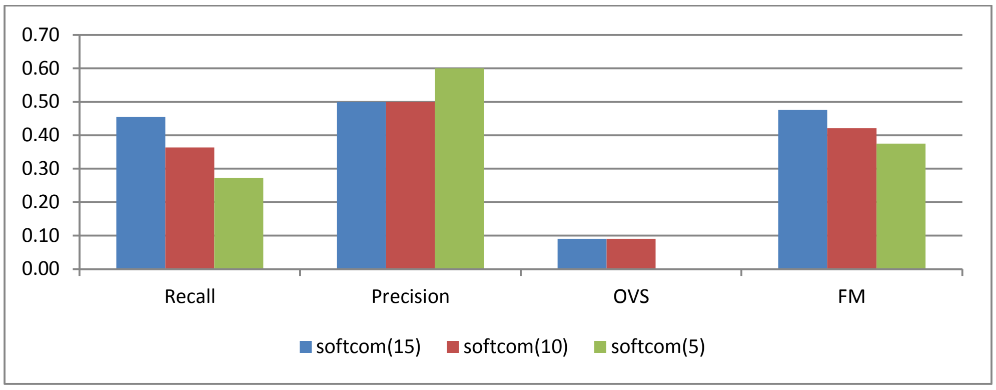


Figure 4.5-13 Requirements Specification *Softcom* Threshold Analysis [RBP+91]

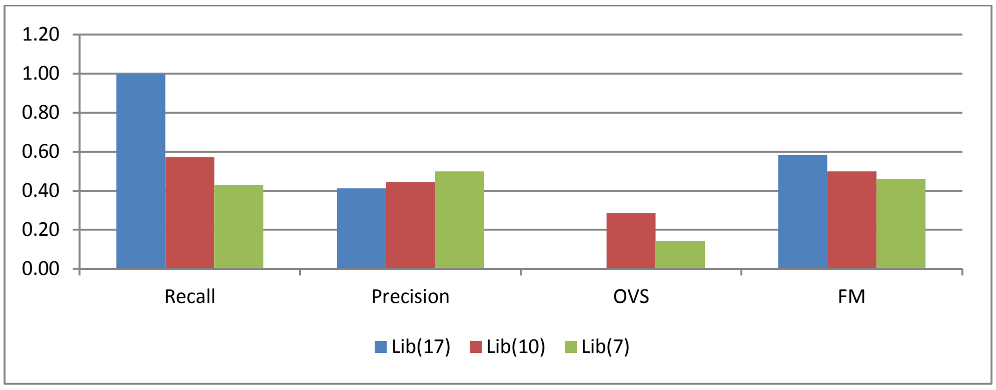


Figure 4.5-14 Requirements specification *Library* Threshold Analysis [EP98]

However, the choice of which threshold to use in the comparison between ASA and NL-OOPS is difficult as one would like to obtain a fair comparison of performance. Therefore, it has been decided that using the higher user defined threshold from each available specification and obtaining their average would give a fair comparison; even though it has an effect on precision, it is one that is negligible. The comparison itself is between three specifications. The reason the third specification is not detailed is there is only one threshold analysis given and it doesn't demonstrate the effect that different threshold decisions have on the overall performance. However, for completeness, the results for the ATM specification analysed by NL-OOPS is as follows: recall = 91%, precision = 71%, over-specification = 0%, F-Measure = 80%.

Overall the ASA performs relatively well in comparison (see Figure 4.5-15) to the NL-OOPS approach noting that the ASA approach does not utilise a user defined threshold to obtain the best results. NL-OOPS demonstrates a better average overall in terms of recall and over-specification than the ASA.

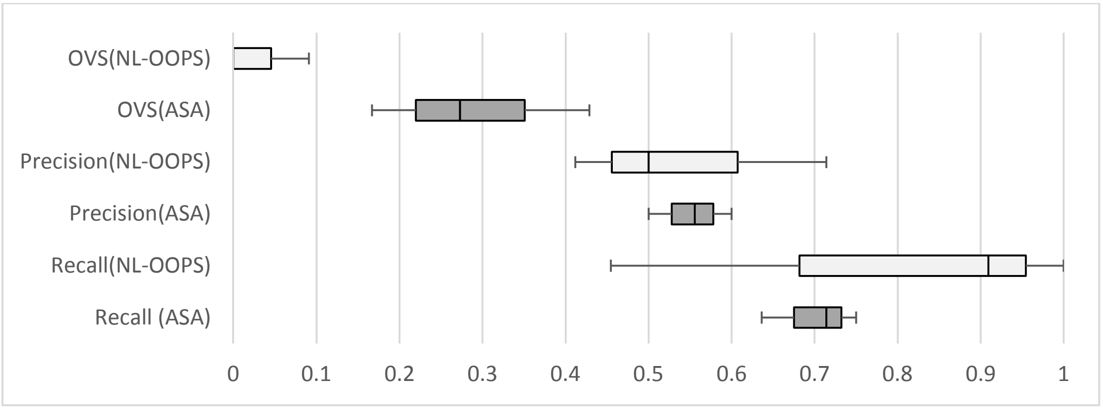


Figure 4.5-15 ASA vs. NL-OOPS Performance Results

NL-OOPS has far lower over-specification average in comparison to the ASA, which is not of any real concern since over-specification identifies candidate classes that by the authors own judgement, and using the context of the problem statement to aid that judgement, demonstrates a useful concept for consideration. However, NL-OOPS's class detection process is also user defined therefore the results can be tweaked until the desired outcome is achieved, whereas the ASA has no means of user intervention. Therefore in light of the both techniques, the ASA approach does well to achieve similar levels of recall and precision.

Table 4.5-11 details the raw data of the previous box plots, the average performance, the standard deviation (STDEV) and identifies that the NL-OOPS approach has a far greater variability in terms of all the measures used whereas ASA's recall and precision are 79% and 69% less variable respectively. This demonstrates that the ASA approach may be more consistent. Nevertheless, it is clear to see by comparing average performance that the NL-

OOPS technique does perform better than the ASA overall; but at the cost of consistency in terms of recall and precision.

Table 4.5-11 ASA and NL-OOPS Comparative Data

Measure	Minimum	Quartile 1	Median	Quartile 3	Maximum	STDEV	Average
Recall (ASA)	0.64	0.67	0.71	0.73	0.75	0.05	0.70
Recall (NL-OOPS)	0.45	0.6818	0.91	0.9545	1.00	0.24	0.79
Precision (ASA)	0.50	0.52	0.56	0.57	0.60	0.04	0.55
Precision (NL-OOPS)	0.41	0.4559	0.5	0.6071	0.71	0.13	0.54
OVS (ASA)	0.17	0.21	0.27	0.35	0.43	0.11	0.29
OVS (NL-OOPS)	0.00	0.00	0.00	0.0455	0.09	0.04	0.03

Figure 4.5-16 details the raw analysis in context of classifications for true positives and both false negatives and positives for both approaches. The approach of the ASA is not that far from being as accurate as NL-OOPS as it has fewer false positives, but it is false negatives, the candidates that the ASA does not find, that impact the ASA's approach.

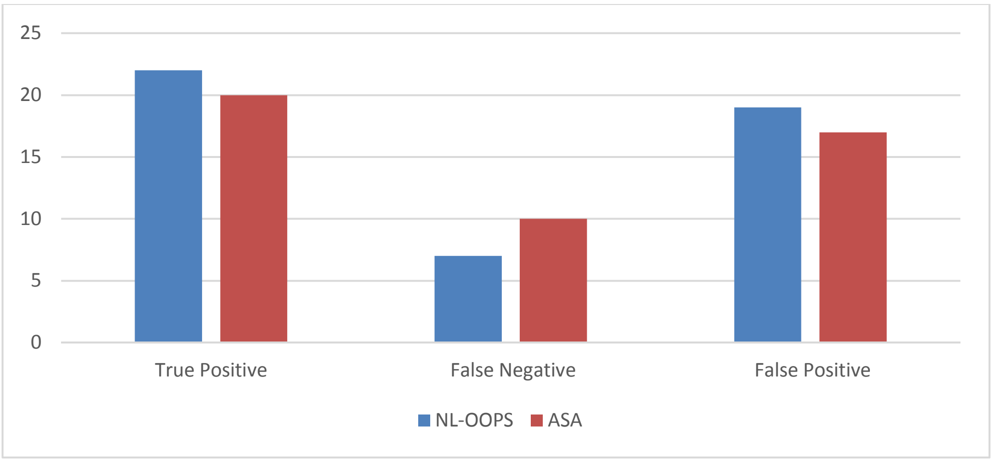


Figure 4.5-16 ASA vs. NL-OOPS Comparative Analysis

The root causes for both false positives and false negatives are the same as those previously investigated and discussed, but it is difficult to identify the root causes for NL-OOPS false positives and negatives.

4.5.5.2 ASA vs. CM-Builder

CM-Builder takes the approach of term frequency analysis for class detection, which is also a threshold that the user can modify to obtain the best results. One would assume a similar affect in relation to both recall and precision as the threshold is modified it would impact either recall and or precision. The thresholds used for each specification within the CM-Builder approach are unknown, not that this would make any difference to the overall results.

It is assumed that the thresholds used to obtain the CM-Builder results are the most efficient in terms of both recall and precision. Figure 4.5-17 details the performance results.

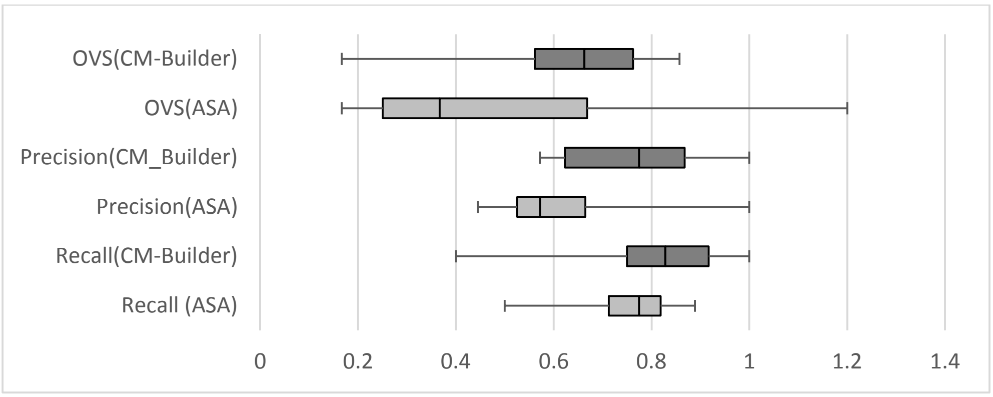


Figure 4.5-17 ASA vs. CM-Builder Performance Results

This comparison is based upon 8 specifications used by CM-Builder in their evaluation that have also been processed by the ASA. The ASA performs relatively well in comparison to recall, precision and over-specification although, in direct comparison, the CM-Builder approach does perform better overall.

Table 4.5-12 details a summary of the previous box plots, average performance and similarly standard deviation.

Table 4.5-12 ASA and CM-Builder Summary of Comparative Performance

Measure	Minimum	Quartile 1	Median	Quartile 3	Maximum	STDEV	Average
Recall (ASA)	0.50	0.7125	0.78	0.8188	0.89	0.12	0.75
Recall (CM-Builder)	0.40	0.75	0.83	0.9167	1.00	0.18	0.81
Precision (ASA)	0.44	0.525	0.57	0.6648	1.00	0.17	0.63
Precision (CM-Builder)	0.57	0.6226	0.78	0.8679	1.00	0.14	0.76
OVS (ASA)	0.17	0.25	0.37	0.6688	1.20	0.34	0.51
OVS (CM-Builder)	0.17	0.5611	0.66	0.7625	0.86	0.21	0.62

The standard deviation demonstrates that the CM-Builder approach has a greater variability in terms recall but not in precision, where the ASA is 33% less variable in terms of recall, but is 21% more variable in terms of precision and 61% in over-specification. Overall, and as with NL-OOPS, it is clear to see from the averages that the CM-Builder technique does outperform the ASA at the cost of consistency in recall only.

Figure 4.5-18 demonstrates that false negatives and false positives are the key impact affecting ASA's overall performance as it only misses 1 true positive when compared to CM-Builder.

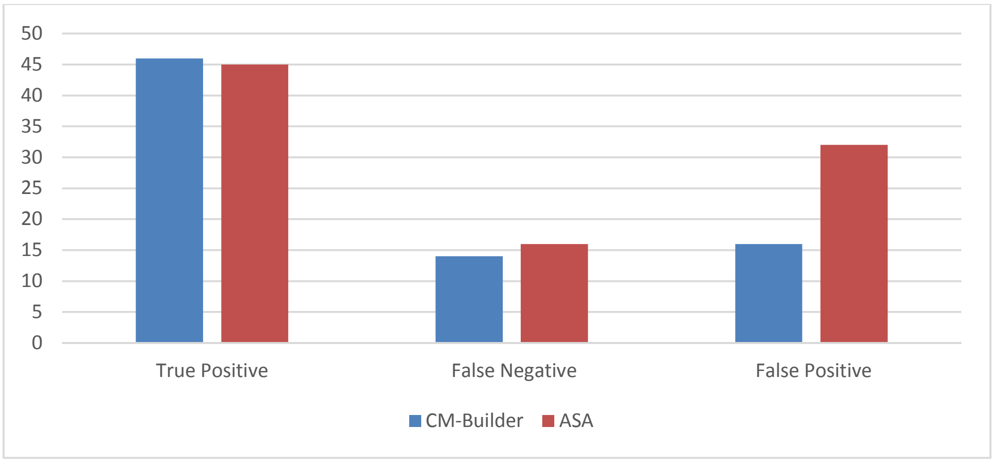


Figure 4.5-18 ASA vs. CM-Builder Comparative Analysis

The root causes for both false positives and false negatives are the same as those previously investigated and discussed, but it is difficult to identify the root causes for the CM-Builder approach. The average of over-specification is comparable to that of the CM-Builder approach, but it is still an area of concern and raises the question *Is there a difference in the author's judgement when classifying a candidate as 'Extra' in comparison to the CM-Builder?*

Table 4.5-13 CM-Builder vs. ASA Library Specification3 [Cur95] Classifications

Ideal Model	CM-Builder	Classification	ASA	Classification
Order	Order	Correct		Missing
Invoice	Invoice	Correct	Invoice	Correct
Book	Book	Correct	Book	Correct
Note	Note	Correct		Missing
Catalogue Note	Catalogue Note	Correct	Catalogue Note	Correct
Delivery Note	Delivery Note	Correct	Delivery Note	Correct
Enquiry Note		Missing		Missing
Person		Missing		Missing
	Delivery	Incorrect	Enquiry	Incorrect
	Detail	Incorrect	Instruction	Incorrect
			Store	Incorrect
			Library Desk	Incorrect
	Account Dept.	Extra	Public	Extra
	Cheque	Extra	Accounts Dept.	Extra
	File	Extra	File	Extra
	Publisher	Extra	Letter	Extra
	Someone	Extra	Publisher	Extra
		Missing	Library	Extra
		Missing	Pending File	Extra

Table 4.5-13 details the candidate classes identified for Library Problem 3 **[Cur95]** (see Appendix B.2) that results in a large over-specification value for ASA. What is of interest is the classification of *Extra* elements and the goal is to identify why there is a difference, if any, within these classifications.

The ASA detects a slightly larger number of extra candidates that need to be considered by a developer as they are not contained within the final UML model produced. Only one candidate *'Cheque'* (identified by CM-builder) is considered by ASA, but not as a class so the option is not available to be considered as an extra item. The term 'Someone' also identified by CM-Builder is not present within the specification at all and its inclusion as *extra* is a quandary. Additional elements such as *Library, Pending File, Letter* and *Public* have been identified and considered as worthy additional candidates that should be considered for inclusion within the model that have not been identified by CM-Builder. The sentences where these extra candidates have been identified are detailed as follows (in no particular order).

1. *When a **library** first receives a book from a publisher it is sent, together with the accompanying delivery note, to the library desk.*
2. *If no corresponding delivery note is found, the invoice is stored in a **pending file**.*
3. *On receipt of an invoice from the **public** the accounts department checks its store of delivery notes.*
4. *If no order can be found to match the note, a **letter** of enquiry is sent to the publishers.*

The judgement for the inclusion of extra candidates is consistent with that of CM-Builder's.

Overall the ASA's approach towards candidate class detection performs relatively well in comparison to the related work, which is encouraging. Even though related works do have better results in terms of recall and precision overall, it is assumed these results have been achieved by choosing the best user definable thresholds for frequency analysis for CM-Builder and graph linkages for NL-OOPS. The ASA has no such user-definable thresholds and relies solely on the syntactic and semantics of a given word as its decision making process for class candidates.

ASA vs CM-Builder discounting missing candidate classes

In addition to earlier discussion discounting the false negatives that are not present within the specification, it has been possible evaluate the impact during the comparative analysis as well.

Figure 4.5-19 boxplots the performance of the ASA vs CM-Builder when only considering false negatives that are present within the specification. The key difference lies with Q1 to Q3 quartile ranges are more positively skewed in comparison to the original results Figure 4.5-17.

Overall, the ASA does demonstrate better consistency over the same range of specifications, but at the cost of precision, where CM-Builder does perform more consistently overall.

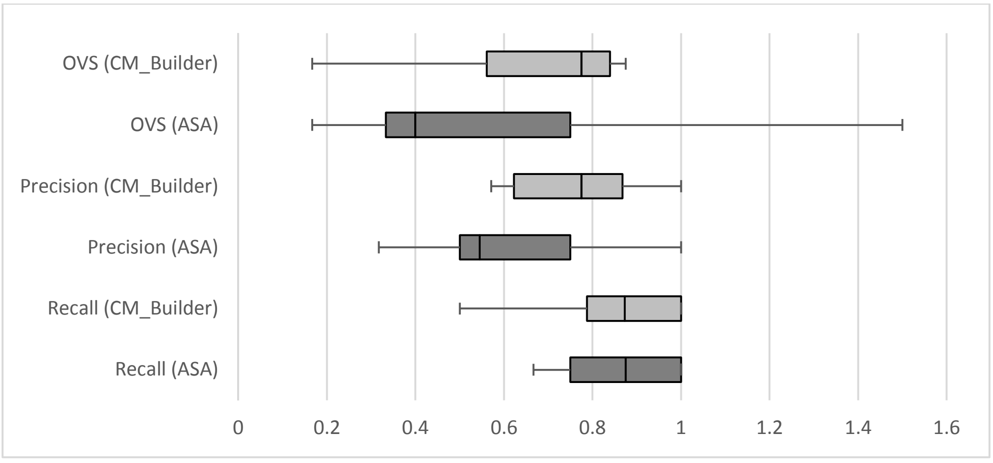


Figure 4.5-19 ASA vs CM-Builder - Discounting Missing Class Candidates – Boxplot[2]

Table 4.5-14 details the summary data of the previous box plot, average performance and standard deviation as before.

Table 4.5-14 ASA vs CM-Builder - Discounting Missing Class Candidates – Summary Data

Measure	Minimum	Quartile 1	Median	Quartile 3	Maximum	STDEV	Average
Recall (ASA)	0.67	0.75	0.88	1	1	0.14	0.86
Recall (CM-Builder)	0.50	0.78	0.87	1	1	0.16	0.85
Precision (ASA)	0.32	0.5	0.55	0.75	1	0.17	0.63
Precision (CM-Builder)	0.57	0.62	0.78	0.86	1	0.14	0.76
OVS (ASA)	0.17	0.33	0.40	0.75	1.5	0.45	0.62
OVS (CM-Builder)	0.17	0.56	0.78	0.83	0.88	0.23	0.67

This time the standard deviation demonstrates that the ASA approach is less variable in terms recall by 12% but both precision and over-specification are more variable in terms of CM-Builder by 21% and 95% respectively. After discounting false negatives not present within the specification both approaches perform relatively similarly.

4.5.5.3 Comparative Relationship Evaluation

Neither CM-builder nor NL-OOPS undertake any evaluation into the aspects of relationship detection and NL-OOPS does not make relation information available for comparison either. Furthermore, the argument exists that relationship detection is variable and open to interpretation when considering relationships discovered from the written requirements

[2] Whiskers are not visible for CM Builder data because the upper quartile is equal to the maximum.

specification. However, this could also be said of class detection as this process is also open to interpretation.

The evaluation of the CM-Builder does provide models that detail the relationships discovered during the evaluation of class detection, however this is not discussed further. Therefore the classification of correct, incorrect, missing and extra have been undertaken by the author in an unbiased manner to obtain the results detailed in Figure 4.5-20.

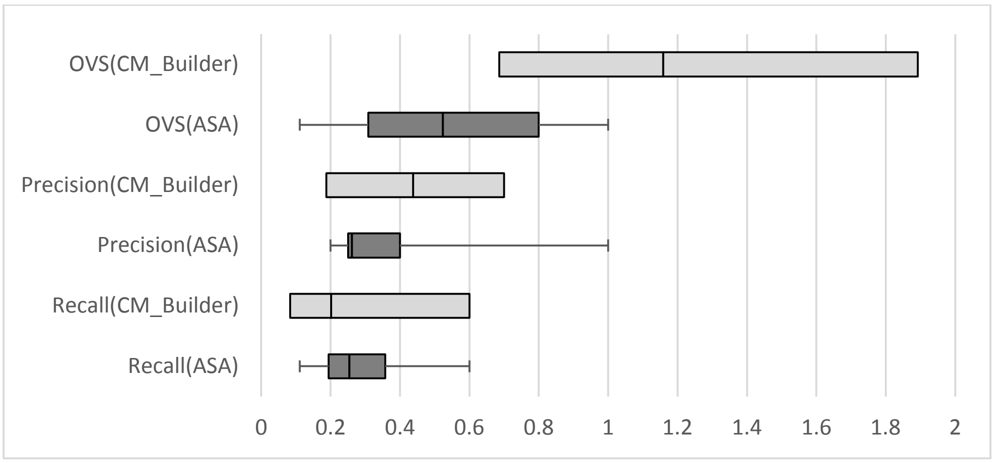


Figure 4.5-20 ASA vs. CM-Builder Relationships Performance

The ASA performs almost as well in comparison to the CM-Builder approach and the root cause analysis undertaken previously identifies the ASA issues (see section 4.5.4). One of the key issues is that the relationship information is not present within the specification.

Table 4.5-15 ASA and CM-Builder Relationship Summary Data

Measure	Minimum	Quartile 1	Median	Quartile 3	Maximum	STDEV	Average
Recall (ASA)	0.11	0.194118	0.25	0.357143	0.60	0.15	0.29
Recall (CM-Builder)	0.00	0.083333	0.20	0.6	0.75	0.28	0.31
Precision (ASA)	0.20	0.25	0.26	0.40	1.00	0.26	0.39
Precision (CM-Builder)	0.00	0.1875	0.44	0.7	1.00	0.37	0.47
OVS (ASA)	0.11	0.308824	0.52	0.8	1.00	0.30	0.54
OVS (CM-Builder)	0.40	0.686275	1.16	1.892857	2.0	0.64	1.22

Table 4.5-15 details the summary data of the previous box plots, average performance and standard deviation as before. The standard deviation demonstrates that the CM-Builder approach has a greater variability in terms recall and precision, where the ASA respectively are 46% and 29% less variable in both cases respectively, but it is clear from the average performance that the CM-Builder technique does outperform the ASA, but at the cost of consistency.

ASA vs CM-Builder discounting missing relationship candidates

Along with previous discussion related to discounting false negatives that are not present within the specification texts to obtain a more accurate view of actual performance, it has also been possible to apply the same technique to this comparative relationship evaluation. Therefore, a missing relationship that is associated to a class that is also missing from the requirements specification text are filtered from this comparative evaluation

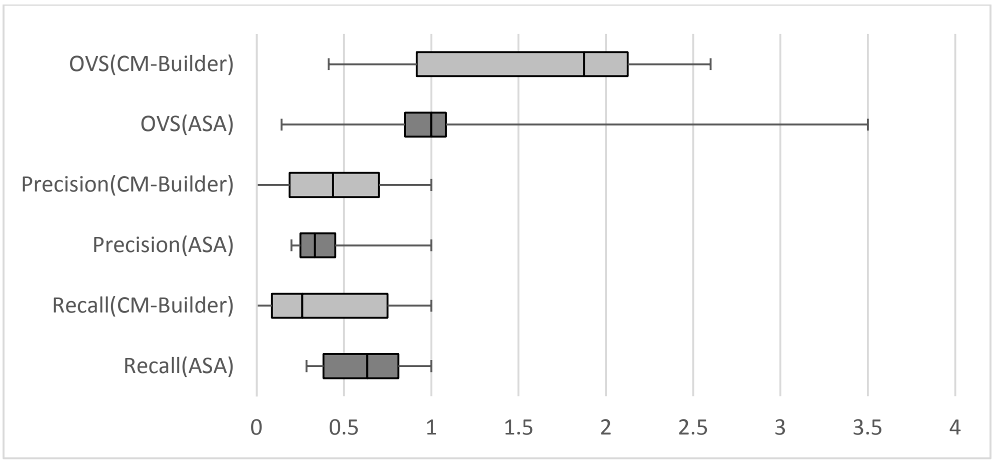


Figure 4.5-21 ASA vs CM-Builder - Discounting Missing Relationship Candidates – Boxplot

Figure 4.5-21 demonstrates that discounting/filtering relationships that are uncovered by candidate classes that are not contained within the specification texts does demonstrate an overall improvement in performance for recall and precision.

Table 4.5-16 ASA vs CM-Builder - Discounting Missing Relationship Candidate – Summary Data

Measure	Minimum	Quartile 1	Median	Quartile 3	Maximum	STDEV	Average
Recall (ASA)	0.29	0.38	0.63	0.81	1	0.26	0.63
Recall (CM-Builder)	0	0.08	0.26	0.75	1	0.37	0.39
Precision (ASA)	0.20	0.25	0.33	0.45	1	0.24	0.42
Precision (CM-Builder)	0	0.18	0.44	0.7	1	0.37	0.47
OVS (ASA)	0.14	0.85	1	1.08	3.5	0.94	1.17
OVS (CM-Builder)	0.41	0.91	1.88	2.12	2.6	0.77	1.62

Table 4.5-16 details the summary data of the previous box plots, average performance and standard deviation and demonstrates that the CM-Builder approach has a greater variability in terms recall and precision, where the ASA values are respectively 29% and 35% less variable in both cases.

It is evident that candidate relationships that are associated to class candidates that are not stated within the requirements specification texts are filtered from this comparative evaluation. Overall, the evaluation has shown an increase in recall average performance by

~161% whereas precision does not benefit in anyway and actually performs 10% worse than the current CM-Builder approach. Even though this view may give an insight into the actually performance of the approach by excluding missing candidate relationships as a result of missing class candidates, it does not demonstrate whether or not the approach would actually identify these candidates if the information where available.

4.6 Evaluation Conclusion

Overall the approach towards class detection performs relatively well, achieving an average recall of 73%; a precision average of 60%; an over-specification average of 47% and F-measure average of 64%. This identifies that techniques employed for automated software specification analysis and model generation can provide a conceptual model as a key starting point to quickly and efficiently understand the requirements of the software system.

This evaluation has investigated and identified the key areas within the ASA that are causing the main issues in terms of loss of recall, precision and the generation of false positives. The key issues identified are Semantics, Rule 27, NLP Toolkit and ASA prototype implementation issues are causing the key problems, with Semantics being the biggest issue for false positives. The remaining issues are the key issues for false negatives and missing (present within the specification). Researching and resolving these issues would most certainly aid the approach by helping to increase both recall and precision of class candidate detection.

The main issue discovered is that 20% of the information is not actually present within the specification itself, that it is discovered through some means of domain knowledge or in some other way. Further still, these insights into the design concerning missing classes may not become apparent to the developer until after the initial model has been created, but this is difficult to evaluate or even confirm at this stage. Even so, there is still a collection of information contained within the documents that has not been processed by the system for various reasons considered throughout this evaluation. The realisation and consideration of these unprocessed classes by the approach will go some way to increasing the overall recall of the system.

As with class detection, the approach also undertakes relationship detection although this performs rather poorly overall. The approach itself has a recall average of 26%, a precision average of 33%, an over-specification average of 68% and finally an F-measure average of 28%. There were still a minority of relations that the approach had not processed for one reason or another, but for the majority the relationship information was not present within the specification.

Through consideration of relationship detection and identifying the key issues concerning unprocessed relationships, it has been discovered that the approach is typically at fault within three key areas. A minority of the relationships have been not processed by the approach; that only through contextual understanding can the relationship be discovered; or the class was missing for reasons being attributed to the approach taken for class detection. The main finding is that even though the majority of relationships between design components are considered to be only discoverable through domain knowledge, a below average number of relationships are contained within the specification and it is possible for these to be extracted through consideration of these key issues.

In comparison to related works the approach does not perform as well, but over the same range of specifications used in the comparative evaluation the ASA approach towards class detection is significantly less variable in terms of recall and precision when considering its standard deviation.

Even though the most closely related works perform better in direct comparison there maybe the possibility of creating a hybrid system that considers the techniques employed in these related works such as linkage analysis to further enhance the results of the ASA in terms of both recall and precision. However, these approaches employ a user-defined variable which can be utilised to try and better the results returned whereas user involvement is a technique purposely avoided within the ASA. It is one of the key goals of ASA to be able to process an uncontrolled specification without the need to make adjustments and to create the best initial class model from the information that is readily available within the specification document and for the task of analysis to not burden the developer/analyst and allow them to work with the produced model efficiently and effectively.

The comparative evaluation between the related work of CM-Builder and the ASA was undertaken for relationship detection. Even though CM-Builder does not undertake this type of evaluation citing that detection of relationships automatically is too variable, their approach does perform marginally better than ASA and identify relationships by means of verb phrase analysis. However, as with class detection the technique employed by the ASA is less variable in terms of the recall and precision over the same range of specifications. Overall the evaluation demonstrates that it is possible to generate comparable results without any need for manual intervention during the analysis process nor manipulation of the requirements specification prior to automated analysis.

Chapter 5
Conclusions & Future Work

5.1 Conclusions

This thesis has presented and evaluated an automated approach towards UML model generation through the analysis of free-form natural language requirements specifications. This has sought to answer the primary research question *To what extent does analysis of an un-restricted natural language specification contributing to a 'better' first-cut design through means of a deep syntactic and semantic analysis?*

Study of the manual requirement analysis methodologies to understand how candidate UML model features are extracted the from natural language specifications was the initial starting point. This identified that language features such as nouns, verbs, adjectives and others are key to the identification of candidates such as classes, operations, relationships and more. Therefore, a means to extract natural language information was also a key requirement for the ASA as well.

The next key step in the process was to also recognise what had and hadn't been done before in the domain of automated requirements analysis and model generation. The assessment of the related works uncovered two differing approaches: fully and partially automated, but it also became apparent that even fully automated approaches required some means of user intervention as well. Understanding of these related works has led this thesis towards an implementation to address the key issues identified from related works, defined as follows:

1. Manual rewriting/simplification of the software requirements specification (SRS) to cater for fully-automated analysis **[NR95, LDP04, LDP05, LDP05a, PRM+07, DR08, DR09, DB09]** and semi-automated analysis **[FGR+93,BV95, BV96, BV97, Mor97, JM00, JM00a, BSC06, BCA06, BSM09, CHK07, GT07, VAD09]** techniques; this may lead to loss of key information via unconscious bias and also result in the introduction of additional information, an increase in time/cognitive effort associated with the modification of the requirements

2. User defined candidate extraction rules and domain model generation that identifies key/all candidate model features and renders automated analysis and identification irrelevant **[Bry00, LB02, LB02a, LB02b, LB02c, LB03, BLC+03, ZZ03, CHK07, IO05, IO06, OI06, Kof05, Kof05a, Kof07, Kof08, Kof09]**

3. No fundamental usage of word semantics **[Bry00, LB02, LB02a, LB02b, LB02c, LB03, BLC+03, ZZ03, PRM+07]** as a mechanism to aid identification of key model features, but only as a process to remove duplicate word features

4. No utilisation of linguistic structures as a means to identify relational/hierarchical model features

With these issue identified, the construction of a prototype implementation was undertaken. The key steps of how the ASA undertakes analysis of free-form natural language requirements specifications and automated model generation can be defined by these steps:

1. Syntactic analysis of free-form natural language requirements specifications, identifying key features such as nouns, verbs, prepositions, phrase constructs & sentence constructs (simple, compound, complex)

2. Semantic analysis of key features, identification of most common semantics and consideration of semantic sets and their implications in terms of UML model features

3. Detection of UML model features such as classes, relationships, attributes, operations, parameters and multiplicity based upon the information extracted from steps 1 and 2, which are then subsequently stored within a candidate feature graph for preservation

4. Creation of a UML model as defined by the candidate feature graph

The ASA is assisted by two key models the Syntactic Analysis Model (SAM, Step 1) and Common Semantic Model (CSM, Step 2) that demonstrate that it is possible to automatically analyse and create an initial UML model that addresses the key research findings. The idea of both SAM and CSM are not new, but what the ASA has done is to take those models to a new level of detail. SAM uses both a syntactic and a deep structural analysis technique; syntactic analysis identifies individual parts of speech such as nouns, verbs that can be processed along with the CSM to identify key model features whereas deep syntactic analysis is used to identify model features from the syntactic structure and not from semantic understanding, this is the key benefit of deep syntactic analysis. The CSM verifies candidates uncovered by the low-level analysis from SAM against the semantic sets that indicate what model feature to create based upon pre-computed semantic to UML feature mappings. It is only when both

CSM and SAM are combined is it possible to create a fully featured model from a natural language requirements specification.

Creation of UML models either automatically or manually will always be subjective, dependent on the requirements specification, users' understanding of the problem and its domain. Therefore, the techniques discussed in this thesis may never be a faultless way of automatically creating a precise software model. However, what it does achieve is the conceptualisation of the proposed software system from a free-form natural language requirements specification; not controlled or modified by any means and with less effort than it would be through manual model generation.

5.1.1 Thesis Contributions

The ASA is a domain-less approach, that can undertake deep syntactic analysis of any given sentence structure. It has the capacity to make the most relevant decision using this information that has led to the creation of a technique which is based purely upon the analysis of the syntactic and semantic features. It has also been the goal of this thesis to demonstrate the following contributions:

1. A means to automatically create a UML model from unrestricted/unmodified natural language requirements specifications.
2. Provision of an independent semantic and syntactic analysis model with no need for manual intervention, configuration or problem domain specialisation.
3. Requirement Traceability: identifying the sentence(s) within the specification where model artefacts have been identified from.
4. A reduction of overall effort associated to the manual analysis approach by means of a fully automated specification analysis and model generation process.

All of the related works (see Table 2.3-1), including the approach discussed herein make use of syntactic analysis, it is a key requirement, and when identifying the core contributions of the ASA in this context it is challenging as all approaches have the same starting point. The key differentiator and core contributions of the ASA are its semantic model which is used to make sense of all the candidate information that has been identified and extracted from the specification during syntactic analysis. The related works from the literature review do not use such a technique to make decisions which are based either on core syntactic features (nouns and verbs), thresholds, dictionaries, specification simplification and/or human intervention.

The syntactic model is utilised not just to identify key candidates for the semantic model, but is also extended to focus on the sentence constructs such as the whole verb and noun phrases ensuring that all parts of speech are used to their full extent. In addition understanding of sentence construction and their key dependencies aids discovery of relationships that are not identifiable from the main sentence verb. Overall it is the unison of both the syntactic and semantic models that gives rise to a completely user independent approach to automated software modelling from unaltered requirements specifications.

5.1.2 Research Findings

The evaluation investigated the performance of the ASA in context of the ideal model and also undertook a comparative evaluation against its most closely related works **[Har00, HG02, Mic96, MMZ02, MG02 & KZM+04]** using the metrics Recall, Precision and F-Measure. Overall the ASA, based on F-Measure and the ideal model, demonstrates an accuracy of 64% for candidate class detection and 30% for relationship detection.

Even though the ASA doesn’t perform quite as well, the techniques employed do demonstrate that its results are less variable and have greater consistency around the mean as identified by the standard deviation measures. The ASA demonstrates average recall and precision standard deviations outcomes are 58% and 23% less variable than both CM-Builder and NL-OOPS outcomes when combined. This indicates that, even with manual intervention and choosing the best result data for each of the most closely related works, the extraction and identification of model features by the ASA has greater consistency over the same range of related works’ data sets.

An impact analysis was also undertaken to understand what the impact is of class candidates that are within the ideal model but are not detailed within the requirements specification. This resulted in a total reduction on the number of false positives which in all cases improved the performance of the ASA and related approaches in terms of both recall and precision, but also impacted the standard deviation measures as well. The key impact of discarding classes and relationships not present within the requirements specification demonstrates how well the technique does perform when all information is available.

In addition to the performance related measures, there are a series of key limitations associated with the ASA in the areas of ambiguity, missing requirements, domain knowledge, intralinguistic variations, threats to validity and demonstration of effort reduction that were identified during the evaluation.

The limitations of ambiguity and intralingusitc variations can result in the creation of either relevant or irrelevant model features, primarily identified as false negatives during the evaluation. This is an intrinsic issue for both the CSM and SAM models, which gives rise to additional research areas for both of these models.

Both missing requirements and domain knowledge limitations are related, as when a requirement is missing, the user may unconsciously access their own knowledge of the domain, hence giving them the ability to infer new features not contained within the requirements specification. When considering missing requirements in the context of the ASA, it can only analyse what it is given and if requirements are not present, there is little that can be done, which is a difficult situation to resolve.

The ASA also adopts a domain-less methodology and this is a problem for explicit domain knowledge that the ASA has no awareness of. This is what gives the ASA its flexibility to operate across many differing domains, but what could be a step towards resolution is a model that feeds back and preserves changes for future use and consideration. As a result, this has been flagged as a future follow-up item.

The most concerning threat to validity is the author's bias during the evaluation. This only impacts the performance analysis of the most closely related works **[Har00, HG02, Mic96, MMZ02, MG02 & KZM+04]**, because when evaluating the ASA in context of the ideal model there has always been a key to validate against, apart from over-specification. Whereas, the performance analysis provided by the related work for NL-OOPS had varying threshold levels with varying levels of performance. The threshold and results chosen in the case of NL-OOPS to compare to the ASA have tried to show NL-OOPs at its best. This has been deliberately done to ensure no discrimination towards any related works or bias towards the ASA, as already discussed in this thesis.

Another threat to validity is the claim of *effort reduction*, which it has not been possible to validate or demonstrate in this thesis. The ASA itself maybe faster during its automated analysis and model creation rather than manual techniques. However, this does not prove a reduction in effort. What needs to be done is to validate whether the perceived benefits of the ASA do translate into an effort reduction as a result this is an item that also requires further investigation.

Positioning the ASA in context of its related works and differentiating its technique is challenging. All related works, either fully automated **[NR95, Mic96, MMZ02, MG02, KZM+04, Per02, PKS+05, Har00, HG02, ZZ03, LDP04, LDP05, LDP05a, IO05, IO06, OI06, PRM+07, DR08, DR09, DB09, SOS08, SRC+07]** or partially automated **[MHH89, FGR+93, GB94, BV95, BV96, BV97, Mor97, JM00, JM00a,AG97, AG99, AG06, GN02, SBB99, Bry00, LB02, LB02a, LB02b, LB02c, LB03, BLC+03, OLR01, Kof05, Kof05a, Kof07, Kof08, Kof09, BSC06, BCA06, BSM09, CHK07, GT07, GK08, VAD09]** - typically use some means of speech, syntactic and or semantic analysis which are then subsequently translated into some form of object/class model. The ASA also makes use of similar strategies however, it is the subtlety of the ASA's approach that separates it from the rest such as it being fully automated without any need for user intervention. A key differentiator is the ASA's enhanced syntactic analysis that allows discovery of relational/hierarchical model features without relying on parts of speech or semantic features. In addition, the usage of word semantics as a means to complement and refine the model features identified by syntactic analysis is novel and aids discovery of classes, relationships, operations, attributes, multiplicities and more. All of these subtle techniques enables the creation of a fully featured UML model that depicts and describes the key natural language requirements.

The approach presented in this thesis simplifies the analysis and design phase allowing the opportunity to build upon a robust, extensible and maintainable UML design delivered by an approach that demonstrates reliable and consist results throughout its evaluation.

5.2 Recommendations for Future Work

Future studies to follow up on the work discussed in this thesis and to address the identified limitations are set out as follows:

1. Common Semantic Model
    a. Disambiguation
    b. Additional Semantic Consideration
2. Syntactic Analysis Model - Contextual Reasoning
3. Effort Reduction Analysis

5.2.1 Common Semantic Model

The Common Semantic Model implementation is one that utilises the most commonly understood semantics as identified from WordNet **[Mil95].** Those semantic sets have been classified into collections that identify candidate UML model features. These semantics are utilised throughout the ASA to aid detection of said candidate features.

Disambiguation: would serve to ensure that the correct semantic for any given word is correctly chosen and is aimed at addressing one of the key limitations, ambiguity. It would help by ensuring that only the appropriate semantics are identified and fed into the decision making process for any given candidate feature. It is considered that through this technique an increase in both recall and precision could be seen through the reduction of false negatives and false positives identified during the evaluation. This could be achieved through creation of additional semantic models or a domain specific model managing only the semantics for that specification. However, this would also have the side effect of creating additional overheads in model creation.

Additional Semantic Considerations: Given the set of all known semantics from WordNet, only 10 out of 14 are actually used in the class detection process in the context of nouns. This leaves a remaining 4 semantic groups that might identify additional model features such as relationships, operations, state, algorithms or attributes. Consideration of these semantics and what they identify could be utilised to create a more complete UML class model.

5.2.2 Syntactic Analysis Model

The Syntactic Analysis Model operates on a sentence by sentence basis which is a key limitation as no surrounding context is considered during its analysis. Currently, it is interested in sentence structure, whether that be simple, compound or complex, at a high level or down to the individual parts of speech, phrases, and sentence components. It is the goal of SAM to extract candidate features and along with CSM it can make the relevant decision regarding a candidate feature.

Paragraph Contextual Reasoning: The premise of paragraph contextual reasoning (PCR) is that additional candidate model relationships could be discovered over and above the current techniques employed. The definition of a paragraph is a small collection of sentences typically dealing with a single theme and begins on a new line. The interesting aspect is the 'single theme' and how everything else in the paragraph relates to this theme.

Using paragraph contextual reasoning could aid identification of the relationship between the paragraph theme and the following sentence artefacts, which could uncover new model candidates and reduce the inclusion of additional/erroneous and contextual issues that resulted in a high rate of false positives identified during the evaluation.

5.2.3 Effort Reduction Analysis

The idea of effort reduction analysis (ERA) is to demonstrate that automated analysis of natural language requirements specifications does lead to a reduction of effort in the key areas analysis and modelling, which over the last 3 decades has seen the majority of project effort shift towards these areas. The key idea of ERA is to perform a study either commercially or academically to identify the benefits that the ASA can bring in the context of effort reduction, if any.

References

[Abo85] Abbot, R. (1985). Program Design by Informal English Description. *Communication of ACM*, 882-894.

[AG97] Ambriola, V., & Gervasi, G. (1997). Processing natural language requirements. *12th IEEE International Conference on Automated Software Engineering*, (pp. 36-45).

[AG99] Ambriola, V., & Gervasi, V. (1999). Experiences with domain-based parsing of natural language requirements. *Proceedings of the Fourth International Conference on Applications of Natural Language to Information Systems. 129.* OCG Schriftenreihe.

[AG06] Ambriola, V., & Gervasi, V (2006). On the Systemactic Analysis of Natrual Language Requirements with Circe. Automated Software Enineering, 107-167.

[AU06] Anandha Mala, G. S., & Uma, G. V. (2006). Object Oriented Visualization of Natural Language Requirement Specification and NFR Preference Elicitation. *IJCSNS International Journal of Computer Science and Network Security, 6*(8), 91-100.

[ANF98] Asium, C., N´Edellec, D., & Faure, D. (1998). Learning sub categorization frames and restrictions of selection. *In 10th European Conference on Machine Learning - Workshop on Text Mining.* Chemnitz.

[Bah99] Bahrami, A. (1999). *Object Oriented Systems Development.* Irwin McGraw-Hill.

[BBL10] Bajwa, I. S., Bordbar, B., & Lee, M. G. (2010). OCL Constraints Generation from Natural Language Specification. *IEEE EDOC (The Enterprise Computing) Conference*, (pp. 204-213). Vitoria, Brazil.

[BCA06] Bajwa, I. S., Choudhary, M., & Abbas, M. (2006). Natural Language Processing based Automated System for UML Diagrams Generation. *Saudi 18th National Conference on Computer Application*, (pp. 171-176).

[BSC06] Bajwa, I. S., Siddique, M. I., & Choudhary, M. A. (2006). Rule based Production Systems for Automatic Code Generation in Java. *1st International Conference on Digital Information Management*, (pp. 300-305).

[BSM09] Bajwa, S., Samad, A., & Mumtaz, S. (2009). Object Oriented Software Modeling Using NLP Based Knowledge Extraction. *European Journal of Scientific Research*, 22-33.

[Bar12] Barnes, D. J. (2012). *Objects First with Java - A Practical Introduction using BlueJ.* Prentice Hall / Pearson Education.

[Ber08] Berry, D. M. (2008). Ambiguity in Natural Language Requirements Documents (Extended abstract) Innovations for Requirement Analysis. From Stakeholders' Needs to Formal Designs. *Lecture Notes in Computer Science, 5230*, 1-7.

[BL91] Berry, D. M., & Lor, K. W. (1991, December). Automatic Synthesis of SARA Design Models from System Requirements. *IEEE Transactions on Software Engineering*, 1229-1240.

[Boo04] Booch, G. (2004). *Object-Oriented Analysis and Design with Applications, 2nd Ed.* Benjamin Cummings.

[BJR00] Booch, G., Jacobson, I., & Rumbaugh, J. (2000). *OMG Unified Modeling Language Specification, Version 1.3 First Edition.* Retrieved from http://www.omg.org/spec/UML/

[Bri94] Brill, E. (1994). Some Advances in Transformation-Based Part of Speech Tagging. *Proceedings of the twelfth national conference on Artificial intelligence*, (pp. 722 – 727).

[Bry00] Bryant, B. (2000). Object-Oriented Natural Language Requirements Specification. *23rd Australasian Computer Science Conference.*

[BLC+03] Bryant, B., Lee, B. S., Cao, F., Zhao, W., Burt, C., Gray, J., . . . Auguston, M. (2003). From Natural Language Requirements to Executable Models of Software Components. *Proc. of the Monterey Language Requirements to Executable Models* *of Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation*, (pp. 51-58).

[BV95] Burg, J. F., & Van de Reit, R. P. (1995). *Color-x: Object modelling profits from lingusitcs.* Amsterdam: Vrije University.

[BV96] Burg, J. F., & Van de Reit, R. P. (1996). Analyzing Informal Requirements Specifications: A First Step towards Conceptual Modelling. *Applications of Natural Language to Information Systems*, 15-27.

[BV97] Burg, J. F., & Van de Riet, R. P. (1997, January). The impact of linguistics on conceptual models: consistency and understandability. *Data & Knowledge Engineering, 21*(2), 131-146.

[Cal94] Callan, R. E. (1994). *Building Object-Oriented Systems: An introduction from concepts to implementation in C++.* Computational Mechanics Publications.

[Che83] Chen, P. P. (1983). English Sentence Structure and Entity-Relationship Diagrams. *Information Sciences*, 127-149.

[CHK07] Christiansen, H., Have, C. T., & Knut, T. (2007). From use cases to UML class diagrams using logic grammars and constraints. *Intl. Conf. Recent Adv. Nat. Lang. Processing.*, (pp. 128-132).

[CIS08] CIS Dept. (2008). Cinema Operation Requirement Specification Course Material. University of Strathclyde.

[Col07] Collins Dictionary. (2007). *Collins Online Dictionary*. Retrieved from http://www.collinslanguage.com/

[Cur95] Curtis, G. (1995). *Business Information systems: Analysis, Design and Practice.* Addison-Wesley Publishing Company.

[DZB+06] Daelemans, W., Zavrel, J., Berck, P., & Gillis, S. (1996). MBT: A Memory-Based Part of Speech Tagger-Generator. *In Proceedings of the 4th Workshop on Very Large Corpora*, (pp. 14-27). Copenhagen, Denmark.

[DB09] Deeptimahanti, D. K., & Babar, M. A. (2009). An Automated Tool for Generating UML Models from Natural Language Requirements. *ASE*, 680-682.

[DR08] Deeptimahanti, D. K., & Ratna, S. (2008). Static UML Model Generator from Analysis of Requirements (SUGAR). *Advanced Software Engineering and Its Applications*, 77-84.

[DR09] Deeptimahanti, D. K., & Ratna, S. (2009). An Innovative Approach for Generating Static UML Models from Natural Language Requirements. *Advances in Software Engineering Communications in Computer and Information Science*, 147-163.

[Der95] Derr, K. W. (1995). *Applying OMT. .* SIGS Books.

[Dic07] Dictionary.com. (2007). *Dictionary.com*. Retrieved from http://dictionary.reference.com/

[Duf95] Duffy, D. (1995). *From Chaos to Classess: Object-Oriented Software Developmentin C++.* McGraw-Hill Book Company.

[EFK98] El-Khouly, M., Far, B. H., & Koono, Z. (1998). *Data Dictionary Support for resuing components in automatic software Design.* The Institutue of Electronics, Information and communication Engineers.

[EP98] Eriksson, H. E., & Penker, M. (1998). *UML Toolkit.* New York: John Wiley.

[Fag89] Fagan, J. L. (1989). The effectiveness of a non-syntactic approach to automatic phrase indexing for document retrieval. *Journal of the American Society for Information Science*, 115-132.

[FGR+93] Fantechi, A., Gnesi, S., Ristori, G., Carenini, M., Vanocchi, M., & Moreschini, P. (1993). Assisting requirement formalization by means of natural language translation. *Formal Methods in System Design, 4*(3), 243-263.

[Fel98] Fellbaum, C. (1998). WordNet: An Electronic Lexical Database. *Cambridge, MA: MIT Press*.

[GK08] Gelhausen, T., & Körner, Sven. (2008). Improving Automatic Model Creation using Ontologies. *Proceedings of the Twentieth International Conference on Software Engineering & Knowledge Engineering*, (pp. 691-696).

[GT07] Gelhausen, T., & Tichy, W. F. (2007). Thematic Role Based Generation of UML Models from Real World Requirements. *International Conference on Semantic Computing ICSC*, (pp. 282-289).

[GN02] Gervasi, V., & Nuseibeh, B. (2000). Lightweight validation of natural language requirements. *Proceedings of 4th IEEE International Conference on Requirements Engineering* (pp. 140-147). Los Alamitos, CA: IEEE CS Press.

[GB94] Goldin, L., & Berry, D. M. (1994). AbstFinder, A Prototype Natural Language Text Abstraction Finder for Use in Requirements Elicitation. *Automated Software Engineering, 4*(4), 375-412.

[HG00] Harmain, H. M., & Gaizauskas, R. J. (2000). CM-builder: An automated NL-based CASE tool. *In Automated Software Engineering*, 45-54.

[Har00] Harmin, H. M. (2000). *Building Object-Oriented Conceptual Models Using Natural Language Processing Techniques.* Sheffield: University of Sheffield.

[HTK93] Homayoun Far, B., Takizawa, T., & Koono, Z. (1993, October). Software Creation: An SDL-Based Expert System for Automatic Software Design. *Procedings of the sixth SDL Forum*.

[HHV+99] Hoppenbrouwers, J., Hoppenbrouwers, S., Van den Heuvel, W., & Weighand de Troyer, H. (1999). *The grammalizer: A CASE Tool based on Textual Analysis.* Tilburg University.

[IV98] Ide, N., & Véronis, J. (1998). Introduction to the Special Issue on Word Sense Disambiguation: The State of the Art. *Computational Linguistics (COLING)*, (pp. 1-40).

[IO05] Ilieva, M. G., & Ormandjieva, O. (2005). Automatic Transition of Natural Language Software Requirements Specification into Formal Presentation. *10th Intl. Conf. on Applications of Natural Language to Information Systems.* Alicante, Spain.

[IO06] Ilieva, M. G., & Ormandjieva, O. (2006). Models Derived from Automatically Analyzed Textual User Requirements. *2006. Fourth International Conference on Software Engineering Research, Management and Applications* (pp. 13-21). Seattle, WA: IEEE.

[IH88] Ince, D. C., & Hekmatpour, S. (1998, March). An approach to automated software design based on product metrics. *Software Engineering Journal*, 53-56.

[JCJ+92] Jacobson, I., Christerson, M., Jonsson, P., & Overgaard, G. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach.* Addison-Wesley.

[Jon72] Jones, K. S. (1972). A STATISTICAL INTERPRETATION OF TERM SPECIFICITY AND ITS APPLICATION IN RETRIEVAL. *Journal of Documentation*, 11-21.

[JM97] Juzgado, N. J., & Moreno, A. (1997). Object Oriented Modelling focusing on a lingustic aproach. *22nd Annual NASA Software Engineering Workshop.*

[JM00] Juzgado, N. J., Moreno, A., & López, M. (2000). How to Use Linguistic Instruments for Object-Oriented Analysis. *IEEE Software, 17*(3).

[KK98] Karimi, J., & Konsynsky, B. R. (1998, Febuary). An automated software design assistant. *Software Engineering, IEEE Transactions*, 194-210.

[Kie09] Keis, D. (n.d.). *Grammar: Words and Their Arrangement - Form and Function of the English Clause*. Retrieved from Modern English Grammar: http://papyr.com/hypertextbooks/grammar/clause.htm

[Kie09a] Kies, D. (n.d.). *Clause and Sentence - Coordination and Subordination*. Retrieved from Modern English Grammar: http://papyr.com/hypertextbooks/grammar/complex.htm

[Kie09b] Kies, D. (n.d.). *Form and Function of Word Classes in English*. Retrieved from Modern English Grammar: http://papyr.com/hypertextbooks/grammar/word.htm

[Kie09c9] Kies, D. (n.d.). *The Phrase in English - Form and Function in the English Phrase*. Retrieved from Modern English Grammar: *http://papyr.com/hypertextbooks/grammar/phrase.htm*

[KZM+04] Kiyavitskaya, N., Zeni, N., Mich, L., & Mylopoulos, J. (2004). Experimenting with Linguistic Tools for Conceptual Modelling: Quality of the Models and Critical Features. *NLDB*.

[KM03] Klein, D., & Manning, C. D. (2003). Accurate Unlexicalized Parsing. *Proceedings of the 41st Meeting of the Association for Computational Linguistics*, (pp. 423-430).

[Kof05] Kof, L. (2005). An application of natural language processing to domain modelling - two case studies. *International Journal on Computer Systems Science Engineering*, 37-52.

[Kof05a] Kof, L. (2005). Natural language processing: Mature enough for requirements documents analysis? *10th International Conference on Applications of Natural Language to Information Systems* (pp. 91-102). Alicante, Spain,: Springer.

[Kof07] Kof, L. (2007). Scenarios: Identifying missing objects and actions by means of computational linguistics. *15th IEEE RE*, (pp. 121-130).

[Kof07a] Kof, L. (2007). Treatment of Passive Voice and Conjunctions in Use Case Documents. *12th International Conference on Applications of Natural Language to Information Systems* (pp. 181-192). Paris: Springer Berlin Heidelberg.

[Kof08] Kof, L. (2008). From Textual Scenarios to Message Sequence Charts: Inclusion of Condition Generation and Actor Extraction. *International Requirements Engineering* (pp. 331-332). Catalunya: IEEE.

[Kof09] Kof, L. (2009). Translation of Textual Specifications to Automata by Means of Discourse Context Modeling. *15th International Working Conference, REFSQ* (pp. 197-211). Amsterdam: Springer Berlin Heidelberg.

[LKK+00] Lavoie, B., Kittredge, R. I., Korelsky, T., & Rambow, O. (2000). A Framework for MT and Multilingual NLG Systems Based on Uniform Lexico-Structural Processing. *ANLP*, 60-67.

[LB02] Lee, B. S., & Bryant, B. R. (2002). Automated Conversion from Requirements Documentation to an Object-Oriented Formal Specification Language. *Proc. of ACM Symposium on Applied Computing (SAC)*, 932-936.

[LB02a] Lee, B. S., & Bryant, B. R. (2002). Automation of Software System Development Using Natural Language Processing and Two-Level Grammar. *Proc.*

2002 Monterey Workshop Radical Innovations of Software and Systems Engineering in the Future, 244-257.

[LB02b] Lee, B. S., & Bryant, B. R. (2002). Contextual Processing and DAML for Understanding Software Requirements Specifications. *19h Int. Conf Computational Linguistics*, (pp. 516-522).

[LB03] Lee, B. S., & Bryant, B. S. (2003). Applying XML Technology for Implementation of Natural Language Specifications. *Comp Syst., Sci. & Eng.*, 3-24.

[LB02c] Lee, L. S., & Bryant, B. R. (2002). Contextual Knowledge Representation for Requirements Documents in Natural Language. *15th Int. Florida AI Research Symp*, (pp. 370-374). Florida.

[LDP04] Li, K., Dewar, R. G., & Pooley, R. J. (2004). *Requirements Capture in Natural Language Problem Statemetns.* Heriot-Watt University.

[LDP05] Li, K., Dewar, R. G., & Pooley, R. J. (2005). Computer-Assisted and Customer Oriented Requirements Elicitation. *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, (pp. 479-480). Edinburgh.

[LDP05a] Li, K., Dewar, R. G., & Pooley, R. J. (2005). *Object-Oriented Analysis Using Natural Language Processing.* Herriot-Watt University.

[LG94] Long, D., & Gargliano, T. (1994). *Reasoning by Anology and Causality: Model and Applications.* Chichester, UK: Ellis Horwood.

[Mac01] Maciaszek, L. A. (2001). Requirements Analysis & System Design, Pearson Education Limited.

[FMG09] Martínez-Fernández, J. L., Martínez, P., & González-Cristóbal, J. C. (2009). Towards an Improvement of Software Development Processes through Standard Business Rules. *In Proceedings of the 2009 International Symposium on Rule Interchange and Applications* , 159-166.

[FGV+08] Martínez-Fernández, J., González, J., Villena, J., & Martínez, P. (2008). A Preliminary Approach to the Automatic Extraction of Business Rules from Unrestricted Text in the Banking Industry. *13th International Conference on*

Applications of Natural Language to Information Systems (pp. 299-310). London: Springer Berlin Heidelberg.

[MDT14] *MDT/UML2*. (2014, 09 04). Retrieved from MDT/UML2: http://wiki.eclipse.org/MDT/UML2

[MBM13] Meth, Hendrick; Brhel, Manuel; Maedche, Alexander (2013). Towards an State of the art in automated requirements elicitation. *Information and Software Technology*, 1695-1709.

[Mic96] Mich, L. (1996). NL-OOPS: from natural language to object oriented requirements using the natural langauge processing system LOLITA. *Cambridge University Press, Natural Language Engineering*, pp. 161-187.

[MG02] Mich, L., & Garigliano, R. (2002). NL-OOPS: A Requirements Analysis tool based on Natural Language Processing. *In Proc. 3rd Int. Conf. On Data Mining*, (pp. 321-330). Bologna.

[MFN04] Mich, L., Franch, M., & Novi Inverardi, P. L. (2004). Market Research for Requirements Analysis Using Linguistic Tools. *Requirements Eng*, 40-56.

[MMZ02] Mich, L., Mylopoulos, J., & Zeni, N. (2002). *Improving the Quality of Conceptual Models with NLP Tools: An Experiment.* Trento: Department of Information and Communication Technologies, University of Trento.

[Mil95] Miller, G. A. (1995). WordNet: A Lexical Database for English. *Communications of the ACM*, 39-41.

[MIT05] MIT. (2005). *Gizmoball 6-170-laboratory-in-software-engineering*. Retrieved from 6-170-laboratory-in-software-engineering: http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-170-laboratory-in-software-engineering-fall-2005/projects/6_170_gizmoball.pdf

[Mor97] Moreno, A. M. (1997). Object-oriented analysis from textual specifications. *9 th International Conference on Software Engineering and Knowledge Engineering .*

[Mor07] Morton, T. (2007). *OpenNLP*. Retrieved from OpenNLP: http://opennlp.apache.org/

[MHH89] Motoshi, S., Hisayuki, H., & Hajime, E. (1989). Software development process from natural language specification. *11th international conference on Software engineering (ICSE '89)* (pp. 64-73). New York, NY, USA: ACM. doi:10.1145/74587.74594

[MS87] Murray, K. J., & Sheppard., S. V. (1987). Automatic model synthesis: using automatic programming and expert systems techniques toward simulation modeling. *19th conference on Winter simulation (WSC '87)* (pp. 534-543). New York, USA: ACM.

[NR95] Nanduri, S., & Rugaber, S. (1995). Requirements validation via automated natural language parsing. *International Conference on System Sciences*, *3*, pp. 362-368.

[NIST02] NIST. (2002). *The Economic Impacts of Inadequate Infrastructure for Software Testing.* National Institute of Standards & Technology. Retrieved from http://www.nist.gov/director/prog-ofc/report02-3.pdf

[OMH04] Omar, N., Mc Kevitt, P., & Hanna, P. (2004). Heuristics-based entity-relationship modelling through natural language processing. *Fifteenth Irish Conference on Articial Intelligence and Cognitive Science*, (pp. 302-313).

[OI06] Ormandjieva, O., & Ilieva, M. (2006). Automatic Comprehension of Textual User Requirements and their Static and Dynamic Modeling. *Software Engineering Research and Practice*, 266-273.

[OLR01] Overmyer, S. P., Lavoie, B., & Rambow, O. (2001). Conceptual Modeling through Linguistic Analysis Using LIDA. *ICSE*, 401-410.

[Oxf07] Oxford Dictionary. (2007). *Oxford Dictionary*. Retrieved from http://oxforddictionaries.com/

[Par72] Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communication ACM*, 1053-1058. doi:DOI=10.1145/361598.361623

[Per02] Perez-Gonzalez, H. G. (2002). Automatically Generating Object Models from Natural language Analysis. *Companion of the 17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, (pp. 86-87).

[PRM+97] Popescu, D., Rugaber, S., Medvidovic, N., & Berry, D. M. (1997). *Quality improvement of requirements specification via automatically created object oriented models*. Retrieved from Quality improvement of requirements specification via automatically created object oriented models: http://www.cc.gatech.edu/projects/dowser/example_usage.html

[PRM+07] Popescu, D., Rugaber, S., Medvidovic, N., & Berry, D. M. (2007). *Improving the quality of requirements specifications via automatically created object-oriented models.* California: University of Southern California.

[PRR80] Porter, M.F., Robertson, S.E., van Rijsbergen, C.J. (1980). *New models in probabilistic information retrieval.* British Library Research and Development Report, no. 5587.

[PKS+05] Prez-Gonzlez, H. G., Kalita, J. K., Salvador Nez Varela, A., & Wiener, R. S. (2005). GOOAL: an educational object oriented analysis laboratory. *In Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (pp. 180-181). New York, NY, USA: ACM.

[QGL+] Quirk, R., Greenbaum, S., Leech, G., & Svartvik, J. (1985). *A Comprehensive Grammar of the English Language.* Longman Group Limited.

[Rat98] Ratnaparkhi, A. (1998). *Maximum Entropy Models for Natural Language Ambiguity Resolution.* Pennsylvania: University of Pennsylvania. Retrieved from http://repository.upenn.edu/cgi/viewcontent.cgi?article=1061&context=ircs_repor ts

[RW91] Reubenstein, H. B., & Waters, R. C. (1991, March). The Requirements Apprentice: automated assistance for requirements acquisition. *IEEE Transactions on Software Engineering*, 226-240.

[RSB93] Richardson, J. E., Schultz, R. C., & Berard, E. V. (1993). *A Complete Object-Oriented Design Example.* Berard Software Engineering.

[Ric99] Richter, C. (1999). *Designing Flexible Object-Oriented Systems with UML.* Macmillan Technical Publishing.

[RP92] Rolland, C., & Proix, C. (1992). A natural language approach for Requirements Engineering. *Lecture Notes in Computer Science, Advanced Information Systems* , 257-277.

[RKP08] Rong, P., Keqing, H., & Peng, L. (2008). Automatic System Modeling Approach Based on Semantic Association. *IEEE International Workshop on Semantic Computing and Systems* (pp. 82-88). Washington, DC, USA: IEEE Computer Society.

[Ros99] Rosenberg, D. (1999). *Use Case Driven Object Modeling with UML: A Practical Approach.* Addison Wesley.

[RBP+91] Rumbaugh, S., Blaha, M., Premerlai, W., Eddy, F., & Lorensen, W. (1991). *Object-oriented Modeling and design.* New Jersey: Prentice-Hall.

[SOS08] Sabra, Samer; Ormandjieva, Olga; Seresht, Shadi Moradi (2008). Automatic Conceptual Analysis of User Requirements with the Requirements Engineering Assistance Diagnostic (READ) Tool. *Proceedings of the 2008 Sixth International Conference on Software Engineering Research, Management and Applications* (pg. 133-142).

[SRC+07] Sampaio, Américo; Rashid, Awais; Chitchyan, Ruzanna; Rayson, Paul (2007) EA-Miner: Towards Automation in Aspect-Oriented Requirements Engineering. *Transactions on Aspect-Oriented Software Development III* (pg 4-39).

[SRG02] Sawyer, P., Rayson, P., & Garside, R. (343-353). REVERE: Support for Requirements Synthesis from Documents. *Information Systems Frontiers (ISF)*, 2002.

[Son91] Soni, M. (2009, June). *Defect Prevention: Reducing Costs and Enhancing Quality*. Retrieved from isixsigma: http://www.isixsigma.com/index.php?option=com_k2&view=item&id=520&Itemid =1&Itemid=1

[SA97] Srikant, R., & Agrawal, R. (1997). Mining generalized association rules. *Future Generation Computer Systems*, (pp. 161-180).

[SP99] Stevens, P., & Pooley, R. (1999). *Using UML: Software Engineering with Objects and Components.* Addison Wesley.

[SBB99] Sylvain, D., Barker, K., & Biskri, I. (1999). Object-Oriented Analysis: Getting Help from Robust Computational Linguistic Tools. *The 4th International Conference on Applications of Natural Language to Information Systems* (pp. 167-171). Klagenfurt, Austria: OCG Schriftenreihe 129.

[VFS06] Videira, C., Ferreira, D., & Silva, A. (2006). A Linguistic Patterns Approach for Requirements Specification. *Proceedings of the 32nd EUROMICRO and Advanced Applications*, (pp. 302-309).

[VAD09] Vinay, S., Aithal, S., & Desai, P. (2009). An Approach towards Automation of Requirements Analysis. *International MultiConference of Engineers & Computer Scientists*, (p. 1080).

[Wik07] Wikitionary. (2007). *Wikitionary*. Retrieved from http://en.wiktionary.org/wiki/Wiktionary:Main_Page

[WW89] Wirfs-Brock, R., & Wilkerson, B. (1989). Oject-Oriented Design: A Responsibility-Driven Approach. *OOPSLA'89 Proceedings*, (pp. 71-75).

[WWW90] Wirfs-Brock, R., Wilkerson, R., Wilkerson, B., & Wiener, L. (1990). *Designing Object-Oriented Software.* Prentice Hall.

[ZZ04] Zhou, N., & Zhou, X. (2004). *Auto-generation of Class Diagram from Free-text Functional Specifications and Domain Ontology.* doi:10.1.1.99.2062

Summary of Rules

- Rule 1 – Class Detection
  - *If a noun's most common semantic belongs to the set of candidate class semantics, then that noun is a candidate class*
- Rule 2 – Class Detection from Non-Class Semantics
  - *If a noun's most common semantic belongs to the set of non-candidate class semantics, and that noun also contains an artefact semantic, then noun is a candidate class*
- Rule 3 - Start Range Multiplicity Detection (Determiners Present)
  - *If a determiner belongs to the set of multiplicity mappings {0, 1, *}, then the start range for multiplicity has been found*
- Rule 4 – Start Range Multiplicity Detection (Missing Determiners)
  - *If the determiner does not exist, then the start range is known as single (1)*
- Rule 5 - End Range Multiplicity Detection Rule (Plural Nouns)
  - *If a noun is a candidate class, as defined by Rule 1 or Rule 2, and the noun is plural, its mapping is known as many (*)*
- Rule 6 – End Range Multiplicity Detection (Non-Plural Nouns)
  - *If the noun is a candidate class, as defined by Rule 1 or Rule 2, and the noun is not plural, then its mapping is known as single (1)*
- Rule 7 – Class Hierarchy Detection Rule
  - *Given the noun phrase, if the head noun is a candidate class as defined by Rule 1 or Rule 2 and the head noun's pre-modifier is also candidate class as defined by Rule 1 or Rule 2, then an interface and abstraction is extracted based on the head noun*

- Rule 8 – Object State
    - *Given the Noun Phrase all determiners (DT) are ignored and the participle modifiers are anything else that precedes the Head Noun that must be within the set of verbs defined as (VBN (past participle), VBG (present participle)). The Noun Head is the last noun (NN) contained within the noun phrase*
- Rule 9 – Noun Preposition Attachment Detection
    - *If the prepositional phrase's parent is a Noun Phrase, then the preposition is said to be attached to that the noun phrase.*
- Rule 10 – Verb Preposition Attachment Detection
    - *If the prepositional phrase's parent is a Verb Phrase, then the preposition is said to be attached to that verb phrase.*
- Rule 11 – Operation Detection
    - *If a verb's most common semantic belongs to the set of candidate operation semantics and the verb's semantic does not belong to the set of candidate relationship semantics, then that verb is a candidate operation*
- Rule 12 - Relationship Detection
    - *If a verb's most common semantic belongs to the set of candidate relationship semantics then that verb is a candidate relationship*
- Rule 13 – Subject Operation Placement
    - *If the verb is in an active form and as defined by Rule 9 is an operation and by Rule 1 or Rule 2 the sentence subject is a class candidate, then the operation will be placed with the subject of the sentence*
- Rule 14 – Object Operation Placement
    - *If the verb is in its passive form and as defined by Rule 9 is an operation and by Rule 1 or Rule 2 the sentence object is a candidate class, then the operation will be placed with the object of the sentence*

- Rule 15 – Active Voice Parameter Creation
  - *If the sentence is in active voice and by Rule 9 an operation exists and by Rule 1 or Rule 2 a class candidate exists for both sentence subjects and objects and by Rule 12 the operation is placed with the subject of the sentence, then the object of sentence is considered as a parameter of that operation*
- Rule 16 – Passive Voice Parameter Creation
  - *If the sentence is in passive voice and by Rule 9 an operation exists and by Rule 1 or Rule 2 a class candidate exists for both sentence subjects and objects and by Rule 13 the operation is placed with the object of the sentence, then the subject of sentence is considered as a parameter of that operation*
- Rule 17 – Verb Derived Attribute Detection
  - *If the verb of the sentence belongs to the set of verb forms {has, had, have} and the noun following the verb is a class candidate as defined by Rule 1 or Rule 2, then that class is transformed into an attribute*
- Rule 18 – Dynamic Verb derived Attribute Detection
  - *If the sentence contains a noun that is class candidate and is preceded by a verb, where the frequency count of that noun is less than the average noun frequency for the document and there exists only one verb within the sentence and the semantics for that verb belongs to the set of {has, had, have} forms, then the noun is said to be an attribute*
- Rule 19 – Class & Relationship Detection
  - *If both subject and objects of the sentence are class candidates as defined by Rule 1 or Rule 2 and the semantics of the subject and objects are not contained within the set of attribute semantics and the term frequencies of both subjects and objects are greater than the term frequencies for the document and the verb belongs to the set of {has, had, have} forms and there is only one verb, then it is said that both subjects and objects are class candidates and an association relationship exists between class candidates*

- Rule 20 – Inheritance Hierarchy Detection
    - *If the verb belongs to the set of 'be' forms and that verb is the only verb in the sentence and both the subject and object of the sentence are candidate classes as defined by Rule 1 or Rule 2, then an inheritance hierarchy is said to exist between both class candidates*
- Rule 21 – Matrix Relationship Detection
    - *If the noun is a class candidate as defined by Rule 1 or Rule 2 and the intersection of both the verb's semantic and the semantic of the preposition belongs to the set of relationship semantics, then a relationship is said to exist between the noun and the object of the preposition*
- Rule 22 - Matrix Parameter Detection
    - *If the noun is a class candidate as defined by Rule 1 or Rule 2 and the verb is an operation as defined by Rule 11 and the intersection of both the verb's semantic and the semantic of the preposition also belong to the set of parameter semantics, then the object of the preposition is said to be the parameter of the operation*
- Rule 23 - Matrix Class Hierarchy Detection
    - *If the noun is a class candidate as defined by Rule 1 or Rule 2 and the object of preposition is also a class candidate as defined by Rule 1 or Rule 2 and the intersection of both the verb's semantic and the semantic of the preposition also belong to the set of class hierarchical semantics, then it is said there exists a class hierarchical relationship between the class candidate and the object of the preposition*
- Rule 24 - Matrix Attribute Detection
    - *If the noun is a class candidate as defined by Rule 1 or Rule 2 and the intersection of both the verb's semantic and the semantic of the preposition also belong to the set of attribute semantics, then the object of the preposition is said to be an attribute of the class candidate*

- Rule 25 - Clausal Relationship Detection
    - *If an independent clause exists and by Rule 1 or Rule 2 a class candidate exists, and if a dependent clause exists and by Rule 1 or Rule 2 a class candidate exists, then it is said an association shall also exist between both independent and dependent clause class candidates*
- Rule 26 - Attribute Detection based on Semantics
    - *If a noun's semantic belongs to the set of attribute semantics, then that noun is considered as an attribute*
- Rule 27 - Attribute Detection based on Semantics, Class Candidates & Term Frequencies
    - *If a class candidate exists as defined by Rule 1 or Rule 2, and within that noun's semantic set it also belongs to the set of attribute semantics, and the frequency count of that noun is less than the average noun frequency count for class candidates for that document, then the noun is said to be an attribute.*
- Rule 28 - Semantic Class Hierarchical Detection
    - *If a class candidate exits as defined by Rule 1 or Rule 2 and the semantics of the class candidate are also contained within the set of candidate class hierarchical semantics, then an interface will also be extracted for that class candidate*

Appendix A.1 Noun/Verb Semantic Classification Tables

	WordNet (noun) Semantic	WordNet Description	Class Modelling Implication		
			Class	Hierarchy	Can Imply
1	Animal	Nouns denoting animals	X	X	
2	Artefact	Nouns denoting man-made objects	X		
3	Body	Nouns denoting body parts	X		
4	Communication	Nouns denoting communicative processes and contents	X		
5	Food	Nouns denoting foods and drinks	X		
6	Group	Nouns denoting groupings of people or objects	X		
7	Location	Nouns denoting spatial position	X		
8	Object	Nouns denoting natural objects (not man-made)	X		
9	Person	Nouns denoting people	X	X	
10	Phenomenon	Nouns denoting natural phenomenon	X		
11	Plant	Nouns denoting plants	X	X	
12	Shape	Nouns denoting two and three dimensional shapes	X	X	
13	Substance	Nouns denoting substances	X		
14	Time	Nouns denoting time and temporal relations	X		
15	Act	Nouns denoting acts or actions			Operation
16	Possession	Nouns denoting possessions and transfer of possessions			Relationship
17	Quantity	Nouns denoting quantities and units of measure			Multiplicity
18	State	Nouns denoting stable states of affairs			Object State
19	Process	Nouns denoting natural processes			Algorithm
20	Motive	Nouns denoting goals			Algorithm
21	Relation	Nouns denoting relations between people, things or ideas			Relationship
22	Attribute	Nouns denoting attributes of people and objects			Class Attribute
23	Event	Nouns denoting natural events			Algorithm/Operation
24	Cognition	Nouns denoting cognitive processes and contents			Algorithm
25	Feeling	Nouns denoting feelings and emotions			Unknown

ID	Involved Entities	WordNet (verb) Semantic	WordNet Description	Relationship/Operation Modelling Implications	
				Relationship	Operation
1	1	Body	Verbs of grooming, dressing and bodily care		X
2	1	Change	Verbs of size, temperature change, intensifying, etc		X
3	2	Cognition	Verbs of thinking, judging, analysis, doubting, etc	X	X
4	2	Communication	Verbs of telling, asking, ordering, singing	X	X
5	2	Competition	Verbs of fighting, athletic activities	X	X
6	1	Consumption	Verbs of eating and drinking		X
7	2	Contact	Verbs of touching, hitting, tying, digging	X	X
8	1	Creation	Verbs of sewing, baking, painting, performing		X
9	1	Emotion	Verbs of feeling		X
10	2	Motion	Verbs of walking, flying, swimming	X	X
11	1	Perception	Verbs of seeing, hearing, feeling		X
12	2	Possession	Verbs of buying, selling, owning	X	X
13	2	Social	Verbs of political and social activities and events	X	X
14	1	Stative	Verbs of being, having, spatial relations		X
15	1	Weather	Verbs of snowing, raining, thawing, thundering		X

Appendix A.2 NLP Tag List

Tag	Description
Clause Level	
S	simple declarative clause
SBAR	Clause introduced by a (possibly empty) subordinating conjunction.
SBARQ	Direct question introduced by a wh-word or a wh-phrase.
SINV	Inverted declarative sentence,
SQ	Inverted yes/no question, or main clause of a wh-question, following the wh-phrase in SBARQ
Phrases Level	
ADJP	Adjective Phrase
ADVP	Adverb Phrase.
CONJP	Conjunction Phrase.
FRAG	Fragment.
INTJ	Interjection.
LST	List marker
NAC	Not a Constituent
NP	Noun Phrase.
NX	Used within certain complex NPs to mark the head of the NP
PP	Prepositional Phrase.
PRN	Parenthetical.
PRT	Particle
QP	Quantifier Phrase
RRC	Reduced Relative Clause.
UCP	Unlike Coordinated Phrase.
VP	Verb Phrase.
WHADJP	Wh-adjective Phrase.
WHAVP	Wh-adverb Phrase.
WHNP	Wh-noun Phrase.
WHPP	Wh-prepositional Phrase.
X	Unkown
Word Level	
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Pre-determiner
POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	To
UN	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

Appendix A.3 Preposition Semantics

with	logical.accomp
for	logical.action
lest	logical.action.false
plus	logical.add
than	logical.comparison
albeit	logical.condition
although	logical.condition
because	logical.condition
but	logical.condition
if	logical.condition
pending	logical.condition
per	logical.condition
providing	logical.condition
save	logical.condition
that	logical.condition
though	logical.condition
unless	logical.condition
vs.	logical.condition
whereas	logical.condition
whether	logical.condition
while	logical.condition
despite	logical.condition.negation
par	logical.equality
so	logical.event
against	logical.false
except	logical.false
neither	logical.false
notwithstanding	logical.false
without	logical.false
into	logical.goal
throughout	logical.inclusion
via	logical.precondition
virtually	logical.probability
about	logical.qty
minus	logical.remove
as	logical.role
like	logical.similarity
unlike	logical.similarity.negation
once	logical.singleton
worth	logical.value
within	spatial.contains
aboard	spatial.contians
above	spatial.relation
across	spatial.relation
along	spatial.relation
alongside	spatial.relation
amid	spatial.relation
among	spatial.relation
around	spatial.relation
at	spatial.relation
atop	spatial.relation
behind	spatial.relation
below	spatial.relation
beneath	spatial.relation
beside	spatial.relation
besides	spatial.relation
between	spatial.relation
beyond	spatial.relation
by	spatial.relation
down	spatial.relation
in	spatial.relation
near	spatial.relation
nearer	spatial.relation
nearest	spatial.relation
of	spatial.relation
off	spatial.relation
on	spatial.relation
opposite	spatial.relation
out	spatial.relation
outside	spatial.relation
over	spatial.relation
past	spatial.relation
round	spatial.relation
through	spatial.relation
toward	spatial.relation
under	spatial.relation
underneath	spatial.relation
up	spatial.relation
upon	spatial.relation
inside	spatiral.relation
next	spatiral.relation
onto	spatiral.relation
till	temporal.duration
until	temporal.duration
bout	temporal.event
from	temporal.event
post	temporal.event
since	temporal.event
then	temporal.event
after	temporal.future
ago	temporal.past
during	temproal.event
before	temproal.past

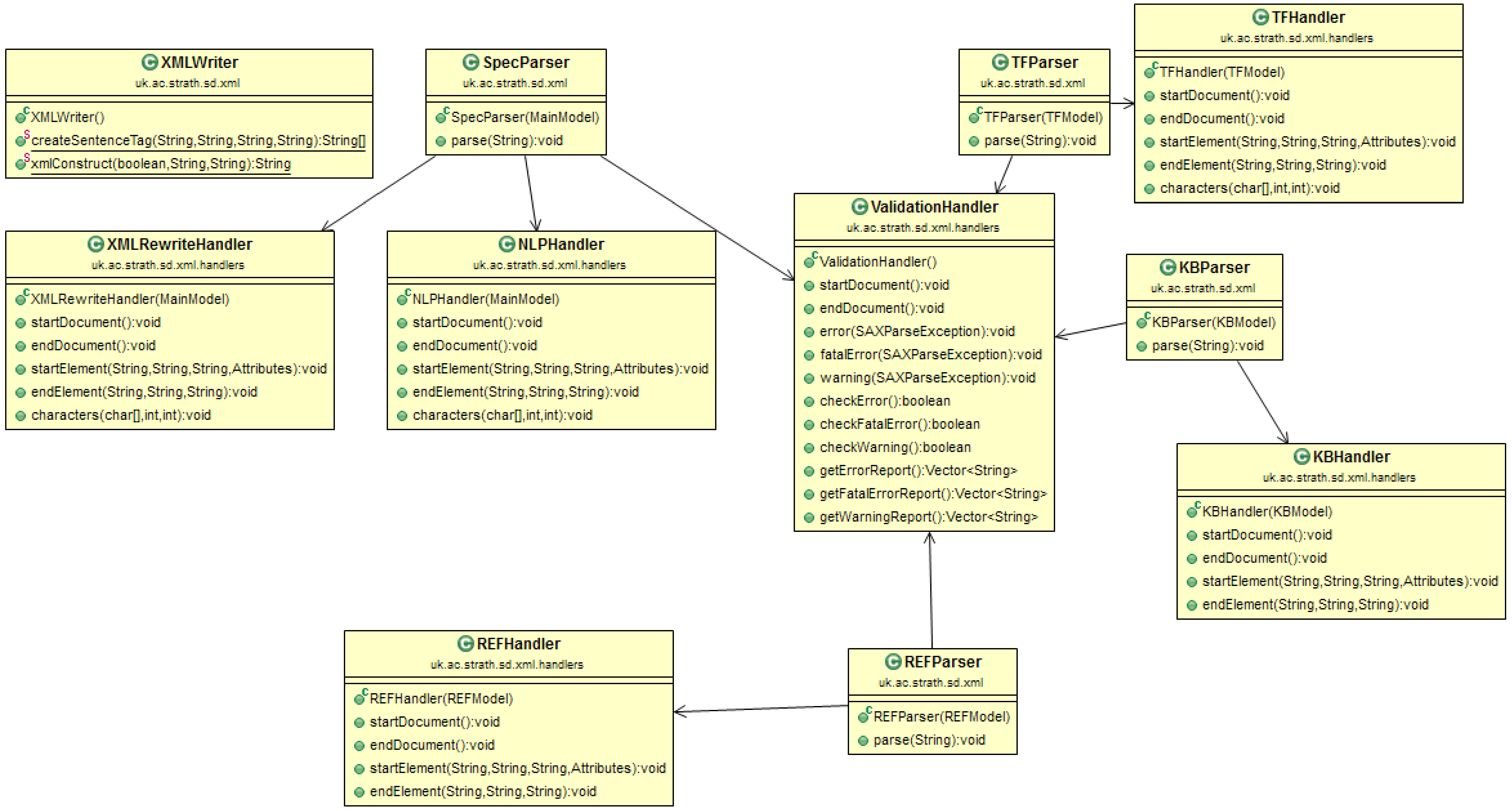
Appendix A.4 Verb Preposition Decision Matrix

	body	change	cognition	communication	competition	consumption	contact	creation	emotion	motion	perception	possession	social	stative
logical.accomp	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel
logical.action	act	act	act	act	act	act	act	act	act	act	act	act	act	act
logical.action.false	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel
logical.add	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel
logical.comparison	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel
logical.condition	rel	act	rel	act	act	rel	act	act	rel	act	rel	act	rel	rel
logical.condition.negation	rel	act	rel	act	act	rel	act	act	rel	act	rel	act	rel	rel
logical.equality	rel	act	rel	act	act	rel	act	act	rel	act	rel	act	rel	rel
logical.event	rel	act	rel	act	act	rel	act	act	rel	act	rel	act	rel	rel
logical.false	rel	act	rel	act	act	rel	act	act	rel	act	rel	act	rel	rel
logical.goal	rel	rel	rel	act	rel	act	act	act	rel	act	rel	act	rel	rel
logical.inclusion	rel	act	rel	act	act	rel	act	act	rel	act	rel	act	rel	rel
logical.precondition	rel	act	rel	act	act	rel	act	act	rel	act	rel	act	rel	rel
logical.probability	rel	act	rel	act	act	rel	act	act	rel	act	rel	act	rel	rel
logical.qty	rel	act	rel	act	act	rel	act	act	rel	act	rel	act	rel	rel
logical.remove	rel	act	rel	act	act	rel	act	act	rel	act	rel	act	rel	rel
logical.role	rel	rel	type_of	act	act	rel	act	act	rel	act	rel	rel	rel	rel
logical.similarity	rel	act	rel	act	act	rel	act	act	rel	act	rel	act	rel	rel
logical.similarity.negation	rel	act	rel	act	act	rel	act	act	rel	act	rel	act	rel	rel
logical.singleton	rel	act	rel	act	act	rel	act	act	rel	act	rel	act	rel	rel
logical.value	rel	act	rel	act	act	rel	act	act	rel	act	rel	act	rel	rel
logical.view	rel	rel	attr	act	act	rel	rel	act	rel	act	rel	act	rel	attr
spatial.contains	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel
spatial.relation	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel	rel
temporal.duration	rel	act	rel	act	act	rel	act	act	rel	act	rel	act	rel	rel
temporal.event	act	act	act	act	act	act	act	act	act	act	act	act	act	act
temporal.future	rel	act	rel	act	act	rel	act	act	rel	act	rel	act	rel	rel
temporal.past	rel	act	rel	act	act	rel	act	act	rel	act	rel	act	rel	rel
to	rel	act	rel	act	act	rel	act	act	rel	act	rel	act	rel	rel

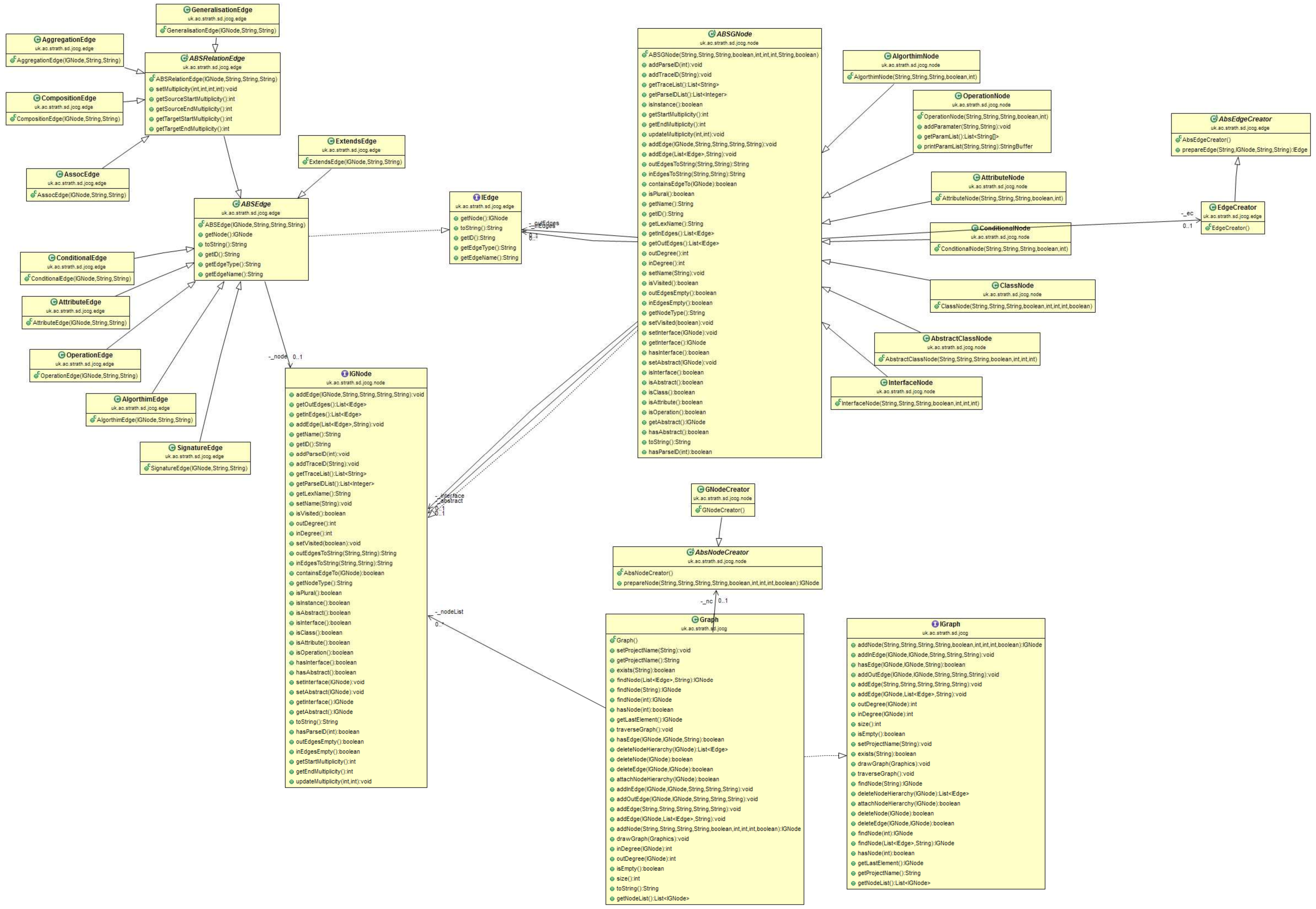
```
Key:
rel – relationship
act – action
attr – attribute
```

Appendix A.5 ASA Package Level UML Models

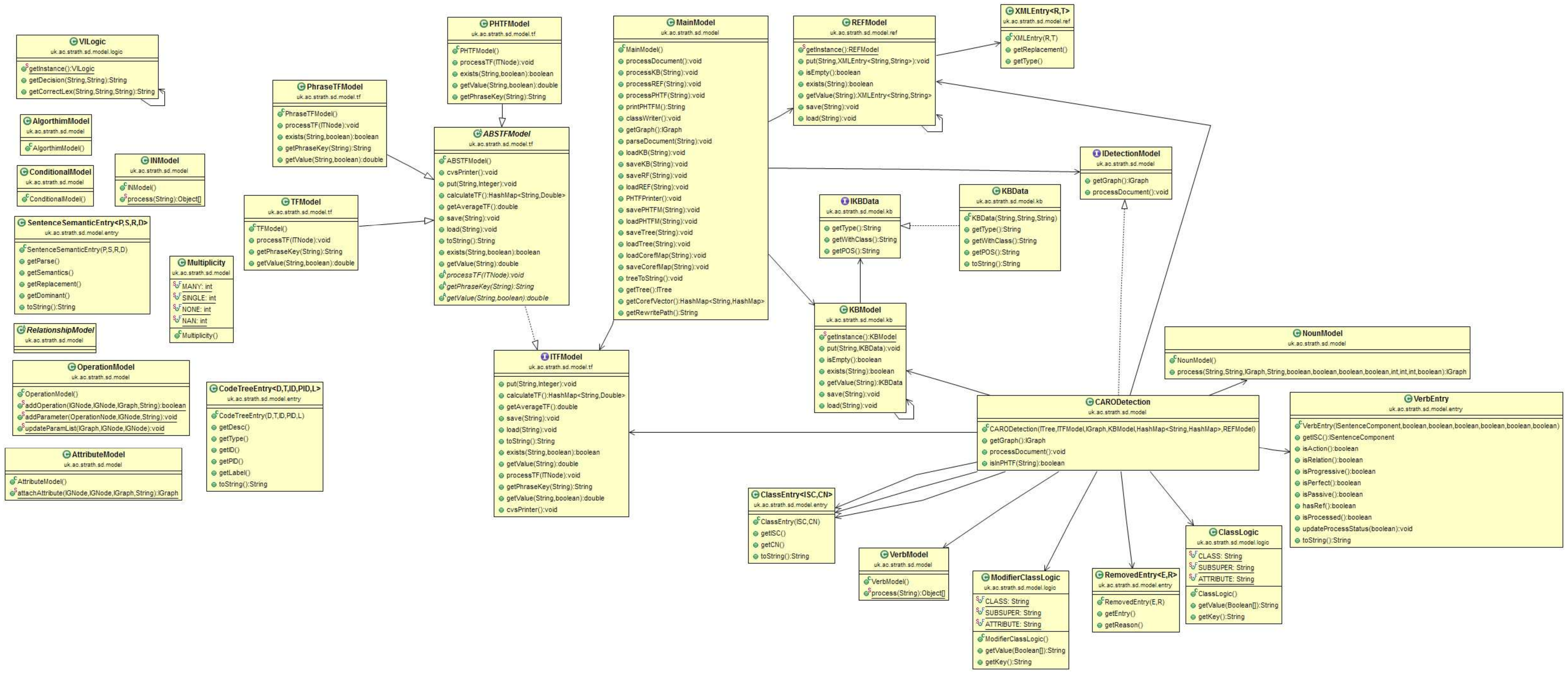
Package: uk.ac.strath.sd.xml


XMLWriter
uk.ac.strath.sd.xml
XMLWriter()
createSentenceTag(String,String,String,String):String[]
xmlConstruct(boolean,String,String):String
SpecParser
uk.ac.strath.sd.xml
SpecParser(MainModel)
parse(String):void
TFParser
uk.ac.strath.sd.xml
TFParser(TFModel)
parse(String):void
TFHandler
uk.ac.strath.sd.xml.handlers
TFHandler(TFModel)
startDocument():void
endDocument():void
startElement(String,String,String,Attributes):void
endElement(String,String,String):void
characters(char[],int,int):void
XMLRewriteHandler
uk.ac.strath.sd.xml.handlers
XMLRewriteHandler(MainModel)
startDocument():void
endDocument():void
startElement(String,String,String,Attributes):void
endElement(String,String,String):void
characters(char[],int,int):void
NLPHandler
uk.ac.strath.sd.xml.handlers
NLPHandler(MainModel)
startDocument():void
endDocument():void
startElement(String,String,String,Attributes):void
endElement(String,String,String):void
characters(char[],int,int):void
ValidationHandler
uk.ac.strath.sd.xml.handlers
ValidationHandler()
startDocument():void
endDocument():void
error(SAXParseException):void
fatalError(SAXParseException):void
warning(SAXParseException):void
checkError():boolean
checkFatalError():boolean
checkWarning():boolean
getErrorReport():Vector<String>
getFatalErrorReport():Vector<String>
getWarningReport():Vector<String>
KBParser
uk.ac.strath.sd.xml
KBParser(KBModel)
parse(String):void
KBHandler
uk.ac.strath.sd.xml.handlers
KBHandler(KBModel)
startDocument():void
endDocument():void
startElement(String,String,String,Attributes):void
endElement(String,String,String):void
REFHandler
uk.ac.strath.sd.xml.handlers
REFHandler(REFModel)
startDocument():void
endDocument():void
startElement(String,String,String,Attributes):void
endElement(String,String,String):void
REFParser
uk.ac.strath.sd.xml
REFParser(REFModel)
parse(String):void

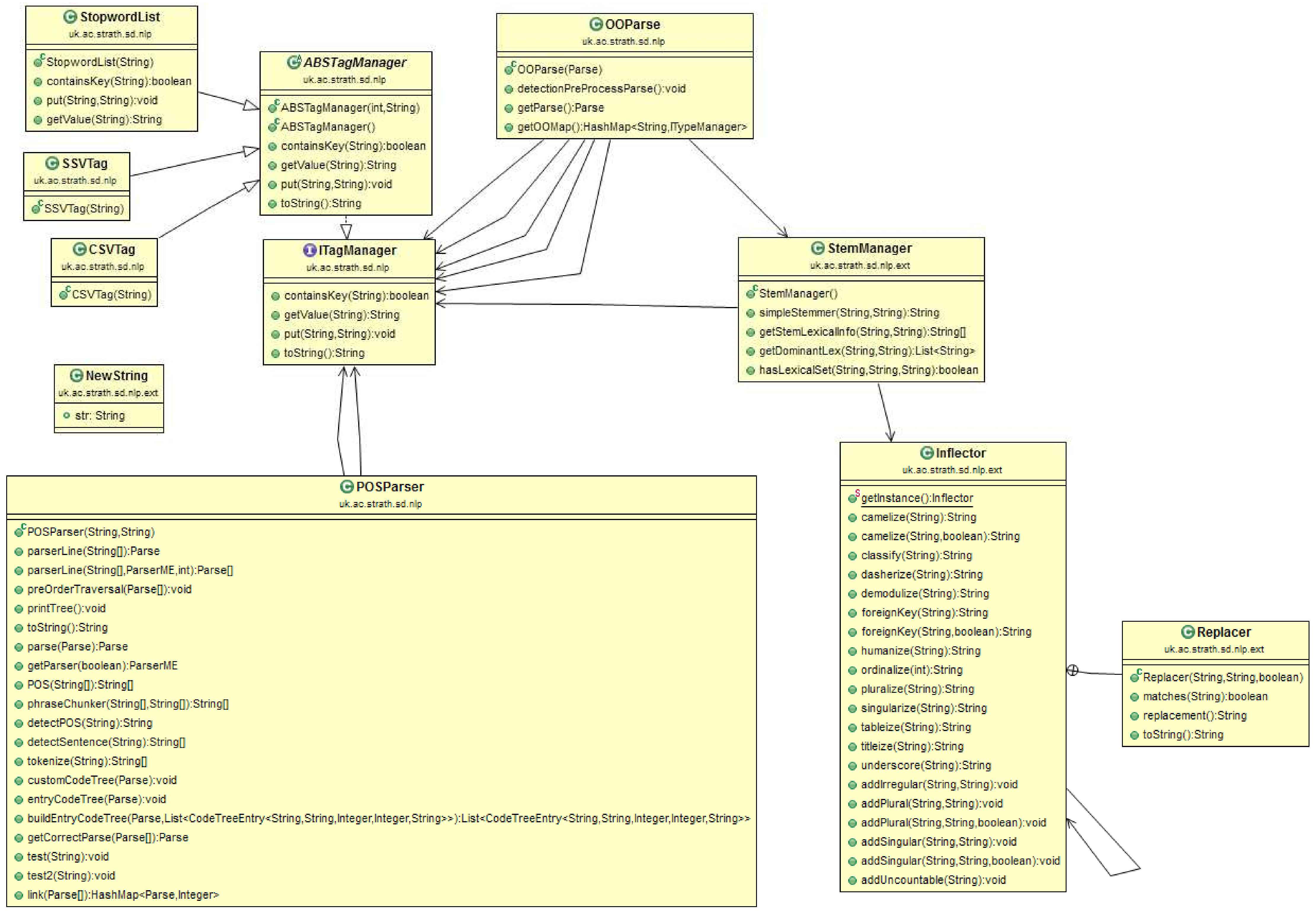
Package: uk.ac.strath.sd.jccg


GeneralisationEdge
uk.ac.strath.sd.jccg.edge
GeneralisationEdge(IGNode,String,String)
AggregationEdge
uk.ac.strath.sd.jccg.edge
AggregationEdge(IGNode,String,String)
ABSRelationEdge
uk.ac.strath.sd.jccg.edge
ABSRelationEdge(IGNode,String,String,String)
setMultiplicity(int,int,int,int):void
getSourceStartMultiplicity():int
getSourceEndMultiplicity():int
getTargetStartMultiplicity():int
getTargetEndMultiplicity():int
CompositionEdge
uk.ac.strath.sd.jccg.edge
CompositionEdge(IGNode,String,String)
AssocEdge
uk.ac.strath.sd.jccg.edge
AssocEdge(IGNode,String,String)
ExtendsEdge
uk.ac.strath.sd.jccg.edge
ExtendsEdge(IGNode,String,String)
ABSEdge
uk.ac.strath.sd.jccg.edge
ABSEdge(IGNode,String,String,String)
getNode():IGNode
toString():String
getID():String
getEdgeType():String
getEdgeName():String
ConditionalEdge
uk.ac.strath.sd.jccg.edge
ConditionalEdge(IGNode,String,String)
AttributeEdge
uk.ac.strath.sd.jccg.edge
AttributeEdge(IGNode,String,String)
OperationEdge
uk.ac.strath.sd.jccg.edge
OperationEdge(IGNode,String,String)
AlgorthimEdge
uk.ac.strath.sd.jccg.edge
AlgorthimEdge(IGNode,String,String)
SignatureEdge
uk.ac.strath.sd.jccg.edge
SignatureEdge(IGNode,String,String)
IEdge
uk.ac.strath.sd.jccg.edge
getNode():IGNode
toString():String
getID():String
getEdgeType():String
getEdgeName():String
-_outEdges
-_inEdges
0..*
0..*
-_node
0..1
IGNode
uk.ac.strath.sd.jccg.node
addEdge(IGNode,String,String,String,String):void
getOutEdges():List<IEdge>
getInEdges():List<IEdge>
addEdge(List<IEdge>,String):void
getName():String
getID():String
addParseID(int):void
addTraceID(String):void
getTraceList():List<String>
getParseIDList():List<Integer>
getLexName():String
setName(String):void
isVisited():boolean
outDegree():int
inDegree():int
setVisited(boolean):void
outEdgesToString(String,String):String
inEdgesToString(String,String):String
containsEdgeTo(IGNode):boolean
getNodeType():String
isPlural():boolean
isInstance():boolean
isAbstract():boolean
isInterface():boolean
isClass():boolean
isAttribute():boolean
isOperation():boolean
hasInterface():boolean
hasAbstract():boolean
setInterface(IGNode):void
setAbstract(IGNode):void
getInterface():IGNode
getAbstract():IGNode
toString():String
hasParseID(int):boolean
outEdgesEmpty():boolean
inEdgesEmpty():boolean
getStartMultiplicity():int
getEndMultiplicity():int
updateMultiplicity(int,int):void
-_interface
-_abstract
0..1
0..1
-_nodeList
0..*
ABSGNode
uk.ac.strath.sd.jccg.node
ABSGNode(String,String,String,boolean,int,int,int,String,boolean)
addParseID(int):void
addTraceID(String):void
getTraceList():List<String>
getParseIDList():List<Integer>
isInstance():boolean
getStartMultiplicity():int
getEndMultiplicity():int
updateMultiplicity(int,int):void
addEdge(IGNode,String,String,String,String):void
addEdge(List<IEdge>,String):void
outEdgesToString(String,String):String
inEdgesToString(String,String):String
containsEdgeTo(IGNode):boolean
isPlural():boolean
getName():String
getID():String
getLexName():String
getInEdges():List<IEdge>
getOutEdges():List<IEdge>
outDegree():int
inDegree():int
setName(String):void
isVisited():boolean
outEdgesEmpty():boolean
inEdgesEmpty():boolean
getNodeType():String
setVisited(boolean):void
setInterface(IGNode):void
getInterface():IGNode
hasInterface():boolean
setAbstract(IGNode):void
isInterface():boolean
isAbstract():boolean
isClass():boolean
isAttribute():boolean
isOperation():boolean
getAbstract():IGNode
hasAbstract():boolean
toString():String
hasParseID(int):boolean
AlgorthimNode
uk.ac.strath.sd.jccg.node
AlgorthimNode(String,String,String,boolean,int)
OperationNode
uk.ac.strath.sd.jccg.node
OperationNode(String,String,String,boolean,int)
addParamater(String,String):void
getParamList():List<String[]>
printParamList(String,String):StringBuffer
AttributeNode
uk.ac.strath.sd.jccg.node
AttributeNode(String,String,String,boolean,int)
ConditionalNode
uk.ac.strath.sd.jccg.node
ConditionalNode(String,String,String,boolean,int)
ClassNode
uk.ac.strath.sd.jccg.node
ClassNode(String,String,String,boolean,int,int,int,boolean)
AbstractClassNode
uk.ac.strath.sd.jccg.node
AbstractClassNode(String,String,String,boolean,int,int,int)
InterfaceNode
uk.ac.strath.sd.jccg.node
InterfaceNode(String,String,String,boolean,int,int,int)
AbsEdgeCreator
uk.ac.strath.sd.jccg.edge
AbsEdgeCreator()
prepareEdge(String,IGNode,String,String):IEdge
-_ec
0..1
EdgeCreator
uk.ac.strath.sd.jccg.edge
EdgeCreator()
GNodeCreator
uk.ac.strath.sd.jccg.node
GNodeCreator()
AbsNodeCreator
uk.ac.strath.sd.jccg.node
AbsNodeCreator()
prepareNode(String,String,String,String,boolean,int,int,int,boolean):IGNode
-_nc
0..1
Graph
uk.ac.strath.sd.jccg
Graph()
setProjectName(String):void
getProjectName():String
exists(String):boolean
findNode(List<IEdge>,String):IGNode
findNode(String):IGNode
findNode(int):IGNode
hasNode(int):boolean
getLastElement():IGNode
traverseGraph():void
hasEdge(IGNode,IGNode,String):boolean
deleteNodeHierarchy(IGNode):List<IEdge>
deleteNode(IGNode):boolean
deleteEdge(IGNode,IGNode):boolean
attachNodeHierarchy(IGNode):boolean
addInEdge(IGNode,IGNode,String,String,String):void
addOutEdge(IGNode,IGNode,String,String,String):void
addEdge(String,String,String,String,String):void
addEdge(IGNode,List<IEdge>,String):void
addNode(String,String,String,String,boolean,int,int,int,boolean):IGNode
drawGraph(Graphics):void
inDegree(IGNode):int
outDegree(IGNode):int
isEmpty():boolean
size():int
toString():String
getNodeList():List<IGNode>
IGraph
uk.ac.strath.sd.jccg
addNode(String,String,String,String,boolean,int,int,int,boolean):IGNode
addInEdge(IGNode,IGNode,String,String,String):void
hasEdge(IGNode,IGNode,String):boolean
addOutEdge(IGNode,IGNode,String,String,String):void
addEdge(String,String,String,String,String):void
addEdge(IGNode,List<IEdge>,String):void
outDegree(IGNode):int
inDegree(IGNode):int
size():int
isEmpty():boolean
setProjectName(String):void
exists(String):boolean
drawGraph(Graphics):void
traverseGraph():void
findNode(String):IGNode
deleteNodeHierarchy(IGNode):List<IEdge>
attachNodeHierarchy(IGNode):boolean
deleteNode(IGNode):boolean
deleteEdge(IGNode,IGNode):boolean
findNode(int):IGNode
findNode(List<IEdge>,String):IGNode
hasNode(int):boolean
getLastElement():IGNode
getProjectName():String
getNodeList():List<IGNode>

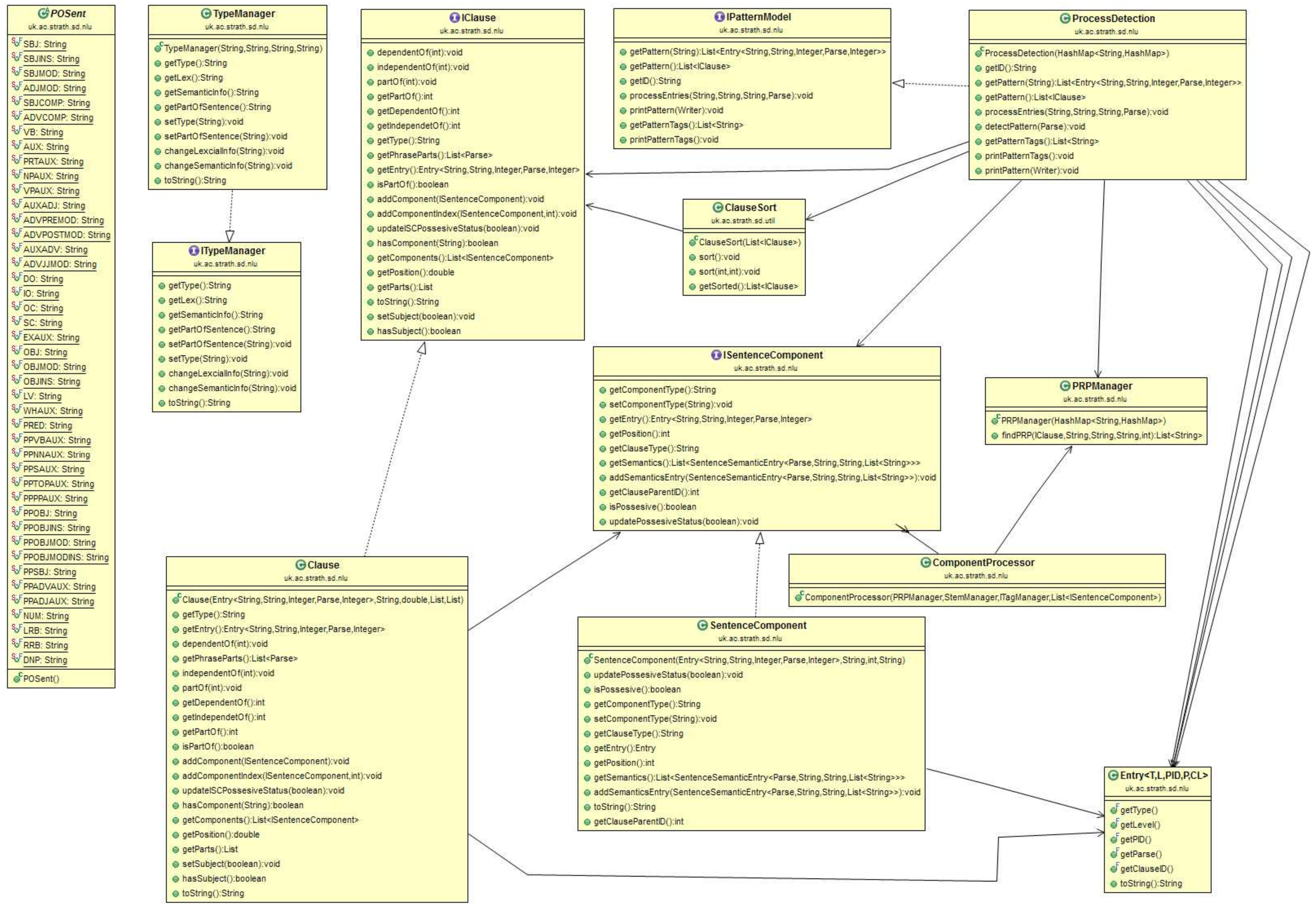
Package: uk.ac.strath.sd.model



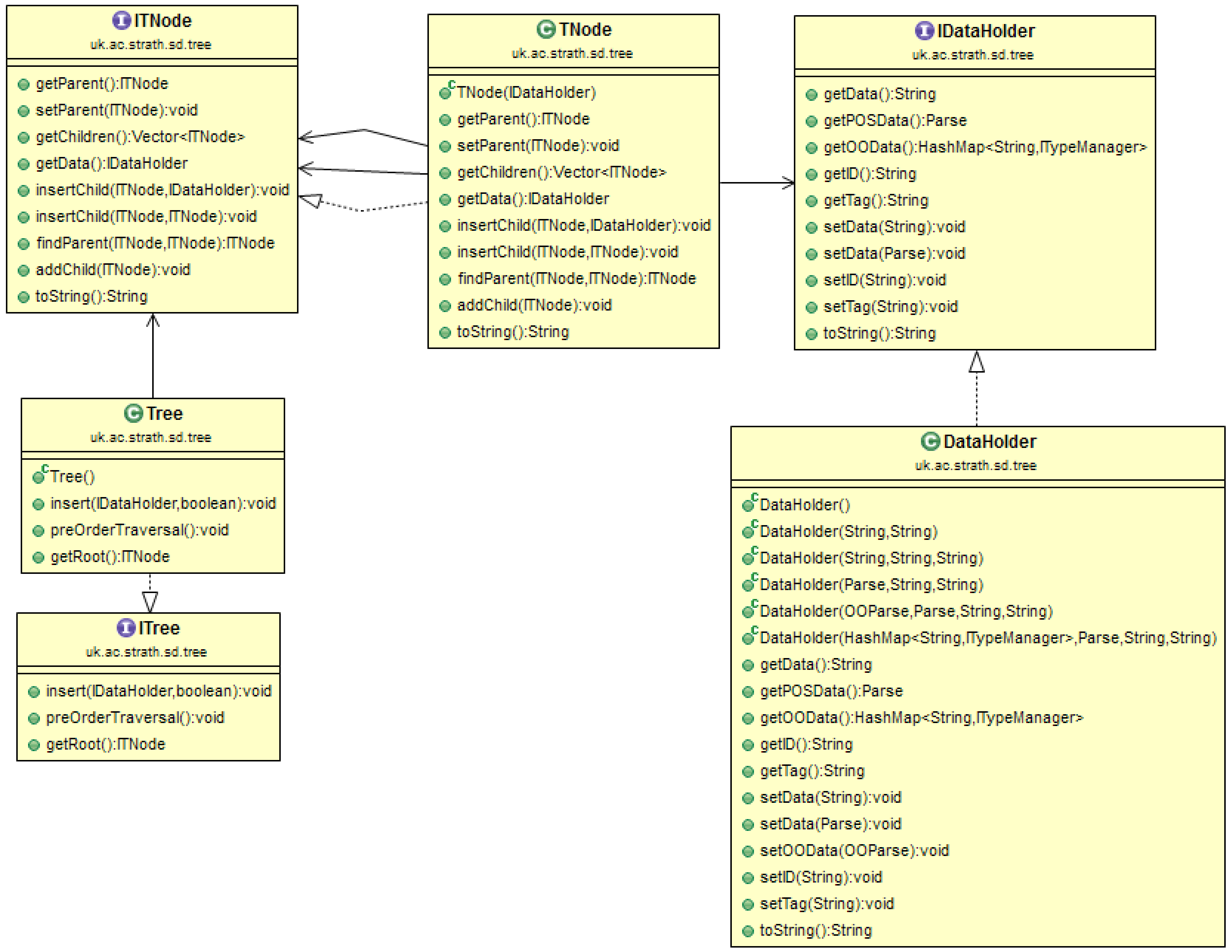
Package: uk.ac.strath.sd.nlp


StopwordList
uk.ac.strath.sd.nlp
StopwordList(String)
containsKey(String):boolean
put(String,String):void
getValue(String):String
ABSTagManager
uk.ac.strath.sd.nlp
ABSTagManager(int,String)
ABSTagManager()
containsKey(String):boolean
getValue(String):String
put(String,String):void
toString():String
OOParse
uk.ac.strath.sd.nlp
OOParse(Parse)
detectionPreProcessParse():void
getParse():Parse
getOOMap():HashMap<String,ITypeManager>
SSVTag
uk.ac.strath.sd.nlp
SSVTag(String)
CSVTag
uk.ac.strath.sd.nlp
CSVTag(String)
ITagManager
uk.ac.strath.sd.nlp
containsKey(String):boolean
getValue(String):String
put(String,String):void
toString():String
StemManager
uk.ac.strath.sd.nlp.ext
StemManager()
simpleStemmer(String,String):String
getStemLexicalInfo(String,String):String[]
getDominantLex(String,String):List<String>
hasLexicalSet(String,String,String):boolean
NewString
uk.ac.strath.sd.nlp.ext
str: String
POSParser
uk.ac.strath.sd.nlp
POSParser(String,String)
parserLine(String[]):Parse
parserLine(String[],ParserME,int):Parse[]
preOrderTraversal(Parse[]):void
printTree():void
toString():String
parse(Parse):Parse
getParser(boolean):ParserME
POS(String[]):String[]
phraseChunker(String[],String[]):String[]
detectPOS(String):String
detectSentence(String):String[]
tokenize(String):String[]
customCodeTree(Parse):void
entryCodeTree(Parse):void
buildEntryCodeTree(Parse,List<CodeTreeEntry<String,String,Integer,Integer,String>>):List<CodeTreeEntry<String,String,Integer,Integer,String>>
getCorrectParse(Parse[]):Parse
test(String):void
test2(String):void
link(Parse[]):HashMap<Parse,Integer>
Inflector
uk.ac.strath.sd.nlp.ext
getInstance():Inflector
camelize(String):String
camelize(String,boolean):String
classify(String):String
dasherize(String):String
demodulize(String):String
foreignKey(String):String
foreignKey(String,boolean):String
humanize(String):String
ordinalize(int):String
pluralize(String):String
singularize(String):String
tableize(String):String
titleize(String):String
underscore(String):String
addIrregular(String,String):void
addPlural(String,String):void
addPlural(String,String,boolean):void
addSingular(String,String):void
addSingular(String,String,boolean):void
addUncountable(String):void
Replacer
uk.ac.strath.sd.nlp.ext
Replacer(String,String,boolean)
matches(String):boolean
replacement():String
toString():String

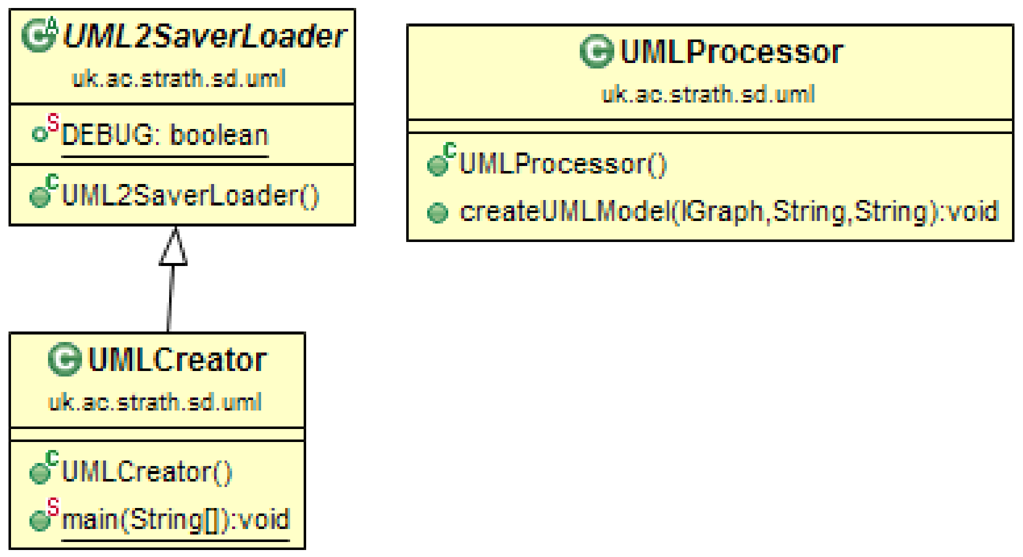
Package: uk.ac.strath.sd.nlu


POSent
uk.ac.strath.sd.nlu
SBJ: String
SBJINS: String
SBJMOD: String
ADJMOD: String
SBJCOMP: String
ADVCOMP: String
VB: String
AUX: String
PRTAUX: String
NPAUX: String
VPAUX: String
AUXADJ: String
ADVPREMOD: String
ADVPOSTMOD: String
AUXADV: String
ADVJJMOD: String
DO: String
IO: String
OC: String
SC: String
EXAUX: String
OBJ: String
OBJMOD: String
OBJINS: String
LV: String
WHAUX: String
PRED: String
PPVBAUX: String
PPNNAUX: String
PPSAUX: String
PPTOPAUX: String
PPPPAUX: String
PPOBJ: String
PPOBJINS: String
PPOBJMOD: String
PPOBJMODINS: String
PPSBJ: String
PPADVAUX: String
PPADJAUX: String
NUM: String
LRB: String
RRB: String
DNP: String
POSent()
TypeManager
uk.ac.strath.sd.nlu
TypeManager(String,String,String,String)
getType():String
getLex():String
getSemanticInfo():String
getPartOfSentence():String
setType(String):void
setPartOfSentence(String):void
changeLexcialInfo(String):void
changeSemanticInfo(String):void
toString():String
ITypeManager
uk.ac.strath.sd.nlu
getType():String
getLex():String
getSemanticInfo():String
getPartOfSentence():String
setPartOfSentence(String):void
setType(String):void
changeLexcialInfo(String):void
changeSemanticInfo(String):void
toString():String
IClause
uk.ac.strath.sd.nlu
dependentOf(int):void
independentOf(int):void
partOf(int):void
getPartOf():int
getDependentOf():int
getIndependetOf():int
getType():String
getPhraseParts():List<Parse>
getEntry():Entry<String,String,Integer,Parse,Integer>
isPartOf():boolean
addComponent(ISentenceComponent):void
addComponentIndex(ISentenceComponent,int):void
updateISCPossesiveStatus(boolean):void
hasComponent(String):boolean
getComponents():List<ISentenceComponent>
getPosition():double
getParts():List
toString():String
setSubject(boolean):void
hasSubject():boolean
IPatternModel
uk.ac.strath.sd.nlu
getPattern(String):List<Entry<String,String,Integer,Parse,Integer>>
getPattern():List<IClause>
getID():String
processEntries(String,String,String,Parse):void
printPattern(Writer):void
getPatternTags():List<String>
printPatternTags():void
ProcessDetection
uk.ac.strath.sd.nlu
ProcessDetection(HashMap<String,HashMap>)
getID():String
getPattern(String):List<Entry<String,String,Integer,Parse,Integer>>
getPattern():List<IClause>
processEntries(String,String,String,Parse):void
detectPattern(Parse):void
getPatternTags():List<String>
printPatternTags():void
printPattern(Writer):void
ClauseSort
uk.ac.strath.sd.util
ClauseSort(List<IClause>)
sort():void
sort(int,int):void
getSorted():List<IClause>
ISentenceComponent
uk.ac.strath.sd.nlu
getComponentType():String
setComponentType(String):void
getEntry():Entry<String,String,Integer,Parse,Integer>
getPosition():int
getClauseType():String
getSemantics():List<SentenceSemanticEntry<Parse,String,String,List<String>>>
addSemanticsEntry(SentenceSemanticEntry<Parse,String,String,List<String>>):void
getClauseParentID():int
isPossesive():boolean
updatePossesiveStatus(boolean):void
PRPManager
uk.ac.strath.sd.nlu
PRPManager(HashMap<String,HashMap>)
findPRP(IClause,String,String,String,int):List<String>
ComponentProcessor
uk.ac.strath.sd.nlu
ComponentProcessor(PRPManager,StemManager,ITagManager,List<ISentenceComponent>)
Clause
uk.ac.strath.sd.nlu
Clause(Entry<String,String,Integer,Parse,Integer>,String,double,List,List)
getType():String
getEntry():Entry<String,String,Integer,Parse,Integer>
dependentOf(int):void
getPhraseParts():List<Parse>
independentOf(int):void
partOf(int):void
getDependentOf():int
getIndependetOf():int
getPartOf():int
isPartOf():boolean
addComponent(ISentenceComponent):void
addComponentIndex(ISentenceComponent,int):void
updateISCPossesiveStatus(boolean):void
hasComponent(String):boolean
getComponents():List<ISentenceComponent>
getPosition():double
getParts():List
setSubject(boolean):void
hasSubject():boolean
toString():String
SentenceComponent
uk.ac.strath.sd.nlu
SentenceComponent(Entry<String,String,Integer,Parse,Integer>,String,int,String)
updatePossesiveStatus(boolean):void
isPossesive():boolean
getComponentType():String
setComponentType(String):void
getClauseType():String
getEntry():Entry
getPosition():int
getSemantics():List<SentenceSemanticEntry<Parse,String,String,List<String>>>
addSemanticsEntry(SentenceSemanticEntry<Parse,String,String,List<String>>):void
toString():String
getClauseParentID():int
Entry<T,L,PID,P,CL>
uk.ac.strath.sd.nlu
getType()
getLevel()
getPID()
getParse()
getClauseID()
toString():String

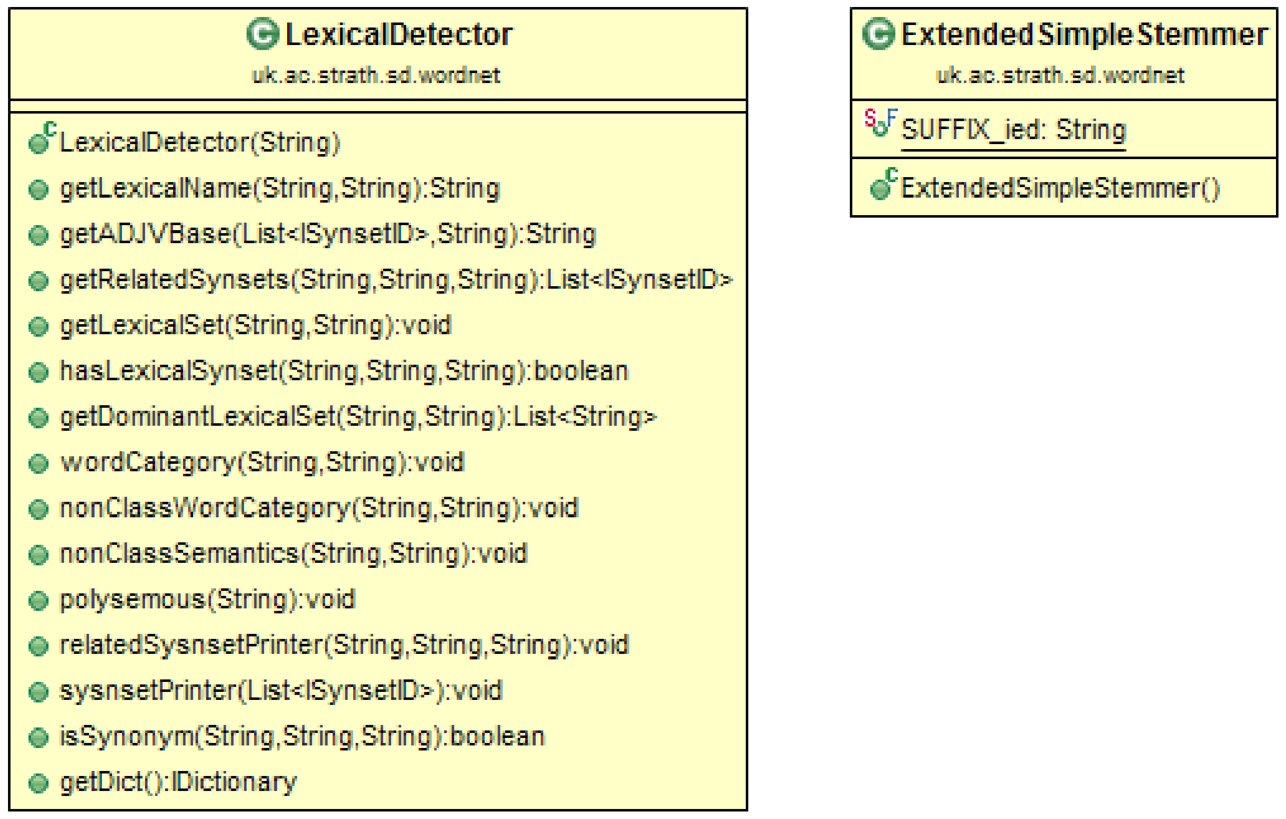
Package: uk.ac.strath.sd.tree



Package: uk.ac.strath.sd.uml



Package: uk.ac.strath.sd.wordnet


LexicalDetector
uk.ac.strath.sd.wordnet
LexicalDetector(String)
getLexicalName(String,String):String
getADJVBase(List<ISynsetID>,String):String
getRelatedSynsets(String,String,String):List<ISynsetID>
getLexicalSet(String,String):void
hasLexicalSynset(String,String,String):boolean
getDominantLexicalSet(String,String):List<String>
wordCategory(String,String):void
nonClassWordCategory(String,String):void
nonClassSemantics(String,String):void
polysemous(String):void
relatedSysnsetPrinter(String,String,String):void
sysnsetPrinter(List<ISynsetID>):void
isSynonym(String,String,String):boolean
getDict():IDictionary
ExtendedSimpleStemmer
uk.ac.strath.sd.wordnet
SUFFIX_ied: String
ExtendedSimpleStemmer()

Appendix B.1 Specification Details

Keyword in Context [Par72]:

The KWIC index system accepts an ordered set of lines, each line is an ordered set of words, and each word is an ordered set of characters. Any line may be "circularly shifted" by repeatedly removing the first word and appending it at the end of the line. The KWIC index system outputs a listing of all circular shifts of all lines in alphabetical order.

Organisation Problem 1:

The Course Administration System database for ABC University needs to keep track of each Instructor with id, name, and address. Each instructor works for one department and each department have at least one instructor. The departments have a unique id and a name. Courses are offered by a single department and have a name, and number unique to each department. Each course has at least one section. Store the course name, credits, and description. A section has numbers for each course for storing the section semester, year, and size. Students have student ids and names. Each student has a single instructor as an advisor. Students enrol in one or more sections. A section must have five students or it is cancelled. A section is taught by at least one instructor.

Organisation Problem 2 [Cur95]:

Each department in an organisation consists of a manager and several departmental staff. Each manager is in charge of only one department and departmental staff are assigned to a single department.

Several projects are attached to each department. All departmental staff are assigned to projects, with some staff being assigned to several projects, not necessarily in the same department. Each project is run by a management group that consists of the manager of the department together with a selection of staff working on the project. No departmental staff member is ever required to sit on more than one management group.

ATM Problem [RBPEL91]:

Design the software to support a computerised banking network including both human cashiers and automatic teller machines (ATMs) to be shared by a consortium of banks. Each bank provides its own computer to maintain its own accounts and process transactions against them Cashier stations are owned by individual banks and communicate directly with their own banks computers. Human cashiers enter account and transaction data. Automatic teller machines communicate with a central computer which clears transactions with the appropriate banks. An automatic teller machine accepts a cash card, interacts with the user, communicates with the central system to carry out the transaction, dispenses cash, and prints receipts. The system requires appropriate recordkeeping and security provisions. The system must handle concurrent accesses to the same account correctly. The banks will provide their own software for their own computers.

Cinema Problem [CIS08]:

The cinema leases films for screening from film distributors. Each lease is for one copy of the film. The cinema may lease more than one copy of films that are very popular.

The cinema operation is organised around a screening schedule, which is a timetable listing the films that will be shown on each screen each day of the week. This screening schedule is different every week. During its release period a particular film can be shown on a number of different screens. The same film cannot be shown on more than one screen at a time unless there are multiple copies.

Screenings are open for ticket sales one week before the date they take place. There are two kinds of screenings: seated and unseated ones. The main difference between the two is that for seated screenings the customer is allocated a particular seat, while for unseated screenings no specific seat is allocated. For each screening the total number of tickets sold should not exceed the seating capacity for that screen. There are a number of different types of tickets associated with each screening, which include normal tickets, concessionary tickets for students and senior citizens, discounted family tickets, etc. The price of each type of ticket may be different for each screening. For example, matinee screenings usually have a lower ticket price than evening screenings, while weekend screenings usually have higher ticket prices.

The cinema wishes to operate a customer cinema card scheme. According to this scheme every subscribed customer pays a monthly subscription, which allows them to buy a fixed number of tickets for any screening during the month.

Regarding the films, information that is important includes the film's classification (determined by the board of film classification) as well as its duration. This information is important as it affects the scheduling process and the allocation of films to screens.

For the system to be able to support the cinema management team it should be able to produce the following kinds of statistics: the number of ticket sales to date per film, the revenue of the ticket sales per film, the percentage of empty seats for each screening for the current or future weeks, the ticket sales and revenue for each screening for the current week, a listing of films ordered by ticket sales or revenue for the current week. It should also allow the management team to enter the new screening schedule and make changes to the current screening schedule.

Ticket sales are handled by cinema staff and payment can be made in three forms: by cash; by credit or debit card; by using cinema membership cards. In the case where the sale is for a seated screening the customer should be able to select the seats they most prefer from those that are available.

Cinema cards are personal (i.e. only the person named on the card can use it) and they are limited to a maximum of four tickets per screening. When signing up for a cinema card, the following are required: a photograph of the customer which is taken on the spot and is attached to the card, customer information such as name and address, and credit or debit card details for the monthly subscription charge. The cards are valid for six months from the date of issue and each month the customer is charged the monthly subscription.

Library Problem 1 [EP98]:

A software system to support a library is to be developed. A library lends books and magazines to borrowers. These borrowers, books and magazines are registered in the system.

A library handles the purchase of new titles for the library. Popular titles are bought in multiple copies. Old books and magazines are removed when they are too old or in poor condition.

The librarian is an employee of the library who interacts with the borrowers and whose work is supported by the system. A borrower can reserve a book or a magazine that is not currently available in the library. So that, when it is returned or purchased by the library, that person is notified.

The reservation is cancelled when the borrower checks out the book or magazine or through an explicit cancelling procedure. The library can easily manage the information about the books. It can create, update or delete the information. The information concerns the titles, the borrowers, the loans and the reservations. The system can run on all popular environments such as Windows, UNIX. It has a modern graphical user interface. The system is also easy to extend with new functionality.

Library Problem 2 [Cal94]:

A library issues loan items to customers. Each customer is known as a member and is issued a membership card that shows a unique member number. Along with the membership number, other details on a customer must be kept such as a name, address, and date of birth.

The library is made up of a number of subject sections. Each section is denoted by a classification mark.

A loan item is uniquely identified by a bar code. There are two types of loan items, language tapes, and books. A language tape has a title language (e.g. French), and level (e.g. beginner). A book has a title, and authors.

A customer may borrow up to a maximum of 8 items. An item can be borrowed, reserved or renewed to extend a current loan. When an item is issued the customer's membership number is scanned via a bar code reader or entered manually. If the membership is still valid and the number of items on loan less than 8, the book bar code is read, either via the bar code reader or entered manually. If the item can be issued (e.g. not reserved) the item is stamped and then issued.

The library must support the facility for an item to be searched and for a daily update of records.

Library Problem 3 [Cur95]:

When a library first receives a book from a publisher it is sent, together with the accompanying delivery note, to the library desk. Here the delivery note is checked against a file of books ordered. If no order can be found to match the note, a letter of enquiry is sent to the publishers. If a matching order is found, a catalogue note is prepared from the details on the validated delivery note.

The catalogue note, together with the book, is sent to the registration department. The validated delivery note is sent to the accounts department, where it is stored.

On receipt of an invoice from the public the accounts department checks its store of delivery notes. If the corresponding delivery note is found then an instruction to pay the publishers is made, and subsequently a cheque is sent. If no corresponding delivery note is found, the invoice is stored in a pending file.

Filing Problem [Der95]:

An electronic filing program (EFP) can be used to store and retrieve text documents. Any document created by a word processor, editor, or other means may be stored in the electronic filing system. Documents may be filed along with keywords, authors, and/or a document description or abstract describing the document. Documents filed in the system may also be removed or deleted.

Documents stored in the EFP are indexed to enable rapid retrieval. Documents are retrievable according to convenient schemes not found in conventional classifications; e.g. users may retrieve or locate documents based on their content, description, author or a user defined keywords. Therefore, the document description, authors, keywords, and/or the actual text document itself may be searched.

A user may specify a search criteria, which results in a number of documents being found that meet the specified search criteria. The user may then continue to specify additional search criteria, successively narrowing down the search until the required documents are found. Documents found that meet the search criteria may then be viewed or printed.

The user is provided with the capability of specifying any extraneous or junk words which if found in the content of the document will not be searched or indexed. The user can also specify which alphanumeric characters will be indexed and searched (the filing character set), thereby limiting the search and index to only portions of a document.

Exam Problem:

Faculty members in ABC University have to prepare exams. If they teach a course more than one time, it makes sense to collect and reuse exams. But no two course instances are identical, and, even if they were, there are other reasons for avoiding exact duplication of exams. Hence, it makes sense to manage exam questions and their answers in some kind of repository and then build custom exams to conform to specific course content. For large courses, electronic grading of exams is desirable, implying that exams have a multiple-choice format.

A software system should be built to manage exam questions and their answers. An exam consists of a set of multiple choice questions. There may be multiple versions of the same exam. A question may appear in more than one version, but it will rarely appear in the same place (i.e., the same question number). Questions may have different point values. Students indicate an answer by providing a letter to denote one of the offered alternatives. Questions may have multiple parts, and partial credit may be awarded for parts correctly answered. There may be more than one correct answer to a question. The system should determine the correctness of each answer, provide a numerical value for it, and sum the scores for the exam. For assessment purposes, it should also be able to gather statistics about student performance on each question regardless of the exam it appeared on.

Students provide answers to questions on a bubble sheet. Automatic processing of the answer sheets is provided by a separate program. That program produces as output an ASCII data file that the new program should process. The data file consists of a sequence of records. Each record is fixed format, consisting of the student's name, student number, and a vector of letters, one for each requested answer. When a student fails to properly bubble in an entry in the form, a space appears in place of the letter. Moreover, if a student bubbles in more than one selection for the same entry, then an asterisk character will appear in the corresponding place in the record.

Personnel Problem [Duf95]:

The personnel department of a large research institute is responsible for the purchase and dissemination of journals to readers in other departments in the organization.

Readers may be interested in certain specific topics relating to their research interests, while it is also possible to be placed on a circulation list. Usually, readers get access to an issue of a journal for a fixed period of time, typically two weeks. It is possible to have access to an issue for a longer period of time, but permission must be granted from the personnel department.

Journals appear on a regular basis and each journal contains information on the publisher, language and frequency of publication. The system should keep readers informed of the topics that are of interest to them and which appear in the different journals. Furthermore, it should be possible for readers to find articles which deal with topics that they are interested in.

Hospital Problem 1 [Cur95]:

The EDP department of a medium-sized hospital wishes to create a system which will assist in the administration of its wards, operating theatres and private rooms. Furthermore, information relating to patients, surgeons and nurses needs to be registered.

The following information should be present in the system:

Patients are assigned to a ward when admitted to the hospital unless they are private patients, in which case they will be assigned to a private room and they are treated by consultants. A consultant is a senior surgeon. Each room has a unique identification number. The attributes of a patient are patient name, number, address, sex, date of birth and blood group.

A nurse may or may not be assigned to a ward. However, a nurse may not be assigned to more than one ward. A ward may have many nurses assigned to it. Nurse attributes include name, address, phone number and grade. Ward attributes include unique identification number and its type (e.g. maternity, paediatric). A patient may undergo a number of operations.

Only one surgeon may perform an operation, while other surgeons may assist at operations. Surgeons are coached by consultants who are experienced surgeons. A consultant may assist or perform at an operation. Surgeon attributes include name, address and phone number. Each consultant specializes in a certain area.

An operation is performed in only one theatre and a given theatre may be the venue for many operations. Theatre attributes include identifying number and some may be specially equipped for certain classes of operation.

A nurse may be assigned to an operating theatre. However, a nurse may not be assigned to more than one theatre. A theatre may have many nurses assigned to it.

Hospital Problem 2 [Duf95]:

A local hospital consists of many wards, each of which is assigned many patients. Each patient is assigned to one doctor, who has overall responsibility for the patients in his or her care. Other doctors are assigned on an advisory basis. Each patient is prescribed drugs by the doctor responsible for that patient.

Each nurse is assigned to a ward and nurses all patients on the ward, though is given special responsibility for some patients. Each patient is assigned one nurse in this position of responsibility. One of the doctors is attached to each ward as an overall medical advisor.

Lift Problem [PRM+07]:

A lift consists of a door, a motor, and a lift controller. The lift controller is responsible for controlling the lift system. Passengers interact with the lift system by passing buttons on the individual floors or on the control panel inside the lift. Normally, the lift stays on the ground floor (0-th floor) of a building. If a passenger enters the lift and presses the button for the k-th floor, the lift will move up to the k-th floor. When the lift arrives at requested floor (say k-th floor), it opens the door for a certain period M seconds of time and closes them. The lift then becomes idle.

A passenger on m-th floor calls a lift by pressing the up or down button. The lift will move to the m-th floor and open the door on arrival. The passenger requests to go to a particular floor by pressing the corresponding button on the control panel inside the lift. If there is no passenger interaction on the control panel within M seconds, then the lift will return to the ground floor.

Meeting Problem [RBP+91]:

Softcom needs a computer system to support athletic meetings for judged sports, such as gymnastics, diving or figure skating. Meetings for these sports take place during the season. A season goes on for several months.

Competitors register to take part to a meeting. They belong to teams and teams belong to leagues. Each meeting consists of various competitions, such as routines, figures or styles. Figures correspond to different difficulties and therefore the have different point values.

Competitors can enter many competitions. In a particular competition, competitors receive a number which is announced and used to split them into groups. There is a panel of judges who give a subjective score for the competitors' performance. Working from stations, the judges can score many competitions.

A competition consists of some trials. Competitors receive a score for each trial of a competition. The scores for the trials are read at each station. The system eliminates both the highest and the lowest score. The other scores are then processed and the net score is determined. Final prizes are based on the net scores.

Taxi Problem [BAR12]:

The company operates both individual taxis and shuttles. The taxis are used to transport an individual (or small group) from one location to another. The shuttles are used to pick up individuals from different locations and transport them to their several destinations.

When the company receives a call from; individual, hotel, entertainment venue, tourist organization, it tries to schedule a vehicle to pick up the fare. If it has no free vehicles, it does not operate any form of queuing system.

When a vehicle arrives at a pick up location, the driver notifies the company. Similarly, when a passenger is dropped off at their destination, the driver notifies the company.

Gizmo-ball Problem [MIT05]:

Your implementation must support two modes of execution building and running. In building mode, the user can add gizmos to the playing area and can modify the existing ones. In running mode, a ball moves around the playing area and interacts with the gizmos.

To describe dimensions in the playing area, we define L be the basic distance unit, equal to the edge length of a square bumper. Corresponding to standard usage in the graphics community, the origin is in the upper left-hand corner with coordinates increasing to the right and down.

The playing area must be at least 20 L wide by 20 L high. That is, 400 square bumpers could be placed on the playing area without overlapping. The upper left corner is (0,0) and the lower right corner is (20,20). When we say a gizmo is at a particular location, that means that the gizmo's origin is at that location. The origin of each of the standard gizmos is the upper left-hand corner of its bounding box, so the location furthest from the origin at which a gizmo may be placed is (19,19) on a 20L x 20L board. The origin of a ball is at its center.

During building mode, Gizmos should snap to a 1 L by 1 L grid. That is, a user may only place gizmos at locations (0,0), (0,1), (0,2), and so on.

During running mode the animation grid may be no coarser than 0.05 L by 0.05 L. Rotating flippers can be animated somewhat more coarsely. If the ball is moving faster than the animation grid size per frame redraw, it need not be redrawn in each animation grid position.

In building mode the user can add any of the available types of gizmos to the playing area. An attempt to place a gizmo in such a way that it overlaps a previously placed gizmo or the boundary of the playing area should be rejected.

Move a gizmo from one place to another on the playing area. An attempt to place a gizmo in such a way that it overlaps a previously placed gizmo or the boundary of the playing area should be rejected.

Apply a 90 degree clockwise rotation to any gizmo. Rotation has no effect on gizmos with rotational symmetry. For example, circular bumpers look and act the same, no matter how many times they have been rotated by 90 degrees.

Connect a particular gizmo's trigger to a particular gizmo's action. The standard gizmos produce a trigger when hit by the ball, and exhibit at most one action (for example, moving a flipper, shooting the ball out of an absorber, or changing the color of a bumper). The trigger that a gizmo produces can be connected to the actions of many gizmos. Likewise, a gizmo's action can be activated by many triggers. The required triggers and actions for the basic gizmos are described below. Note that triggers do not chain. That is, when A is connected to B and B is connected to C, a ball hitting A should only cause the action of B to be triggered.

Connect a key-press trigger to the action of a gizmo. Each keyboard key generates a unique trigger when pressed. As with gizmo-generated triggers, key-press triggers can also be connected to the actions of many gizmos.

Delete a gizmo from the playing area.

Add a ball to the playing area. The user should be able to specify a position and velocity. An attempt to place the ball in such a way that it overlaps a previously placed gizmo or the boundary of the playing area should be rejected (i.e., it should have no effect). There is one exception in the standard gizmo set: a stationary ball may be placed inside an absorber.

Save to a file named by the user. You must be able to save to a file in the standard format given in Appendix 2. You may, if you wish, define an extension to the standard format that handles special features of your implementation. If you do so, the user must have the choice of saving in the standard format or in your special format. The saved file must include information about all the gizmos currently in the playing area, all of the connections between triggers and actions, and the current position and velocity of the ball.

Load from a file named by the user. You must be able to load a game saved in the standard format.

Switch to running mode.

Quit the application.

In running mode the user can press keys, thereby generating triggers that may be connected to the actions of gizmos.

Switch to building mode at any time If the user requests to switch to building mode while a flipper is in motion, it is acceptable to delay switching until the flipper has reached the end of its trajectory. Similar short delays in order to finish transitional states of gizmos you create are also acceptable.

Quit the application.

Provide visually smooth animation of the motion of the ball. The ball by default must have a diameter of approximately 0.5 L. Ball velocities must range at least from 0.01 L/sec to 200 L/sec and can cover a larger range if you wish. 0 L/sec (stationary) must also be supported. An acceptable frame rate should be used to generate a smooth animation. We have found that 20 frames per second tends to work well across a reasonably wide range of platforms.

Provide intuitively reasonable interactions between the ball and the gizmos in the playing area. That is, the ball should bounce in the direction and with the resulting velocity that you would expect it to bounce in a physical pinball game.

Continually modify the velocity of the ball to account for the effects of gravity. You should support the standard gravity value of 25 L/sec2, which resembles a pinball game with a slightly tilted playing surface.

Continually modify the velocity of the ball to account for the effects of friction. You should model friction by scaling the velocity of the ball using the frictional constants mu and mu2. The default value of mu should be 0.025 per second. The default value of mu2 should be 0.025 per L.

There are seven standard gizmos that must be supported: bumpers (square, circular, and triangular), flippers (left and right), absorbers, and outer walls. A coefficient of reflection of 1.0 means that the energy of the ball leaving the bumper is equal to the energy with which it hit the bumper, but the ball is traveling in a different direction. As an extension, you may support bumpers with coefficients above or below 1.0 as well.

A square bumper is a square shape with an edge length of 1L. Its trigger is generated whenever the ball hits it and no action. The coefficient of reflection is 1.0.

A circular bumper is a circular shape with a diameter of 1L. Its trigger is generated whenever the ball hits it and no action. The coefficient of reflection is 1.0.

A triangle bumper is a right-triangular shape with sides of length 1L and hypotenuse of length Sqrt(2)L. Its trigger is generated whenever the ball hits it and no action required. The coefficient of reflection is 1.0.

A flipper is generally rectangular rotating shape with bounding box of size 2Lx2L. Its trigger is generated whenever the ball hits it and rotates 90 degrees. The coefficient of reflection is 0.95.

Flipper's are required to come in two different varieties, left flippers and right flippers. A left flipper begins its rotation in a counter-clockwise and a right flipper begins its rotation in a clockwise direction.

An absorber is a square shape with integral length sides. Its trigger is generated whenever the ball hits it and shoots out a stored ball. The absorber as no coefficient of reflection, but the ball is captured

When a ball hits an absorber, the absorber stops the ball and holds it in the bottom right-hand corner of the absorber.

If the absorber is holding a ball, then the action of an absorber, when it is triggered, is to shoot the ball straight upwards in the direction of the top of the playing area. By default, the initial velocity of the ball should be 50L/sec. (With the default gravity and the default values for friction, the value of 50L/sec gives the ball enough energy to lightly collide with the top wall, if the bottom of the absorber is at y=20L.) If the absorber is not holding the ball, or if the previously ejected ball has not yet left the absorber, then the absorber takes no action when it receives a trigger signal.

Absorber's cannot be rotated.

Outer walls are impermeable barriers surrounding the playing field. Its trigger is generated whenever the ball hits it and has no action. The coefficient of reflection is 1.0.

A Gizmoball game supports exactly one set of outer walls. The user cannot move, delete, or rotate the outer walls. The outer walls lie just outside the playing area

Appendix B.2 Manual Specification Classifications Data

Organisation Problem 1:

Class Detection Data

Orig Model	Classes	Correct	Missing	Incorrect	Extra	SYM
Department(D)	Department(D)	X				
Course(C)	Course (C)	X				
Student(S)	Student(S)	X				
Section(SE)	Section(SE)	X				
Instructor(I)	Instructor(I)	X				
	Instructor Work(IW)					
	Section Semester(SS)			X		
	Year(Y)			X		
	Number(N)			X		
	Course Name (CN)			X		
	Name(NA)			X		
	Student ID(SI)			X		
	Advisor(A)				X	
	Description			X		
	Track			X		
	Store			X		
	University				X	
Totals		5	0	9	2	0

Relationships Detection Data

Orig Rel	ASA Rel	Correct	Missing	Incorrect	Extra
D→I			X		
I→SE	I→SE	X			
SE→C	SE→C	X			
C→D			X		
S→I			X		
S→SE	S→SE	X		X	
	N→C			X	
	N→SE			X	
	C→SE			X	
	SS→S			X	
	IW→I			X	
	I→N			X	
	N→S			X	
	SE→Y			X	
	S→SI			X	
	S→A				X
Totals		3	3	10	2

Measures Results

	Recall	Precision	Over-Specification	F-Measure
Classes	*1.0*	*0.33*	*0.4*	*0.5*
Relationships	*0.5*	*0.23*	*0.33*	*0.31*

Organisation Problem 2 [Cur95]:

Class Detection Data

Orig Model	Classes	Correct	Missing	Incorrect	Extra	SYM
Department(D)	Department	X				
Project(P)	FN		X			
Staff(S)	Staff	X				
Manager(M)	Manager	X				
	Staff Member					X
	Management Group				X	
	Charge(C)			X		
Totals		3	1	1	1	1

Relationships Detection Data

Orig Model	Classes	Correct	Missing	Incorrect	Extra
P→S			X		
D→P			X		
D→M	M→D	X			
	D→S			X	
	D→S				X
	M→C			X	
	C→S			X	
Totals		1	2	3	1

Measures Results

	Recall	Precision	Over-Specification	F-Measure
Classes	0.57	0.50	0.43	0.53
Relationships	0.0	0.0	1.14	0.0

ATM Problem [RBPEL91]:

Class Detection Data

Orig Model	Classes	Correct	Missing	Incorrect	Extra	SYM
Account (A)	Account(A)	X				
ATM (ATM)	Automatic Teller Machine(ATM)	X				
Bank (B)	Bank(B)	X				
Cash Card (CC)	Cash Card (CC)	X				
Cashier (C)	Cashier	X				
Cashier Station (CS)	fn		X			
Central Computer (CCO)	Computer	X				
Remote Transaction (RT)	fn		X			
Cashier Transaction (CT)	fn		X			
Bank Computer (BC)	Bank Computer (BC)	X				
Consortium(CO)	Consortium(CO)	X				
Customer (CU)	User(U)	X				
	Human Cashier (C)					X
	Banking Network (BN)				X	
	Design (D)			X		
	ATMs) (ATM)					X
	RecordKeepe (RK)			X		
	Teller Machine (ATM)					X
	Software (S)			X		
	System (SY)			X		
	Security Provision (SP)				X	
	Communicate (COM)			X		
	Interact (I)			X		
Totals		9	3	6	2	3

Relationship Detection Data

Orig Model	Relationships	Correct	Missing	Incorrect	Extra
ATM→RT			X		
CCO→ATM	CCO→ATM	X			
CO→CCO			X		
CCO→BC			X		
CO→B	CO→B	X			
BC→CS			X		
B→BC			X		
B→CS			X		
B→C	B→C	X			
B→A			X		
RT→A			X		
RT→CC			X		
CS→CT			X		
A→CT			X		
A→CC			X		
A→CU			X		
CC→CU			X		
	CC→ATM				X
	ATM→BN				X
	CCO→S			X	
	B→S			X	
	BN→S			X	
	BN→C				X
	C→A				X
	A→SY			X	
	SY→SP			X	
	SY→COM			X	
	COM→I			X	
	I→U			X	
Totals		3	12	8	4

Measures Results

	Recall	Precision	Over-Specification	F-Measure
Classes	*0.75*	*0.60*	*0.17*	*0.67*
Relationships	*0.20*	*0.27*	*0.27*	*0.23*

Cinema Problem [CIS08]:

Class Detection Data

Orig Model	Classes	Correct	Missing	Incorrect	Extra	SYM
Film(F)	Film(F)	X				
Screen(S)	Screen(S)	X				
Weekly Showing Schedule (WSS)	TimeTable (TT)	X				
Showing(SH)	Screening (SC)	X				
Seated Showing(SS)	Seated Screen(SS)	X				
Unseated Showing (US)	fn		X			
Family Ticket (FT)	fn		X			
Ticket(T)	Ticket(T)	X				
Sale(S)	fn		X			
Card Sale(CS)	np		X			
Cinema Card Sale (CCS)	np		X			
Cinema Card (CC)	Cinema Card (CC)	X				
Customer (C)	Customer (C)	X				
	Student(ST)				X	
	Place (P)			X		
	Type(TY)			X		
	Number(N)			X		
	Citizen(CI)				X	
	Month(M)			X		
	Film Distributor(FD)				X	
	Seat(SE)				X	
	Photograph(P)			X		
	Cinema(CIN)				X	
	Copy(CO)			X		
	List(L)				X	
	Day(D)			X		
	Week(W)			X		
	Matinee Screening (MS)				X	
	Weekend Screening(WS)				X	
	Cinema Management Team(CMT)				X	
	Management Team(MT)					X
	Subscription Charge(SC)			X		
	Cinemas Staff(CST)				X	
	Person(P)					X
	Release Period(RP)			X		
	Regard(R)			X		
	Followe(F)			X		
	Information(IN)			X		
	Board(B)				X	
	Debit Card (DC)				X	
	Cinema Membership Card (CMC)					X
	I(I)			X		
	Card(CA)				x	
	Date(DA)			X		
Totals		8	5	15	13	3

Relationship Detection Data

Orig Model	Relationships	Correct	Missing	Incorrect	Extra
F→SH	F→T	X			
S→SH			X		
WSS→SH			X		
SS→SH			X		
US→SS			X		
T→SS			X		
T→US			X		
FT→T			X		
FT→T			X		
T→S			X		
S→CS			X		
CCS→S			X		
CC→CCS			X		
CC→C			X		
	D→W			X	
	P→T			X	
	ST→T				X
	CI→T				X
	M→T			X	
	TY→T			X	
	TY→N			X	
	N→T			X	
	T→S				X
	N→S			X	
	N→S			X	
	S→SS				X
	FD→SE			X	
	C→SE				X
	P→C			X	
	F→S				X
	S→F				X
	CI→F				X
	CI→CO			X	
	F→L			X	
	M→CA				X
	CA→I			X	
	CA→DA			X	
	F→IN			X	
	B→IN			X	
Totals		1	13	16	9

Measures Results

	Recall	Precision	Over-Specification	F-Measure
Classes	*0.62*	*0.36*	*0.92*	*0.46*
Relationships	*0.07*	*0.06*	*0.64*	*0.06*

KWIC [Par72]

Class Detection Data:

Original Model	Classes	Correct	Missing	Incorrect	Extra	SYM
Master Control(MC)	KWIC Index System (KIS)	X				
Input(I)	np		X			
Output(O)	np		X			
Character(C)	Word(W)	X				
Circular Shift(CS)	Shift(S)	X				
Alphabetic Shift(AS)	Shift(S)	X				
	End(E)			X		
	Line(LI)				X	
	Ordered Set (OS)					X
	Set(SE)				X	
	List(L)				X	
Totals		4	2	1	3	1

Relationship Detection Data:

Original Model	Relationships	Correct	Missing	Incorrect	Extra
C→CS			X		
CS→ASL			X		
	KIS→SE				X
	KIS→OS				X
	KIS→L				X
	L→S				X
	S→LI				X
	SE→W				X
	SE→LI				X
	LI→E			X	
Totals		0	2	1	7

Measure Results:

	Recall	Precision	Over-Specification	F-Measure
Classes	0.67	0.80	0.50	0.73
Relationships	*0.0*	*0.0*	*3.5*	*0.0*

Library Problem 1 [EP98]:

Class Detection Data

Orig Model	Classes	Correct	Missing	Incorrect	Extra	SYM
Book Title(BT)	Book (B)	X				
Borrower(BR)	Borrower(BR)	X				
Item(I)	np		X			
Loan(L)	fn		X			
Magazine Title (MT)	Magazine(M)	X				
Reservation(R)	Reservation(R)	X				
Title(T)	fn		X			
	OldBook (OB)					X
	Copy(C)			X		
	Library(L)				X	
	Employee(E)				X	
	Information(IN)			X		
	Work(W)			X		
	System(S)			X		
	Librarian(LI)				X	
Totals		4	3	4	3	1

Relationship Detection Data:

Orig Model	Classes	Correct	Missing	Incorrect	Extra
BR→R			X		
BR→L			X		
BR→I			X		
L→I			X		
I→T			X		
BT→T			X		
MT→T			X		
	B→L				X
	B→L				X
	L→E				X
	E→LI				X
	L→M				X
	M→BR				X
	B→BR				X
	B→IN			X	
	R→IN			X	
	L→IN			X	
	BR→IN			X	
	BR→S				X
	IN→S			X	
	W→S			X	
Totals		0	7	6	8

Measures Results

	Recall	Precision	Over-Specification	F-Measure
Classes	0.57	0.50	0.43	0.53
Relationships	0.0	0.0	1.14	0.0

Library Problem 2 [Cal94]:

Class Detection Data

Orig Model	Classes	Correct	Missing	Incorrect	Extra	SYM
Book (B)	Book	X				
Customer (C)	Customer	X				
Language Tape (LT)	Language Tape (LT)	X				
Library (L)	Library(L)	X				
Loan Item (LI)	Loan Item (LI)	X				
Section (S)	Section(S)	X				
Member Card (MC)	Membership Card (MC)	X				
Loan Transaction(LTR)	Np		X			
	Record (R)				X	
	Update (U)			X		
	Type (T)			X		
	Birth (B)			X		
	Membership (M)					X
	Bar Code Reader (BCR)				X	
	Number (N)			X		
	Member					X
Totals		7	1	4	2	2

Relationship Detection Data

Orig Model	Relationships	Correct	Missing	Incorrect	Extra
L→S			X		
S→LI			X		
L→MC			X		
MC→C	MC→C	X			
C→LI	LI→C	X			
C→LTR			X		
LTR→LI			X		
LI→LT			X		
LI→B			X		
	L→LI				X
	LI→T			X	
	LI→N			X	
	N→S			X	
	T→B			X	
	T→LT			X	
	R→U			X	
Totals		2	7	6	1

Measures Results

	Recall	Precision	Over-Specification	F-Measure
Classes	*0.88*	*0.64*	*0.25*	*0.74*
Relationships	*0.20*	*0.14*	*0.40*	*0.17*

Library Problem 3 [Cur95]:

Class Detection Data

Orig Model	Classes	Correct	Missing	Incorrect	Extra	SYM
Order (O)	fn		X			
Invoice (I)	Invoice (I)	X				
Book (B)	Book(B)	X				
Note (N)	fn		X			
Catalogue Note (CN)	Catalogue Note (CN)	X				
Delivery Note (DN)	Delivery Note (DN)	X				
Enquiry Note (EN)	np		X			
Person (P)	np		X			
	Enquiry (E)			X		
	Public (P)				X	
	Accounts Department (AD)				X	
	Account (A)				X	
	Store (S)			X		
	Pending File (PF)				X	
	Letter (L)				X	
	Instruction (INS)			X		
	Library Desk (LD)			X		
	Publisher (PUB)				X	
	Library(LIB)				X	
	File(F)				X	
Totals		4	4	4	8	0

Relationship Detection Data

Orig Model	Relationships	Correct	Missing	Incorrect	Extra
O→B			X		
I→B			X		
B→N			X		
B→P			X		
EN→N			X		
CN→N		X			
DN→N		X			
	LIB→B				X
	LIB→PUB				X
	B→F				X
	F→AN				X
	E→L			X	
	AD→A				X
	AD→S			X	
	S→AN			X	
	AN→I				X
	I→PF				X
	I→PUB			X	
Totals		2	5	4	7

Measures Results

	Recall	Precision	Over-Specification	F-Measure
Classes	*0.63*	*0.63*	*1.0*	*0.63*
Relationships	*0.29*	*0.33*	*1.0*	*0.31*

Filing Problem [Der95]:

Class Detection Data

Orig Model	Classes	Correct	Missing	Incorrect	Extra	SYM
Author (A)	Author (A)	X				
Keyword (K)	Keyword (K)	X				
Abstract (AB)	DocumentDescription (DD)	X				
ASCII Character (AC)	FilingCharacterSet(FCS)	X				
Junk Word (JW)	JunkWord (JW)	X				
Text Document (TD)	TextDocument (TD)	X				
Index (I)	fn		X			
Page (P)	Np		X			
Line (L)	Np		X			
Word (W)	np		X			
	SearchCriterium (SC)				X	
	SpecifiedSearchCriterium(SSC)					X
	User (U)				X	
	E.g (EG)			X		
	EFP (E)			X		
	Document (DOC)				X	
	WordProcessor (WP)			X		
	Portion (P)			X		
	Content (C)			X		
	Description (D)					X
	Editor (ED)				X	
Totals		6	4	5	4	2

Relationship Detection Data

Orig Model	Relationships	Correct	Missing	Incorrect	Extra
TD→A			X		
TD→K			X		
TD→P			X		
P→L			X		
L→W			X		
L→JW			X		
W→AC			X		
TD→AB		X			
I→TD			X		
	U→JW				X
	D→DD				X
	D→WP			X	
	D→E				X
	D→P			X	
	D→C			X	
Totals		1	8	3	3

Measures Results

	Recall	Precision	Over-Specification	F-Measure
Classes	0.6	0.54	0.4	0.57
Relationships	0.11	0.25	0.33	0.15

Exam Problem:

Class Detection Data

Orig Model	Classes	Correct	Missing	Incorrect	Extra	SYM
Faculty Member(FM)	Faculty Member(FM)	X				
Course(C)	Course(C)	X				
Student(S)	Student(S)	X				
Exam(E)	Exam(E)	X				
Item(I)	np		X			
Choice(CH)	Entry(EN)	X				
Scan Tron Sheet(STS)	Bubble Sheet(BS)	X				
Sheet Reader(SR)	NP(auto processing)		X			
ASCII File(AF)	Data File(DF)	X				
Analyzer(A)	np		X			
Part Group (PG)	np		X			
Part(P)	Part(P)(STWR)	X				
Record(R)	Record(R)	X				
Letter(L)	Letter(L)	X				
Question(Q)	Question(Q)	X				
Answer(ANS)	Answer(A)	X				
Repository(RE)	Repository(R)	X				
Answers(ANSS)	Answer Sheet(AS)	X				
Blank(B)	Space(SP)	X				
A-Z(AZ)	Requested Answer(RA)	X				
*(*)	FN(asterisk Character)		X			
	Custom Exam(CE)				X	
	Duplication(D)			X		
	Set(SE)					X
	Version(V)				X	
	Exam Question (EQ)					X
	System(SY)				X	
	Place(PL)			X		
	Bubble(B)			X		
	Name			X		
	Output			X		
	University				X	
	Course Content				X	
	Credit				X	
	Student Performance(SPE)				X	
Totals		16	5	5	7	2

Relationship Detection Data:

Orig Rel	ASA Rel	Correct	Missing	Incorrect	Extra
FM→E	FM→E	X			
FM→C			X		
C→S			X		
E→I			X		
STS→CH			X		
SR→STS			X		
SR→AF			X		
AF→A			X		
A→RE			X		
AF→R			X		
R→L			X		
L→B			X		
L→*			X		
L→AZ			X		
I→P			X		
I→PG			X		
PG→P			X		
P→Q			X		
P→A			X		
Q→ANS	Q→ANS	X			
A→ANSS			X		
Q→ANSS			X		
ANSS→RE			X		
A→RE			X		
	E→D			X	
	E→SE				X
	E→V				X
	V→Q			X	
	Q→ANS				X
	Q→SPE			X	
	Q→BS				X
	ANS→S			X	
	S→R				X
	S→L			X	
	L→PL			X	
	PL→SY			X	
	PL→SP			X	
	S→SP			X	
	S→B			X	
	B→E			X	
Totals		2	22	12	5

Measures Results

	Recall	Precision	Over-Specification	F-Measure
Classes	*0.76*	*0.76*	*0.38*	*0.76*
Relationships	*0.08*	*0.14*	*0.21*	*0.11*

Personnel Problem [Duf95]:

Class Detection Data

Orig Model	Classes	Correct	Missing	Incorrect	Extra	SYM
Reader (R)	Reader (R)	X				
Journal (J)	Journal (J)	X				
Topic (T)	Topic (T)	X				
Article (A)	Article (A)	X				
Issue (I)	M(np)		X			
	System (S)			X		
	Publication (P)				X	
	Access (AC)				X	
	Research Institute (RA)				X	
	Permission (PE)					X
	Personnel Department (PD)				X	
	Department (D)				X	
	Week (W)			X		
	Period (PER)			X		
	Frequency (F)			X		
	Language (L)			X		
	Publisher (PUB)				X	
Totals		4	1	5	6	1

Relationship Detection Data

Orig Model	Relationships	Correct	Missing	Incorrect	Extra
R→J	R→J	X			
R→T			X		
T→A			X		
A→I			X		
J→I			X		
	RI→PD				X
	PE→D				X
	R→A				X
	R→AC				X
	J→W			X	
	J→PER			X	
	J→F			X	
	J→L			X	
Totals		1	4	4	4

Measures Results

	Recall	Precision	Over-Specification	F-Measure
Classes	0.80	0.44	1.2	0.57
Relationships	0.20	0.20	0.8	0.20

Hospital Problem 1 [Cur95]:

Class Detection Data

Orig Model	Classes	Correct	Missing	Incorrect	Extra	SYM
Patient(P)	Patient(P)	X				
Ward(W)	Ward(W)	X				
Room(R)	Room(R)	X				
Consultant(C)	Consultant(C)	X				
Surgeon(S)	Surgeon(S)	X				
Nurse(N)	Nurse(N)	X				
Normal Patient(NP)	Patient(P)	X				
Private Patient(PP)	Patient(P)	X	X			
Staff(ST)	np		X			
	Theatre(T)				X	
	Operating Theatre (OT)					X
	Given Theatre (GT)					X
	Venue(V)			X		
	Number(NU)			X		
	Type(T)			X		
	Birth(B)			X		
	Date(D)			X		
	Hospital(H)				X	
	EDP Department(EDP)				X	
	Name(NA)			X		
	Address(A)			X		
Totals		8	1	7	3	2

Relationship Detection Data

Orig Model	Relationships	Correct	Missing	Incorrect	Extra
NP→W			X		
PP→R		X			
ST→SU			X		
ST→N			X		
S→C		X			
N→W		X			
C→R			X		
	N→W				X
	H→EDP				X
	T→N				X
	T→N				X
	T→V			X	
	P→NU			X	
	P→C				X
Totals		3	4	2	5

Measures Results

	Recall	Precision	Over-Specification	F-Measure
Classes	0.89	0.53	0.33	0.67
Relationships	0.43	0.6	0.71	0.50

Hospital Problem 2 [Duf95]:

Class Detection Data

Orig Model	**Classes**	**Correct**	**Missing**	**Incorrect**	**Extra**	**SYM**
Patient(P)	Patient(P)	X				
Ward(W)	Ward(W)	X				
Doctor(D)	Doctor(D)	X				
Nurse(N)	Nurse(N)	X				
Prescription(PRE)			X			
	Drug(DR)				X	
	Hospital(H)				X	
	Advisor(A)				X	
Totals		4	1	0	3	0

Relationship Detection Data

Orig Model	**Relationships**	**Correct**	**Missing**	**Incorrect**	**Extra**
W→D			X		
W→N		X			
N→P		X			
D→P		X			
P→PRE			X		
	H→W				X
	W→P				X
	N→P				X
	P→DR				X
Totals		3	2	0	4

Measures Results

	Recall	Precision	Over-Specification	F-Measure
Classes	0.80	1.0	0.60	0.89
Relationships	0.60	1.0	0.80	0.75

Lift Problem [PRM+07]:

Class Detection Data

Orig Model	Classes	Correct	Missing	Incorrect	Extra	SYM
Down Button(DB)	Button(B)	X				
Lift Button(LB)	Button(B)	X				
Up Button(UB)	Button(B)	X				
Floor Panel UI(FPU)	np		X			
Floor Number Button(FNB)	Button(B)	X				
Lift Controller(LC)	Lift Controller(LC)	X				
Motor(M)	Motor(M)	X				
Lift Panel UI(LPU)	np		X			
Lift(L)	Lift(L)	X				
Door(D)	Door(D)	X				
	Passenger(P)				X	
	PassengerInteract(PI)			X		
	Build(BU)				X	
	Period(P)			X		
	Second(S)			X		
	PeriodSecond(PS)			X		
	ControlPanel(CP)					X
	Pres(PR)			X		
	Floor(F)				X	
Totals		8	2	5	3	1

Relationship Detection Data:

Orig Rel	ASA Rel	Correct	Missing	Incorrect	Extra
DB→LB			X		
UB→LB			X		
FNB→LB			X		
FPU→DB			X		
FPU→UB			X		
FNB→LPU			X		
FNB→LC			X		
FPU→LC			X		
UB→LC			X		
DB→LC			X		
LC→M			X		
LPU→LC			X		
LC→L	LC→L	X			
M→L	M→L	X			
LC→D			X		
LC→D			X		
L→D			X		
	B→P				X
	P→PR			X	
	P→F				X
	P→L				X
	P→L				X
	PR→F			X	
	L→F				X
	L→F				X
	L→CP				X
	L→D				X
	F→D			X	
	F→BU				X
	F→P			X	
	S→F			X	
	PS→F			X	
Totals		2	15	6	9

Measures Results

	Recall	Precision	Over-Specification	F-Measure
Classes	*0.90*	*0.75*	*0.20*	*0.82*
Relationships	*0.12*	*0.25*	*0.53*	*0.16*

Meeting Problem [RBPEL91]:

Class Detection Data

Orig Model	Classes	Correct	Missing	Incorrect	Extra	SYM
Competition(C)	fn		X			
Competitor(CO)	Competitor(CO)	X				
Figure(F)	fn		X			
Judge(J)	Judge (J)	X				
League(L)	League(L)	X				
Meeting(M)	Meet(M)	X				
Score(S)	fn		X			
Season(S)	Season(S)	X				
Station(ST)	Station(ST)	X				
Team(T)	Team(T)	X				
Trial(TR)	fn		X			
	Part(P) (STWR)			X		
	Place(PL)			X		
	Month(MO)			X		
	Group(G)				X	
	Figures Correspond(FC)			X		
	Softcom(SC)				X	
	Performance(P)			X		
	Final Prize(FP)				X	
	Work(W)			X		
	Panel(PA)			X		
Totals		7	4	7	3	0

Measures Results

	Recall	Precision	Over-Specification	F-Measure
Classes	0.64	0.50	0.36	0.56

Taxi Problem [BAR12]:

Class Detection Data:

Orig Model	Classes	Correct	Missing	Incorrect	Extra	SYM
Passenger Source (PS)	np		X			
Taxi Company (TC)		X				
Vehicle (V)		X				
Shuttle (S)		X				
Taxi (T)		X				
Passenger (P)		X				
Location (L)		X				
	Call (C)				X	
	Driver (D)				X	
Totals		6	1	0	2	0

Relationship Detection Data:

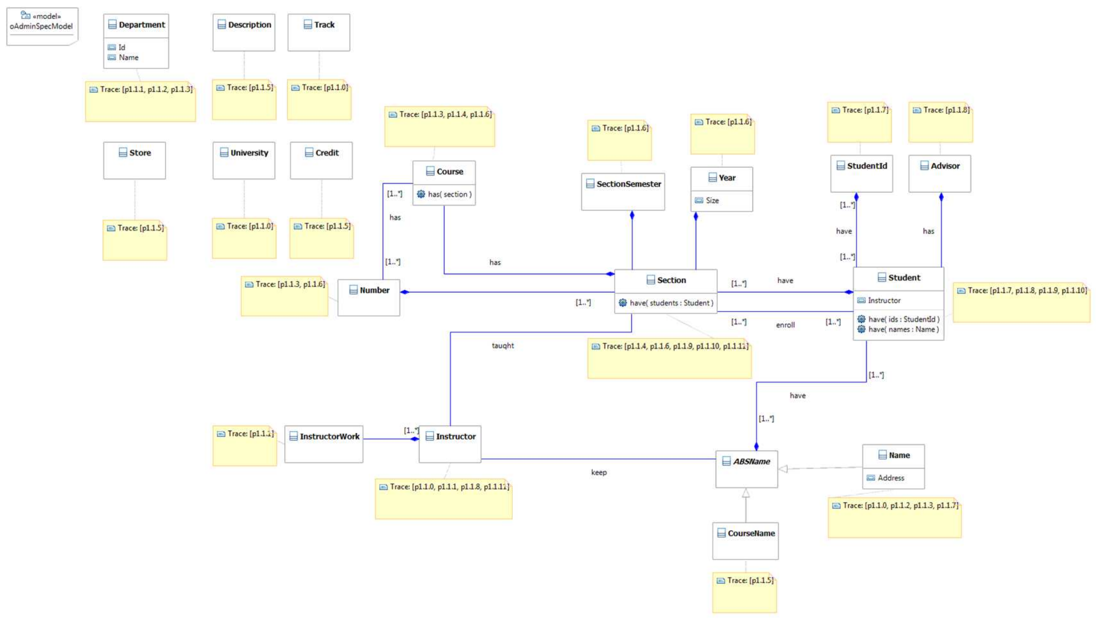
Orig Rel	ASA Rel	Correct	Missing	Incorrect	Extra
PS→TC			X		
TC→V		X			
TC→V			X		
V→L		X			
V→L			X		
P→L		X			
P→L		X			
S→P		X			
S→L			X		
	S→TC				X
	T→P				X
	T→TC				X
	D→TC				X
	C→TC				X
	P→TC				X
Totals		5	4	0	6

Measures Results

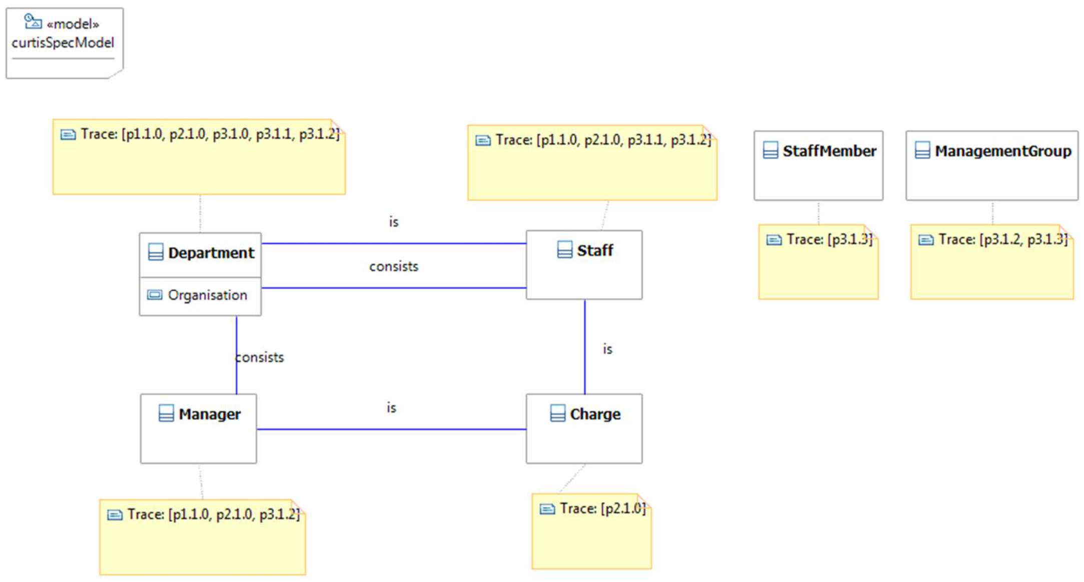
	Recall	Precision	Over-Specification	F-Measure
Classes	*0.86*	*1.0*	*0.29*	*0.92*
Relationships	*0.55*	*1.0*	*0.67*	*0.71*

Appendix B.3 UML Models Generated by the ASA

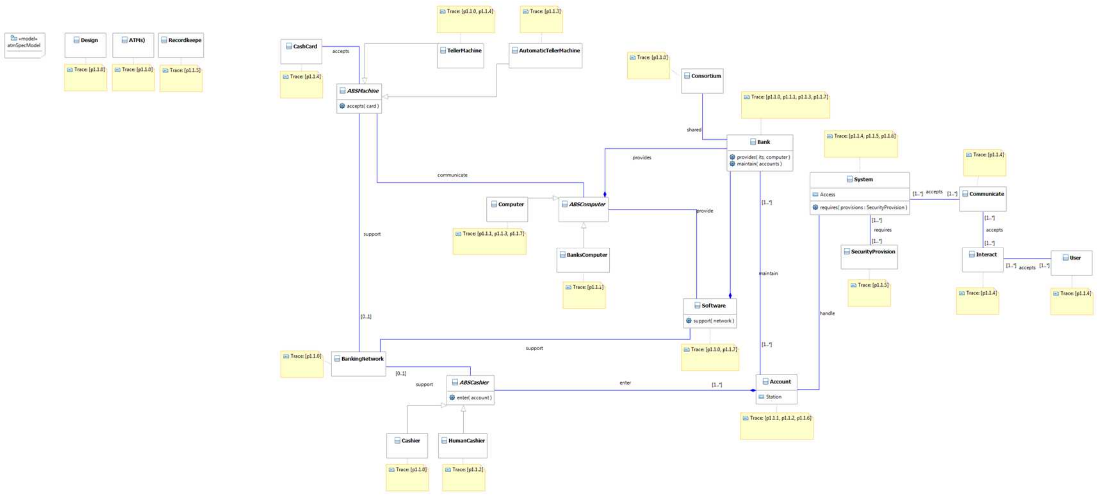
Organisation Problem 1:



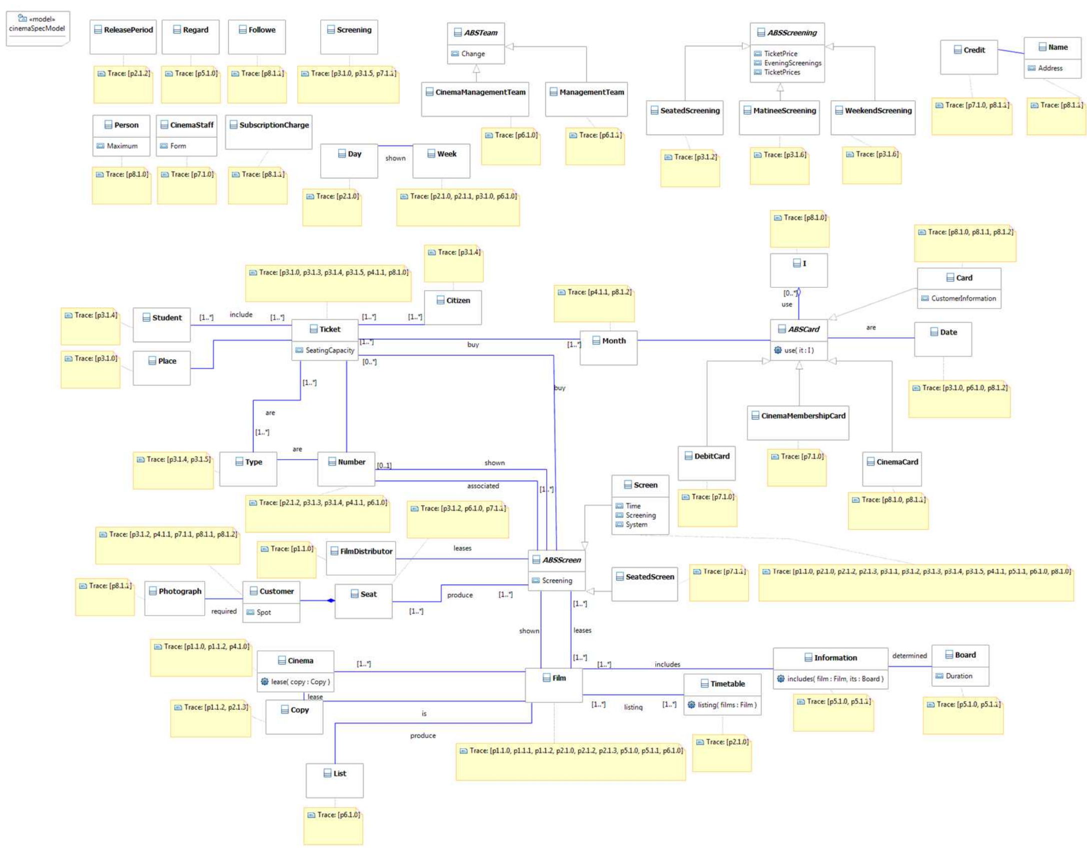
Organisation Problem 2 [Cur95]:



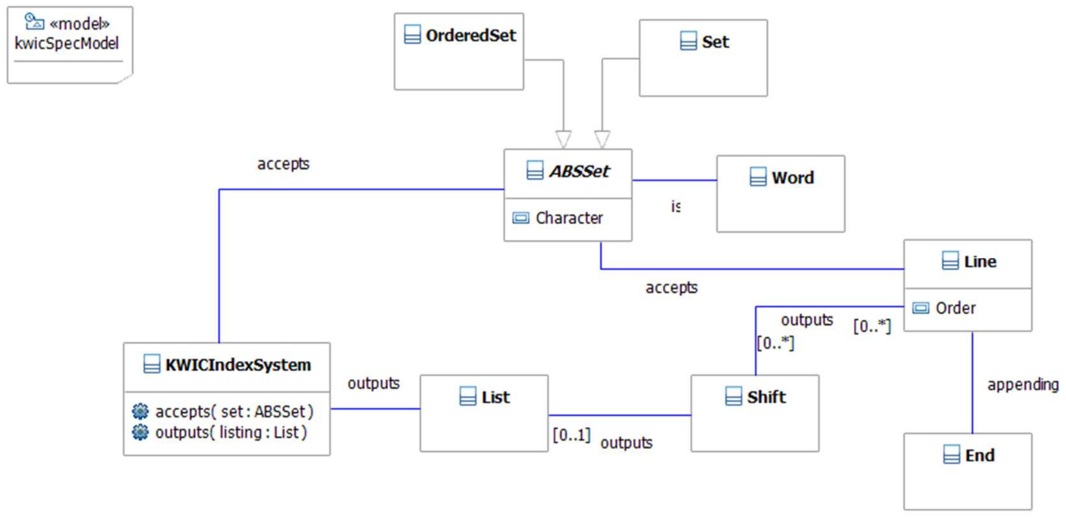
ATM Problem [RBPEL91]:



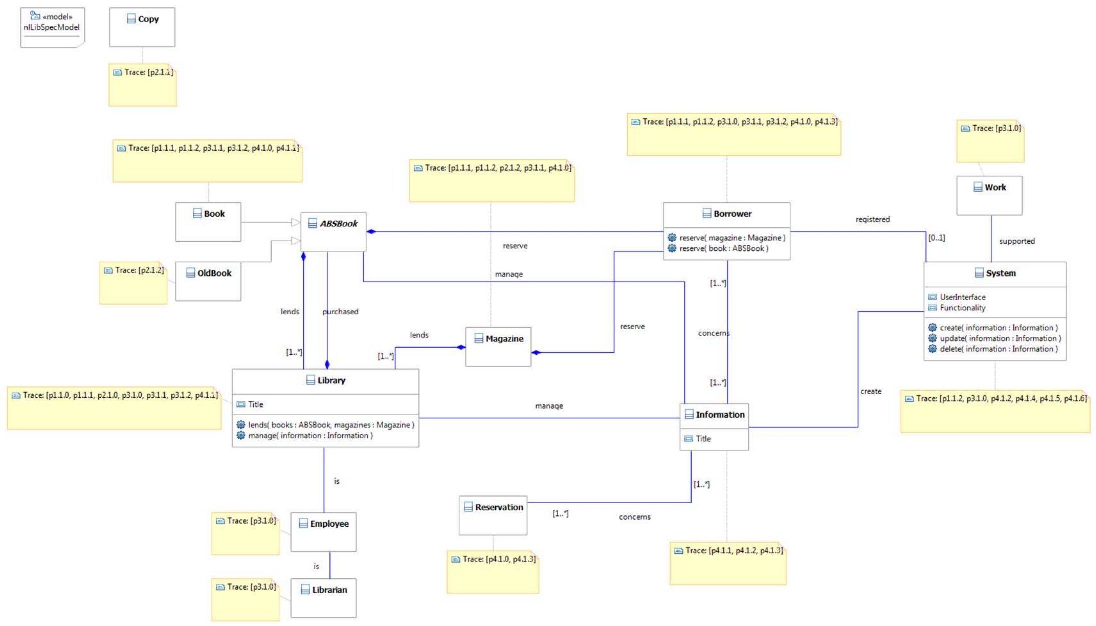
Cinema Problem [CIS08]:



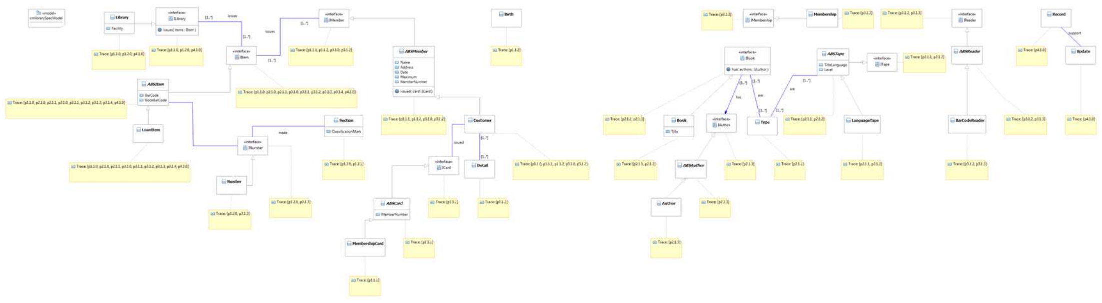
KWIC [Par72]



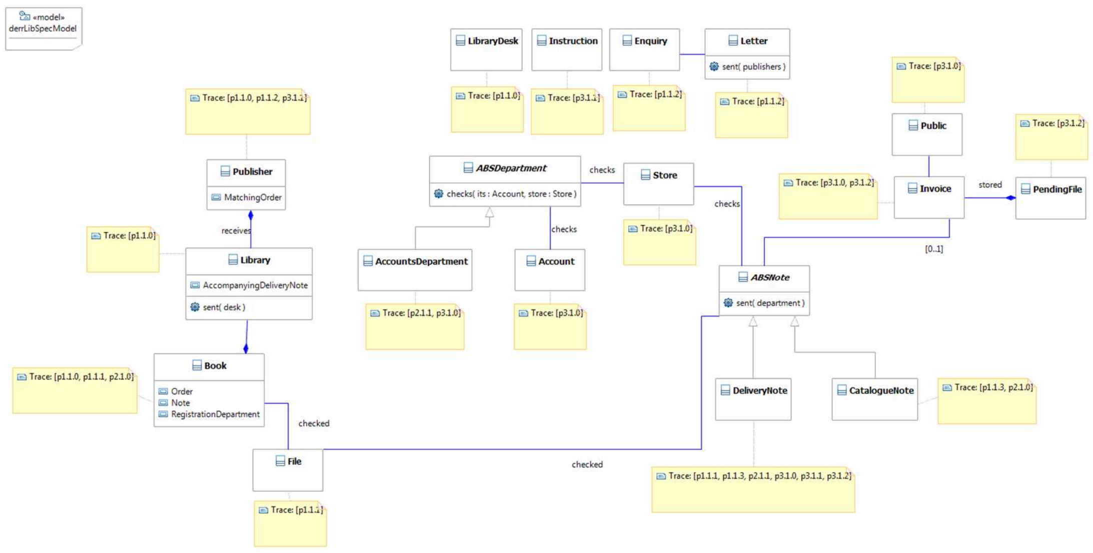
Library Problem 1 [EP98]:


«model»
nlLibSpecModel
Copy
Trace: [p2.1.1]
Trace: [p1.1.1, p1.1.2, p3.1.1, p3.1.2, p4.1.0, p4.1.1]
Trace: [p1.1.1, p1.1.2, p2.1.2, p3.1.1, p4.1.0]
Trace: [p1.1.1, p1.1.2, p3.1.0, p3.1.1, p3.1.2, p4.1.0, p4.1.3]
Trace: [p3.1.0]
Work
Book
ABSBook
Borrower
reserve(magazine : Magazine)
reserve(book : ABSBook)
registered
[0..1]
supported
reserve
Trace: [p2.1.2]
OldBook
manage
System
UserInterface
Functionality
create(information : Information)
update(information : Information)
delete(information : Information)
[1..*]
lends
purchased
lends
reserve
concerns
[1..*]
[1..*]
Magazine
[1..*]
Library
Title
lends(books : ABSBook, magazines : Magazine)
manage(information : Information)
Trace: [p1.1.0, p1.1.1, p2.1.0, p3.1.0, p3.1.1, p3.1.2, p4.1.1]
manage
create
Trace: [p1.1.2, p3.1.0, p4.1.2, p4.1.4, p4.1.5, p4.1.6]
Information
Title
is
[1..*]
Reservation
[1..*]
concerns
Trace: [p3.1.0]
Employee
is
Trace: [p4.1.0, p4.1.3]
Trace: [p4.1.1, p4.1.2, p4.1.3]
Trace: [p3.1.0]
Librarian

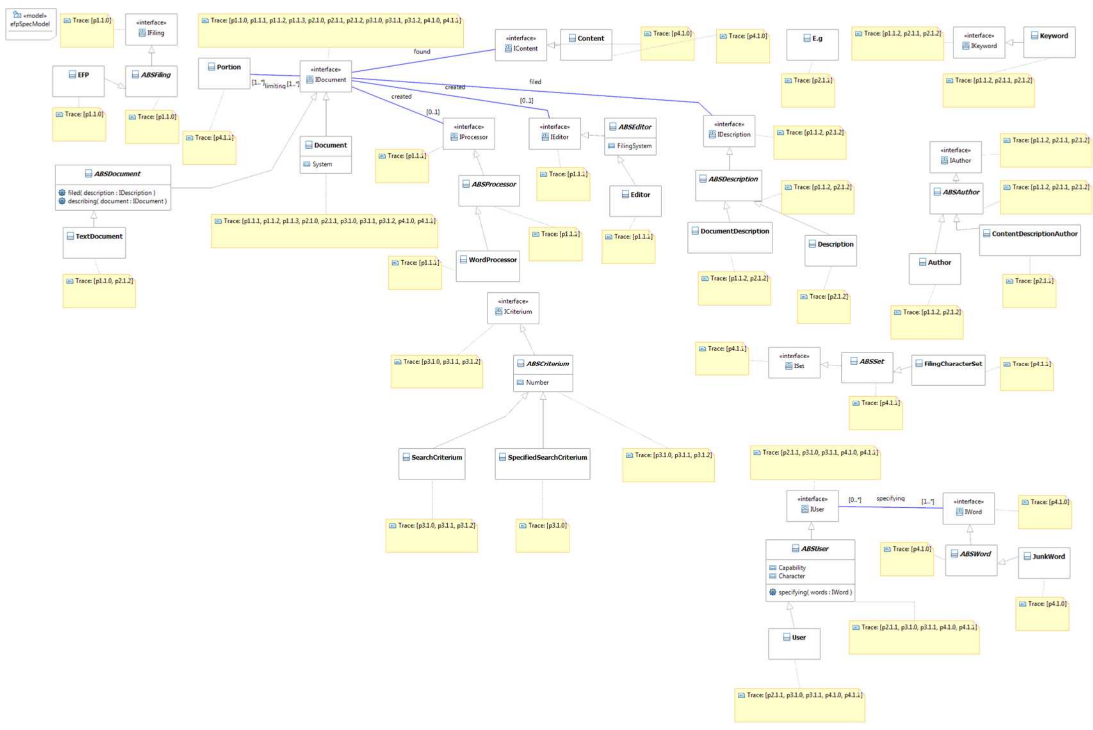
Library Problem 2 [Cal94]:



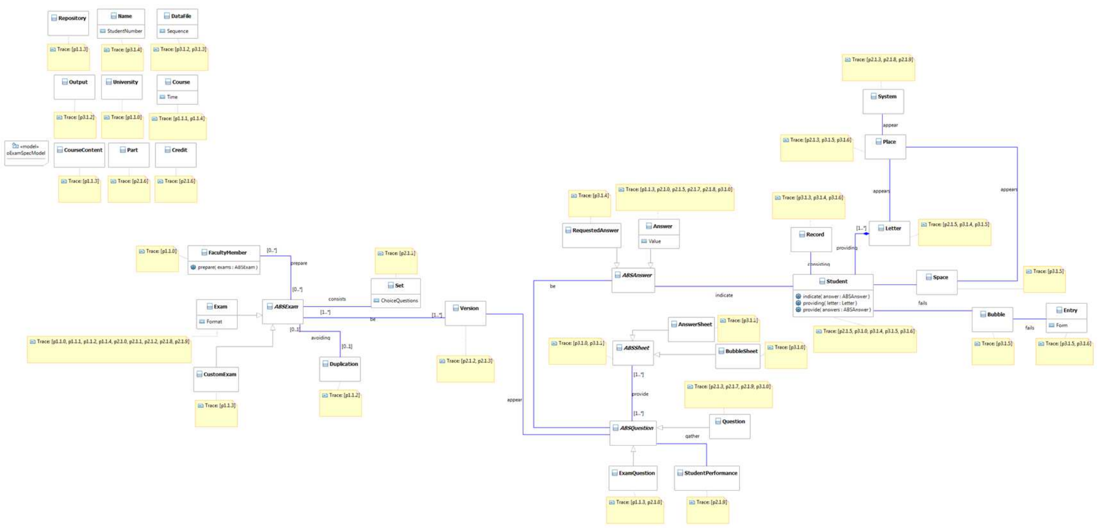
Library Problem 3 [Cur95]:



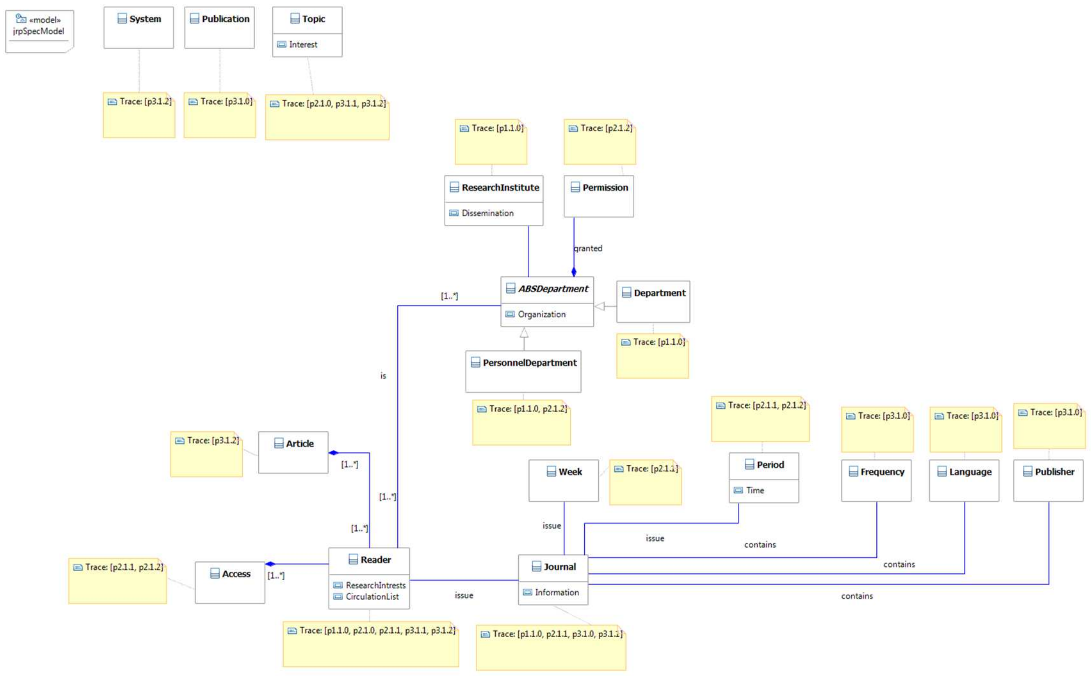
Filing Problem [Der95]:



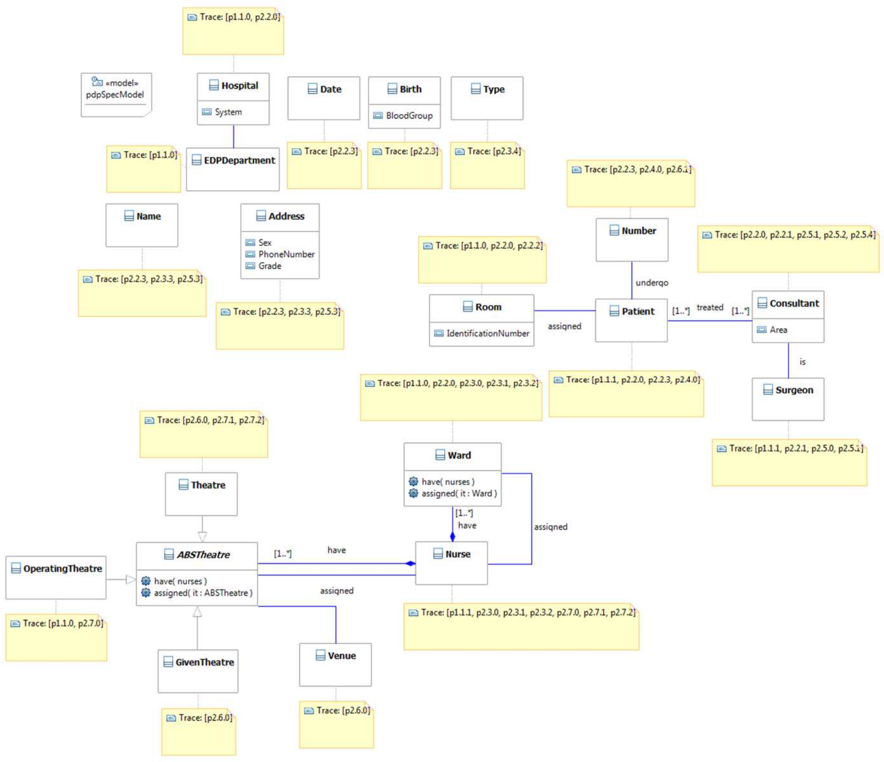
Exam Problem:



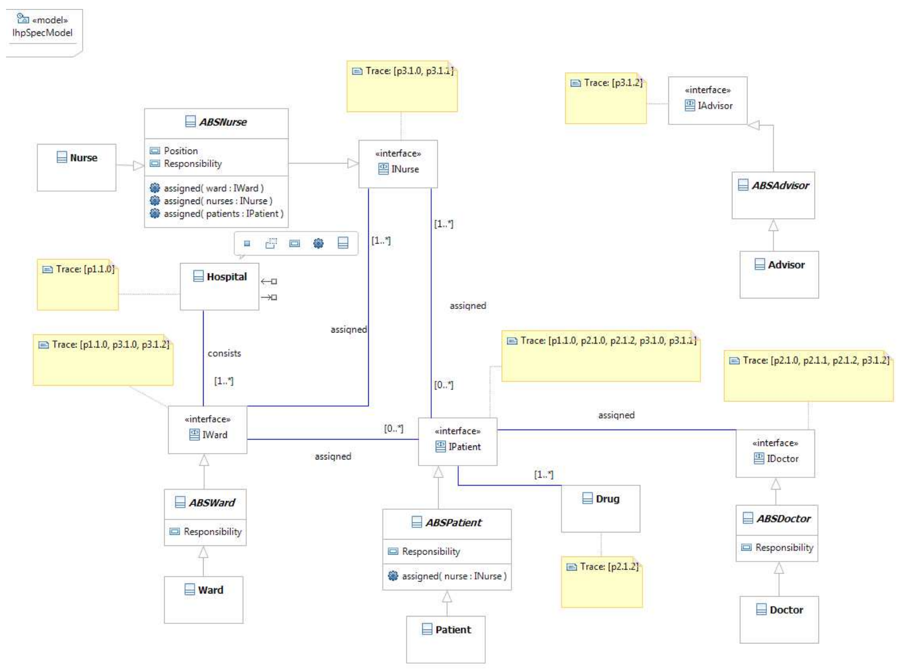
Personnel Problem [Duf95]:



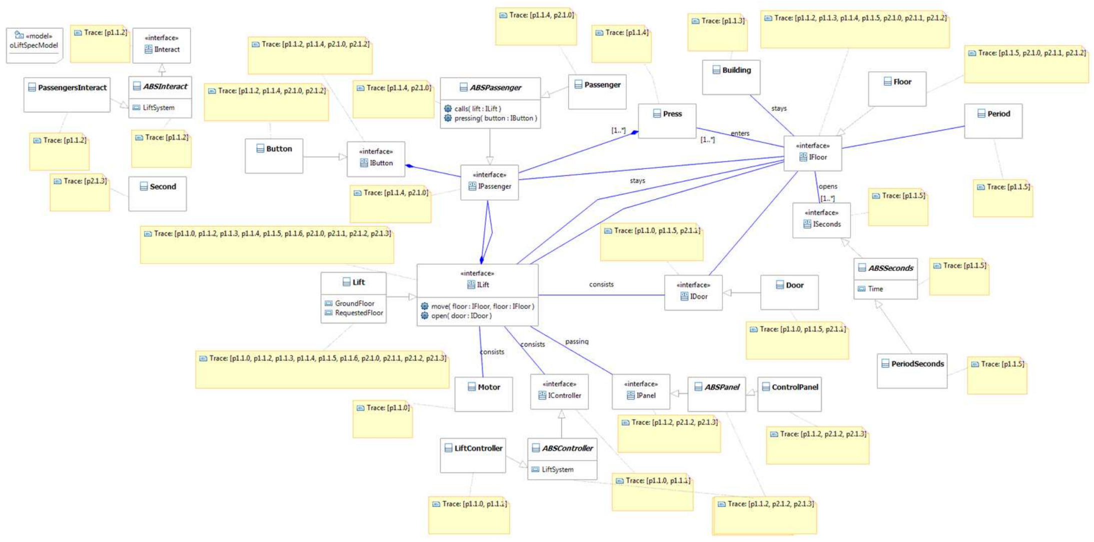
Hospital Problem 1 [Cur95]:



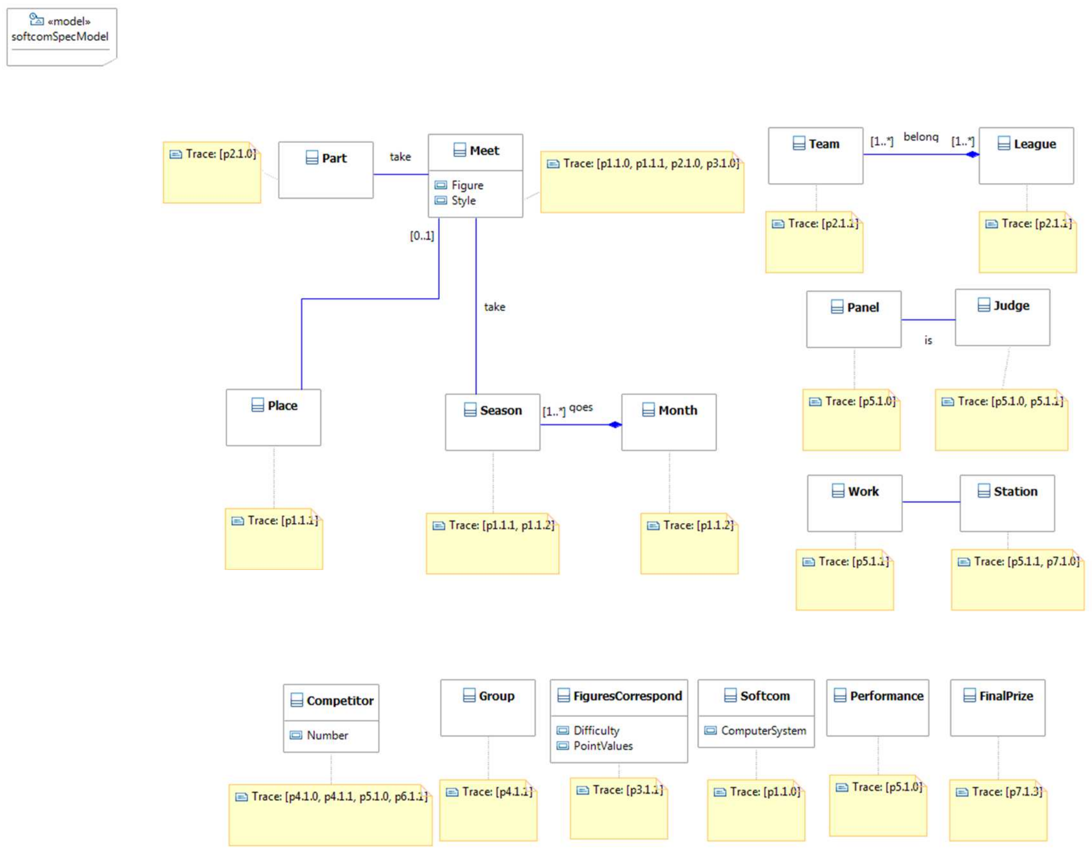
Hospital Problem 2 [Duf95]:



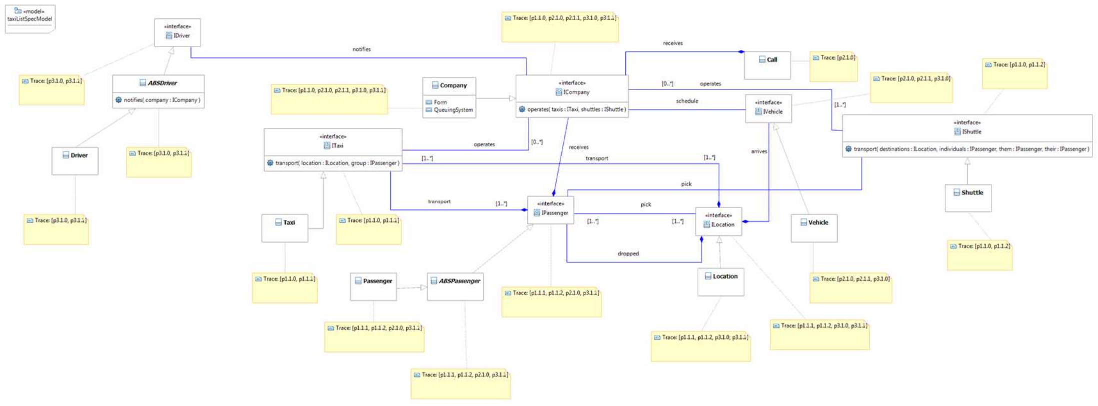
Lift Problem [PRM+07]:



Meeting Problem [RBPEL91]:



Taxi Problem [BAR12]:


«model»
taxiListSpecModel
«interface»
IDriver
notifies
Trace: [p1.1.0, p2.1.0, p2.1.1, p3.1.0, p3.1.1]
receives
Call
Trace: [p2.1.0]
Trace: [p1.1.0, p1.1.2]
Trace: [p3.1.0, p3.1.1]
ABSDriver
notifies(company : ICompany)
Trace: [p1.1.0, p2.1.0, p2.1.1, p3.1.0, p3.1.1]
Company
Form
QueuingSystem
«interface»
ICompany
operates(taxis : ITaxi, shuttles : IShuttle)
[0..*]
operates
schedule
Trace: [p2.1.0, p2.1.1, p3.1.0]
«interface»
IVehicle
[1..*]
«interface»
IShuttle
transport(destinations : ILocation, individuals : IPassenger, them : IPassenger, their : IPassenger)
«interface»
ITaxi
transport(location : ILocation, group : IPassenger)
operates
[0..*]
receives
transport
[1..*]
[1..*]
arrives
Driver
Trace: [p3.1.0, p3.1.1]
pick
transport
[1..*]
«interface»
IPassenger
pick
[1..*]
[1..*]
«interface»
ILocation
Shuttle
Trace: [p3.1.0, p3.1.1]
Taxi
Trace: [p1.1.0, p1.1.1]
Vehicle
Trace: [p1.1.0, p1.1.2]
dropped
Location
Trace: [p1.1.0, p1.1.1]
Passenger
ABSPassenger
Trace: [p1.1.1, p1.1.2, p2.1.0, p3.1.1]
Trace: [p2.1.0, p2.1.1, p3.1.0]
Trace: [p1.1.1, p1.1.2, p2.1.0, p3.1.1]
Trace: [p1.1.1, p1.1.2, p3.1.0, p3.1.1]
Trace: [p1.1.1, p1.1.2, p3.1.0, p3.1.1]
Trace: [p1.1.1, p1.1.2, p2.1.0, p3.1.1]

Appendix B.4 Class Candidate Key Issues Analysis Raw Data

Classification	Word	Core Issue
False Negative (P)	character	Rule 27
False Negative (P)	family ticket	Rule 27
False Negative (P)	character	Rule 27
False Negative (P)	station	Rule 27
False Negative (P)	figure	Rule 27
False Negative (P)	note	Rule 27
False Negative (P)	order	Rule 27
False Negative (P)	station	Rule 27
False Negative (P)	automatic processing(sheet reader)	Semantics Issue
False Negative (P)	sale	Semantics Issue
False Negative (P)	transaction(remote)	Semantics Issue
False Negative (P)	transaction(cashier)	Semantics Issue
False Negative (P)	loan	Semantics Issue
False Negative (P)	competition	Semantics Issue
False Negative (P)	score	Semantics Issue
False Negative (P)	trial	Semantics Issue
False Negative (P)	index	Semantics Issue
False Negative (P)	project	Semantics Issue
False Negative (P)	flipper left	Rule 27
False Negative (P)	unseated showing	Semantics Issue
False Negative (P)	right flipper	Semantics Issue
False Negative (P)	circle bumper	Semantics Issue
False Negative (P)	output	nlp
False Negative (P)	title	Semantics Issue
False Positive	number	Rule 27
False Positive	type	Rule 27
False Positive	feature	Rule 27
False Positive	game	Rule 27
False Positive	type	Rule 27
False Positive	way	Rule 27
False Positive	press	Rule 27
False Positive	Part	Rule 27
False Positive	portion	Rule 27
False Positive	number	Rule 27
False Positive	charge	Rule 27
False Positive	interact	nlp
False Positive	interact	nlp
False Positive	bubble	Semantics Issue
False Positive	duplication	Semantics Issue
False Positive	name	Semantics Issue
False Positive	output	Semantics Issue
False Positive	place	Semantics Issue
False Positive	copy	Semantics Issue
False Positive	date	Semantics Issue
False Positive	day	Semantics Issue
False Positive	following	Semantics Issue
False Positive	i .e	Semantics Issue
False Positive	information	Semantics Issue
False Positive	month	Semantics Issue
False Positive	photograph	Semantics Issue
False Positive	place	Semantics Issue
False Positive	release period	Semantics Issue
False Positive	subscription charge	Semantics Issue
False Positive	week	Semantics Issue
False Positive	bottom	Semantics Issue
False Positive	center	Semantics Issue
False Positive	chain	Semantics Issue
False Positive	clockwise direction	Semantics Issue
False Positive	corner	Semantics Issue
False Positive	direction	Semantics Issue

False Positive	end	Semantics Issue	False Positive	place	Semantics Issue
False Positive	execution building	Semantics Issue	False Positive	working	Semantics Issue
False Positive	frame	Semantics Issue	False Positive	content	Semantics Issue
False Positive	frame redraw	Semantics Issue	False Positive	word processor	Semantics Issue
False Positive	graphic community	Semantics Issue	False Positive	enquiry	Semantics Issue
False Positive	hypotenuse	Semantics Issue	False Positive	instruction	Semantics Issue
False Positive	length side	Semantics Issue	False Positive	library desk	Semantics Issue
False Positive	platform	Semantics Issue	False Positive	store	Semantics Issue
False Positive	second	Semantics Issue	False Positive	frequency	Semantics Issue
False Positive	surface	Semantics Issue	False Positive	language	Semantics Issue
False Positive	switch	Semantics Issue	False Positive	period	Semantics Issue
False Positive	course name	Semantics Issue	False Positive	system	Semantics Issue
False Positive	description	Semantics Issue	False Positive	week	Semantics Issue
False Positive	instructor work	Semantics Issue	False Positive	address	Semantics Issue
False Positive	name	Semantics Issue	False Positive	birth	Semantics Issue
False Positive	number	Semantics Issue	False Positive	date	Semantics Issue
False Positive	section semester	Semantics Issue	False Positive	name	Semantics Issue
False Positive	store	Semantics Issue	False Positive	number	Semantics Issue
False Positive	student id	Semantics Issue	False Positive	type	Semantics Issue
False Positive	track	Semantics Issue	False Positive	venue	Semantics Issue
False Positive	year	Semantics Issue	False Positive	birth	Semantics Issue
False Positive	end	Semantics Issue	False Positive	type	Semantics Issue
False Positive	m second	Semantics Issue	False Positive	update	Semantics Issue
False Positive	period m second	Semantics Issue	False Positive	regarding	nlp
False Positive	communicate	Semantics Issue	False Positive	coarser	nlp
False Positive	software	Semantics Issue	False Positive	continually	nlp
False Positive	system	Semantics Issue	False Positive	corresponding	nlp
False Positive	copy	Semantics Issue	False Positive	counter-clockwise	nlp
False Positive	information	Semantics Issue	False Positive	l/sec stationary	nlp
False Positive	system	Semantics Issue	False Positive	location furthest	nlp
False Positive	work	Semantics Issue	False Positive	passenger interaction	Semantics Issue
False Positive	month	Semantics Issue	False Positive	design	Semantics Issue
False Positive	panel	Semantics Issue	False Positive	recordkeeping	Semantics Issue
False Positive	performance	Semantics Issue	False Positive	figure correspond	nlp

False Positive	e.g	nlp	False Negative (NP)	input	domain knowledge
False Negative (NP)	analyzer	domain knowledge	False Negative (NP)	floor pane UI	domain knowledge
False Negative (NP)	item	domain knowledge	False Negative (NP)	lift panel UI	domain knowledge
False Negative (NP)	part group	domain knowledge	False Negative (NP)	passenger source	domain knowledge
False Negative (NP)	cinema card sale	domain knowledge	False Negative (NP)	item	domain knowledge
False Negative (NP)	card sale	domain knowledge	False Negative (NP)	page	domain knowledge
False Negative (NP)	application window	domain knowledge	False Negative (NP)	line	domain knowledge
False Negative (NP)	application listener	domain knowledge	False Negative (NP)	enquiry note	domain knowledge
False Negative (NP)	game window	domain knowledge	False Negative (NP)	person	domain knowledge
False Negative (NP)	play listener	domain knowledge	False Negative (NP)	issue	domain knowledge
False Negative (NP)	build listener	domain knowledge	False Negative (NP)	staff	domain knowledge
False Negative (NP)	controller	domain knowledge	False Negative (NP)	loan transaction	domain knowledge
False Negative (NP)	object handler	domain knowledge	False Negative (NP)	prescription	domain knowledge
False Negative (NP)	movable	domain knowledge	False Negative (NP)	build window	domain knowledge
False Negative (NP)	game object	domain knowledge	False Negative (NP)	game mode	domain knowledge
False Negative (NP)	saveable	domain knowledge	False Negative (NP)	word	domain knowledge
False Negative (NP)	collideable	domain knowledge			
False Negative (NP)	visible	domain knowledge			
False Negative (NP)	triggerable	domain knowledge			
False Negative (NP)	still line	domain knowledge			
False Negative (NP)	still circle	domain knowledge			
False Negative (NP)	moving circle	domain knowledge			
False Negative (NP)	rotating edge	domain knowledge			

Appendix B.5 Missing Relationship Classification Raw Data

Relationship	**Classification**
BR→L	FN - Class not present
BT→T	FN - Class not present
MT→T	FN - Class not present
ATM→RT	FN - Class not present
BC→CS	FN - Class not present
B→CS	FN - Class not present
RT→A	FN - Class not present
RT→CC	FN - Class not present
CS→CT	FN - Class not present
A→CT	FN - Class not present
I→TD	FN - Class not present
O→B	FN - Class not present
B→N	FN - Class not present
P→S	FN - Class not present
D→P	FN - Class not present
L→*	FN - Class not present
L→AZ	FN - Class not present
WSS→SH	FN - Class not present
US→SS	FN - Class not present
T→US	FN - Class not present
FT→T	FN - Class not present
FT→T	FN - Class not present
T→S	FN - Class not present
S→CS	FN - Class not present
B→A	Class Present Relation across Paragraphs
A→CC	Class Present Relation across Paragraphs
A→CU	Class Present Relation across Paragraphs
NP→W	Class Present Relation across Paragraphs
L→S	Class Present Relation across Paragraphs
S→LI	Class Present Relation across Paragraphs
L→MC	Class Present Relation across Paragraphs
FM→C	Class Present Relation across Paragraphs
C→S	Class Present Relation across Paragraphs
Q→ANSS	Class Present Relation across Paragraphs
ANSS→RE	Class Present Relation across Paragraphs
CS→ASL	Class Present Relation across Paragraphs
SS→SH	Class Present Relation across Paragraphs
T→SS	Class Present Relation across Paragraphs
BR→R	Class Present Relation in same sentence
CO→CCO	Class Present Relation in same sentence
CCO→BC	Class Present Relation in same sentence
CC→CU	Class Present Relation in same sentence
C→R	Class Present Relation in same sentence
W→D	Class Present Relation in same sentence
R→T	Class Present Relation in same sentence
T→A	Class Present Relation in same sentence
TD→K	Class Present Relation in same sentence
LC→M	Class Present Relation in same sentence
LC→D	Class Present Relation in same sentence
AF→R	Class Present Relation in same sentence
R→L	Class Present Relation in same sentence
L→B	Class Present Relation in same sentence
P→Q	Class Present Relation in same sentence
D→I	Class Present Relation in same sentence
C→D	Class Present Relation in same sentence
C→CS	Class Present Relation in same sentence
S→SH	Class Present Relation in same sentence
S→L	Class Present Relation in same sentence
LC→D	domain knowledge - double relationship
TC→V	domain knowledge - double relationship
V→L	domain knowledge - double relationship
LI→LT	domain understanding
LI→B	domain understanding
DB→LB	domain understanding
UB→LB	domain understanding
UB→LC	domain understanding
DB→LC	domain understanding
STS→CH	domain understanding
BR→I	FN - class not present
L→I	FN - class not present
I→T	FN - class not present
ST→SU	FN - class not present
ST→N	FN - class not present
P→PRE	FN - class not present
A→I	FN - class not present
J→I	FN - class not present
TD→P	FN - class not present
P→L	FN - class not present
L→W	FN - class not present
L→JW	FN - class not present
W→AC	FN - class not present

I→B	FN - class not present
B→P	FN - class not present
EN→N	FN - class not present
C→LTR	FN - class not present
LTR→LI	FN - class not present
FNB→LB	FN - class not present
FPU→DB	FN - class not present
FPU→UB	FN - class not present
FNB→LPU	FN - class not present
FNB→LC	FN - class not present
FPU→LC	FN - class not present
LPU→LC	FN - class not present
E→I	FN - class not present
SR→STS	FN - class not present
SR→AF	FN - class not present
AF→A	FN - class not present
A→RE	FN - class not present
I→P	FN - class not present
I→PG	FN - class not present
PG→P	FN - class not present
P→A	FN - class not present
A→ANSS	FN - class not present
CCS→S	FN - class not present
CC→CCS	FN - class not present
CC→C	FN - class not present
PS→TC	FN - class not present
A→RE	FN - class not present

Appendix B.6 Training Specifications

Airport Specification:

All aircraft must have a transponder. The transponder is used to transmit aircraft position to the ground station monitor. The monitor can query an aircraft for information. The monitor keeps a database that maintains this information. A graphics display is generated from the current information. The ground station monitor updates the graphics display frequently. A graphics display is generated from the current information. The ground station monitor updates the graphics display frequently. The monitor checks for dangerous situations. The controllers may query the monitor for additional flight information. Controllers may also query the aircraft for this information.</paragraph>

Music Store Specification:

The musical store receives tape requests from customers. The musical store receives new tapes from the Main office. Musical store sends overdue notice to customers. Store assistant takes care of tape requests. Store assistant update the rental list. Store management submits the price changes. Store management submits new tapes. Store administration produces rental reports. Main office sends overdue notices for tapes. Customer request for a tape. Store assistant checks the availability of requested tape. Store assistant searches for the available tape. Store assistant searches for the rental price of available tape. Store assistant checks status of the tape to be returned by customer. Customer can borrow if there is no delay with return of other tapes. Store assistant records rental by updating the rental list. Store assistant asks customer for his address.

Video Store Specification:

A new video store intends to offer rentals of video tapes and disks to the wider public. The store management is determined to launch its operations with the support of a computer system. The management has already sourced a number of small business software packages that might be suitable for customisation and further development. To assist with the package selection, the store hired a business analyst whose job is to determine and specify the requirements.

The video store will initially keep stock of about a thousand video tapes, and five hundred video disks. The inventory has already been ordered from one supplier, but mode suppliers will be approached in the future orders. All video tapes and disks will be bar coded so that a scanning machine integrated with the system can support the rentals and returns. The customer membership cards will also be bar coded

Existing customers will be able to place reservations on videos to be collected at a specific date. The system must have a flexible search engine to answer customer enquiries, including inquiries about movies that the video store does not stock (but may order them on request).

The video store keeps in stock an extensive library of current and popular movie titles. A particular movie may be held on video tapes or disks. Video tapes are in either 'Beta' or 'VHS' format. Video disks are in 'DVD' format.

Each movie has a particular rental period (expressed in days), with a rental charge for that period.

The video store must be able to answer immediately any inquiries about movie's stock availability and how many tapes and/or disks are available for rental (the current condition of each tape and disk must be known and recorded).

The rental charge differs depending on the video medium: tape or disk (but it is the same for the two categories of tape: Beta and VHS).

Although the DVD disk is the only format of video disks currently kept in the store, the users want the system to extend easily to other disk formats in the future.

The employees of the video store tend to remember the codes of the most popular movies. They frequently use a movie code, instead of a movie title, to identify the movies. This is a useful practice because the same movie title may have more than one release by different directors.

Payroll Specification:

A university wishes to develop a new payroll system. The university employs both full-time and casual lecturers.

Employees in the Human Resources department will use the system to maintain employee information, record and manage annual leave, record and manage sick leave and make payments to lecturers at the end of every month. In addition, they must be able to add new employee records to the system (when new staff join the university) and delete employee records when existing staff leave the university.

At the end of every month, the system must pay each lecturer the correct amount, on time, and by the payment method requested by each lecturer (pay-cheque or bank transfer). Both full-time and casual lecturers are able to view and modify their chosen payment method and personal information online. However, only full-time lecturers are able to view their payment details and leave entitlement (casual lecturers do not have leave entitlement).

Full-time lecturers are paid a flat salary, but casual lecturers work by the hour and are paid an hourly rate. This means they must submit time-cards that show the dates and number of hours worked each month. This information is used by the system to calculate the salary owed to each casual lecturer. Because casual lecturers are unable to view their payment details online, the system generates a pay slip for each casual lecturer which is then mailed to that lecturer.

Medical Specification:

The host is powered up and all software subsystems are available. The pump software system is now in the wait operating state. The patient with IV/pump running is placed onto the host. The pump cable is connected to the host. The host now provides power for the pump.

Appendix C.1 ASA Technical Evaluation

Automated Software Architect Technical Evaluation

Mark Meiklejohn

2010

Department of Computer & Information Sciences

University of Strathclyde

Evaluation Methodology

The main goal of this evaluation is to investigation the quality of the class model created by Automated Software Architect (ASA) from a requirements specification. These results are then compared to the human developed class model using the metrics of precision, recall and over-specification.

It is envisaged that during this evaluation, automation will demonstrate high levels of both recall and precision in relation to the human design. In addition, this evaluation will also attempt to identify interesting and useful information that can also be used to better the automated results.

This evaluation is based on the evaluation carried out by H.M. Harmain **[Har00]** and their implementation of an automated software developer 'CM-Builder' which is most closely related to my own work. Their evaluation utilises the metrics of both recall and precision of information extraction (IE) systems and they have further developed a new metric 'over-specification.' This metrics aims to measure how much the automatically generated model is over-specified compared to that of the ideal solution.

Since we are using Harmain's evaluation, it will only consider the actual classes identified and not any of the relationships, operations or attributes detected by the solution. This in my opinion could limit the accuracy of the evaluation, since we are not considering other aspects of the design. Harmain's reason why they only consider classes is threefold; 1) classes are more prominent than relationships, 2) there are different ways to represent the same relationship between classes and 3) software development is an iterative process. However, I believe that it is possible using these metrics to evaluate the quality of other design components in relation to the actual human design (ideal solution). With regard to point 2, it is true that relationships between classes can be represented in many ways however, is it not the point that automation can at least identify similar relationships? Although, these relationships may not be of the correct type, these should at least be considered for the quality of identification.

Criterion

The main goal is to compare both the results of automation against the human model (ideal solution). Since, different developers can produce different models and there is no gold standard per say, for a given requirements specification. There is no confident way to identify whether a design is the correct or not. However, it is possible to say whether a design is good or bad. Therefore, Harmain state that any class model that has been published in Object Oriented text book is an ideal solution.

Measures

Harmain evaluation utilises three measures; recall, precision and over-specification, which are defined below:

Recall

Their definition of recall reflects the completeness of the results produced by automation versus the correct and relevant information contained within the ideal solution produced by the human developer:

$$recall = \frac{N_{correct}}{N_{correct} + N_{mis\sin g}}$$

Where $N_{correct}$, refers to the number of correct identifications made by the system, and $N_{missing}$ refers to classes contained within the ideal solution but not identified the automated model.

Precision

This reflects the accuracy of the system based on how much of the extracted information is correct:

$$precision = \frac{N_{correct}}{N_{correct} + N_{incorrect}}$$

Where $N_{correct}$, is as above and $N_{incorrect}$ refers to incorrect identifications made by automation.

Over-Specification

This reflects how much extra correct information automation has identified in comparison to that of the ideal solution:

$$over - specification = \frac{N_{extra}}{N_{correct} + N_{mis\sin g}}$$

Where $N_{correct}$, is as above, where N_{extra} is not within the ideal solution, but subsequently considered correct for inclusion and $N_{missing}$ is within the ideal solution but not within the automated model.

Methodology Application

The classification of results first requires the ASA to process a specification, where an ideal solution exists. Once automation has completed its analysis; the classification of its results is a manual process.

The manual classification process involves reviewing each class created by automation in comparison to that of the ideal solution and by using the below definitions determine the relevant classification.

Correct

An element is considered correct if either it exactly matches or is synonymously relevant to an element contained in the ideal model. The ideal model is the model produced by the human developer.

Missing

An element is considered missing if it is contained within the ideal solution but not within the automatically derived answer.

Incorrect

An element is considered incorrect if it is not within the ideal model and both the specification and by our own judgement confirms that it is incorrect.

Extra

An element is considered extra if by our own judgment it is deemed correct, but is not contained within the ideal solution.

Measure & Method Discussion

Both precision and over-specification metrics in particular $N_{incorrect}$ and N_{extra} classifications rely on our own judgment. However, these classifications could be unconsciously biased towards producing a more positive result that over states both precision and over-specification.

As a result of this unconscious bias there are only two solutions that could be utilised by having an impartial review to classify the automated results, or to modify both precision and over-specification metrics and definitions to remove the notion of bias.

Precision

In my opinion, precision represents the correct identification of relevant terms by all returned terms within the specification. As such, the current precision metric does not actually give an accurate picture of the results returned as it is considered within the context of the human model. Therefore, to obtain an accurate identification of automations precision the metric should be revised and would be best served by this new definition; $precision = \frac{N_{correct}}{N_{all} + N_{mis\sin g}}$. Where N_{all} represents all results returned by automation, where $N_{correct}$ and $N_{missing}$ are as previously defined.

Over-Specification

Harmain's justification of over-specification relies on the generally accepted OO community definition that it is better to over specify than under **(Larman, 1998; Martin and Odell, 1995; Meyer, 1997)**. However, the ideal solution itself represents a model that may have gone through several design iterations to realise the final design. Thus, the over-specification metric contradicts the ideal solution by stating that it is not actually ideal. As a result, this would invalidate both recall and precision metrics by its definition.

On the other hand though, it is more than probable that the ideal solution has already been through this kind of over-specification phase and through design iterations much of the additional information classified as extra by myself could have already been considered and removed through this iterative process.

Therefore, the over-specification metric is important as it allows for the identification of possibly overlooked classes and relationships contained within the specification. As a result of their identification, the extra information can be presented to the developer and for them to make the relevant decision based on their expertise within a particular domain for their inclusion. However, this still leaves the issue of unconscious bias that can only be realistically resolved through an impartial classification of the results.

Results Discussion

Table 0-1 details recall, precision and over-specification metric results and Appendix Two details the actual results along with their classifications. The metric results in bold (see Table 0-1) have been used to establish the metric average. Attention must be drawn to the additional lift and taxi specification contained within the metric results table. Since the UML diagram for the lift specification is not available, the results and their classification have been based on the results obtained from a different automated system 'Dowser' [**ref**]. The Dowser system utilises a controlled grammar and therefore requires the specification to be modified to reflect this controlled grammar. As a result, the classification has been based on the results from two runs using the modified and original specifications. Furthermore, the taxi specification has also had two runs one with a language inconsistency model (see Language Inconsistency Model Effect) and one without. The reason being to determine the effect of the language inconsistency model has on recall, precision and over-specification metrics.

Overall, both recall and precision demonstrate a high average, with both taxi and library specifications demonstrating the highest recall and precision. However, is this elevated level recall and precision for taxi and library specifications a result of specification led algorithm development in relation to automation rather than a generalised algorithm development? The definition of generalised development aims to capture the abstract essence of the language structures, thus allowing the algorithm to process any specification without bias to a particular style of specification. However, both of these specifications are very concise and to the point. The lift and cinema specifications have low results for recall, these both form parts of a system analysis and design coursework obtained from the University of Strathclyde and the University of Minnesota, and were chosen for the reason that they contain spurious information and are possibly more related to real world specifications rather than the ideal solutions, which could explain their low metric results.

Over-specification on the other hand demonstrates an extremely high average although concerning there are likely causes for this. The first and most prominent consideration is that the specification being processed may contain supplementary information, which is could be either relevant or not to the final design. Furthermore, there could also be the presence of language inconsistencies that are not taken into consideration as with the library, lift and cinema specifications. Another probability is the incorrect application of the dominant semantics, which could result in erroneous classes. There has also been several different language constructs identified that express differing properties towards either reducing or identifying a greater probability of class inclusion. Nevertheless, over-specification by its own nature is identified through a manual (human) classification process (see Criterion) rather than being identified by automation, which could also account for an elevated over-specification.

Table 0-1 Metrics Results

Specification	**Recall**	**Precision**	**Over-Specification**
Taxi (without language inconsistency model)	**1**	**1**	**1.3**
Taxi (with language inconsistency model)	1	1	0.16
Lift (modified spec)	0.36	0.71	0.42
Lift (original spec)	**0.36**	**0.83**	**0.36**
Cinema	**0.5**	**0.43**	**2.7**
Library	**1**	**0.875**	**1.375**
Average Result	**0.715**	**0.784**	**1.45**

Language Inconsistency Model Effect

The language inconsistency model relates to the situation where a specific term is introduced into the specification and further in the specification it is referred to not by its original term, but by a term that could be or could not be synonymously related to it. For example, given the situation where a 'passenger' is introduced, but further in the specification it is referred to as either an 'individual' or 'group', both meaning a passenger from our contextual understanding of it. The model itself is a simple XML document that details the inconsistent (noun) term (in this example individual) and what its replacement is (passenger). As such, during processing the model is checked for any of the inconsistent terms which subsequently replaced with the correct term. This requires resolution as automation is unable to contextually resolve the situation itself. The main benefit of the language inconsistency model is primarily a reduction in the number of duplicate classes that are created and is demonstrated by a ~87% reduction in over-specification for the taxi specification (see Table 0-1). The library, lift and cinema specifications do not have this model defined which could be a factor for the elevated over-specification result.

Result Classification Analysis

Appendices Three – Six contain the UML models generated by the ASA, the human and where possible other automated systems.

This discussion aims to justify the manual classification of extra and incorrect classes. This is not evaluating whether automation has made the correct judgement based on its semantics it has, but considering the term and its context within the sentence and whether my judgement is correct for its inclusion with the final model of which is still open to interpretation and bias. However, this evaluation shall be as impartial as possible.

The discussion process will follow the below template (see Appendix One for details):

Class Name:	The name of the class created.
Trace:	Where the term has been identified within the sentence which is automatically identified by the ASA during processing.
Presence in Specification:	Discussion of the sentence in which the class is discovered.

Detected by (HFS|DS|AS):	The semantics discovered that are used to make a relevant decision regarding class creation. See Candidate Class Detection section for a definition of High Frequency Semantics (HFS), Dominant Semantics (DS) and Artefact Semantic (AS) detections.
System Impact:	Investigates what impact the class would have if included in the final design.
Original Classification:	Extra – something that the developer should consider. Although, the discovered class may not be a class but some other component of the systems first-cut design. Incorrect – something that should not be included in the design.
New Classification:	Correct – that it should be included in the design as a class Extra – something that the developer should consider. Although, the discovered class may not be a class but some other component of the systems first-cut design. Incorrect – something that should not be included in the design.

Candidate Class Detection

Prior to discussing the results of these classifications, it is best to briefly discuss the detection process. All candidate classes are nouns and the class detection algorithm bases its decision to create a class on the semantics (see Table 0-2).

As such, if either the (noun) term's highest frequency sense semantic (HFS) obtained from WordNet**[ref]** is a frequency count of the number of times a word has been seen in that sense**.** As such, it defines the most common sense understanding of a given word.

The dominant semantics (DS), which is similarly an identification of the highest frequency count for a sense, for a given (noun) term, contained within the set of all senses for a given (noun) term. Also obtained and calculated from WordNet.

If the (noun) term is within has either a HFS or DS semantics or it has an artefact semantic (AS - (similarly obtained from WordNet)) indicating a man-made object, then a class will be created.

Table 0-2 Candidate Class Semantics

Semantic:	**Description:**
Animal	Nouns denoting animals
Artefact	Nouns denoting man-made objects
Body	Nouns denoting body parts
Communication	Nouns denoting communicative processes and contents
Food	Nouns denoting foods and drinks
Group	Nouns denoting groupings of people or objects
Location	Nouns denoting spatial position
Object	Nouns denoting natural objects (not man-made)
Person	Nouns denoting people
Plant	Nouns denoting plants
Shape	Nouns denoting two and three dimensional shapes
Substance	Nouns denoting substances
Time	Nouns denoting time and temporal relations

Nouns that indicate something else

With the remaining semantic groups (see Table0-3Table 0-3 Functional Semantics) also obtained from WordNet**[ref]** are used to determine aspects of the model that may exhibit actions, relations, state, multiplicity, algorithms or attributes.

These candidates are identified during the detection process. However, the algorithm only processes nouns that have an attribute semantic. As a result of this identification, it will create a class attribute. The remaining semantics still require further consideration of their properties and how they would be applied within the first-cut model.

Table 0-3 Functional Semantics

Semantic:	**Indicates:**
Act	Action
Possession	Relation
Quantity	Multiplicity
State	State
Process	Algorithm
Motive	Algorithmic/Conditional-?
Relation	Relationships
Attribute	Class attributes

Evaluation Conclusion

In relation to overall candidate class detection process, the algorithm itself achieves a high level of both recall (71%) and precision (78%) regarding the metric analysis. This demonstrates that decisions based purely on the semantics for a given (noun) term can detect a greater than average number of classes contained within the human model. This has been determined by averaging metric the results obtained from Table 0-1.

The process also demonstrated a very high over-specification on average of 145%. These identifications allow the developer to investigate other features of the system that have been detected and to make a decision regarding their inclusion within the final design. The detection algorithm identifies the creation of each design component and tracks it through traceability links. Thus, allowing the developer to go directly to the part of the specification where the decision was made. As a result, allowing a decision to be made quickly and effectively, regarding the inclusion of additional components within the final model.

The results classification analysis focused on the additional information (over-specification and incorrect classifications) detected as candidate classes by automation, which is not inclusive of the final (human) design. This raises an issue regarding the semantics used in detection of candidate classes. Since, the decision is based on an 'or-ing' of the Highest Frequency (HFS), Dominant (DS) and Artefact (AS) semantics. Where the decision to create a class component is based on the DS and when the HFS is not within the candidate set could this represent a possible problem with the dominant semantics definition, when there is a failing to identify a design component based on the HFS or AS semantics.

The classification analysis also identified some misguided classifications, that should be changed from either extra to incorrect or correct and from incorrect to extra or correct (see Criterion). The overall impact that this would have would either increase in precision and a decrease in over-specification metric results or vice versa. However, consideration must also be given to the fact that some choices to change a classification from extra to correct. Would result in invalidating the ideal solution and could be an issue of interpretation. As a result of this identification, a consideration of what the affect would be still has to be considered.

Each of the classifications were originally considered and identified by myself. Each additional candidate (extra and incorrect classifications) was assessed for their suitability to be included within the first-cut design by an unbiased group discussion consisting of Mark Meiklejohn, Dr Marc Roper and Dr Murray Wood.

To briefly highlight, the additional information indicate either attributes in the majority cases; probable under-specifications; marker words introducing important class components; class hierarchical structures; parser miss-interpretations; language inconsistency/synonymous terms and the identification of system boundary classes or actors.

Table 4 identifies the main issues, their type, associated problems, the main finding, examples of their usage and their average frequency of occurrence. The frequency of occurrence was established by evaluating the commonality of these issues that arose across the specifications used in this evaluation and is an indication of the probability of these occurring in unseen specifications (see Appendix Seven: Frequency of Occurrence).

These discussions identified that not all of the additional information considered as extra or incorrect are relevant and should be created as a classes, but can indicate other aspects of the design (such as attributes, operations or class hierarchical structures) or are either just irrelevant to the specification and should not be included in the final design.

A common pattern was the identification related to specific linguistic structures that have a high frequency of occurrence throughout the specification document. The main issues of these structures are that the components/terms are managed individually rather than being considered as a whole, within the confines of the structure. Therefore, consideration of how these could be used and how they can assist in the creation of an enhanced first-cut design should be sought.

Of those that are not associated with linguistic structures, there are some cases where automation will fail with respect to under-specifications/boundary classes. Under-specification itself has been identified through consideration of the results obtained from automation. This typically refers to the situation where some potential design component demonstrates limited functionality, but infers that there is possibly more functionality is associated with it, but has not defined within the specification. Whereas boundary classes have similarly been identified in this way, but identify components that exhibit limited if not no behaviour associated with them and represent external components to the system itself. As such, there is no option but to have human intervention regarding these components.

Since, automation itself does not have the relevant knowledge to make a specific decision regarding their inclusion.

Furthermore, under-specification/boundary classes can indicate probable actors within the system that are fuzzy with regards to whether they should be included in the design or not. In addition, some describe the system themselves (i.e. the library, the taxi company) which could result in the creation of god classes.

The algorithm itself relies solely on the semantics of the term for its decision making process regarding class creation and does so in an effective manner by creating a greater than average number of classes that are contained within the ideal solution. However, relying on these semantics alone may not be enough for class creation and inclusion within the first-cut design.

The issues (see Table 0-4) fit nicely within our research questions and contributions to extract more information from the structure of the language, to understand what is being expressed in an abstract way to better create a first-cut design.

Key Problems to be tackled

- Linguistic Structures:
    - Since, 'X of Y' is dealt with independently of each other, when there is a probable relation between the components?
        - Considerations
            - Is Y some super type or focus of this structure?
            - Is X specific to Y but not within the confines of that type?
    - 'Has' Structure
        - Differentiation between attribute and class types.
            - Could this be achieved by considering term frequencies?
    - Verb & Preposition Linkages
        - A solution to this issue is through the use of a prepositional-verb decision matrix, where the intersection gives the relevant decision to take as a result of their combination.
        - Abstraction of the semantics for a given preposition should allow
    - Consideration of existential sentence and their high probability of introducing classes.
- Language inconsistency/Synonymous Terms
    - These typically resulting in the creation of duplicate classes.

        - The simplest route is to manually create a language inconsistency model to identify the synonymous terms and replace them during specification processing.

- Under-Specification/Boundary classes
    - Used to infer and identify possible actors, external components, or areas of the specification that requires further investigation.
    - Some resolution could be achieved through consideration of the semantics related to actors of the system, but others would require possible human intervention.
- General & common phrases terms
    - This could be resolved through the manual development of a data dictionary that could be used to perform a key word/phrase analysis to identify these terms.
        - Therefore indicating the likelihood of some new feature of the system being introduced. This could be achieved by identifying these terms and could be given special consideration during the creation of the first-cut design.
- Relationship detection
    - Although this feature is partially operational and is focused on the verb of the sentence for its identification. There still exist many relationships through the structure of the language that is currently not implemented. Such as preposition and verb constructs and the relationships between different clause types that identify implied relationships.
- Attribute detection
    - Currently only the algorithm only considers nouns that have attribute semantics, but consideration of attributive adjectives must also be considered as well to ensure that a well-formed first-cut design is produced. Furthermore, the placement of attributes is not implemented and consideration of this issue requires resolution.
- Operation & Parameter Detection
    - Although implemented the voice of sentence still requires consideration. As it will affect the placement of the operation and its parameters.
- Over-Specification
    - In the majority of cases, over-specification itself is aimed to be resolved through the resolution of these key problems. It is not that over-specification will disappear. Although it is considered that through consideration of these problems will give supporting evidence for the inclusion of design components that may otherwise go unconsidered by developer. Therefore, there may still be some developer interaction with the final model and consideration of new design components.
- Class detection based on semantics alone.
    - A consideration is that are semantics alone enough to identify all the relevant classes contained within the specification.

- A solution to the issues could also include other factors. Such as, the frequency of occurrence within the specification and consideration of a weighting, given a term and its relative position within the sentence? On the other hand, rather than relying on frequencies and weightings; could the linguistic structures be used to help identify potential classes, attributes and operations from nouns?

Table 0-4 Identified Issues

Issue: Linguistic Structure	**Type: 'X of Y'**	**Average Frequency Count: 7.75**
Examples: 1. 'Each elevator has a **set of buttons**, one button for each floor.' 2. 'All requests for floors within elevators must be serviced eventually, with floors being serviced sequentially in the **direction of travel**.' 3. 'Along with the membership number, other details on a customer must be kept such as a name, address, and **date of birth**.' 4. 'There are two **types of loan items**, language tapes, and books.' 5. 'A customer may borrow up to a **maximum of 8 items**.' 6. 'If the membership is still valid and the **number of items** on loan less than 8, the book bar code is read, either via the bar code reader or entered manually.'		
Associated Problems: There are twelve senses[ref] regarding 'X of Y' combinations and the disambiguation of these within the context of the sentence is an arduous and error prone task for automation. Additionally each component ('X' and 'Y') are handled individually of each other when they should be considered as a whole		
Findings: Of theses 12 sense, not all may be relevant when considering the language of a requirements specification document. When investigating the structure of 'X of Y' combinations, they take the following form: where 'Y' is in the majority of cases is a prepositional phrase and 'X' is either a noun or verb phrase. Similarly, in the majority of cases 'X' is typically a noun phrase. Since, a		

prepositional phrase indicates a relationship between the complement of the prepositional phrase and some other component contained in the sentence.

Therefore, in the case where ‘X’ is a noun, does this indicate that the complement of the Y is a candidate class? Similarly where the ‘X’ is a verb phrase, does the ‘Y’ indicate an attribute of some class?

Issue: Linguistic Structure	**Type: Verb & Prepositions**	**Average Frequency Count: 7**

Examples:

1. During its release period a particular film **can be shown on a number of different screens**.
2. When a vehicle **arrives at a pick up location**, the driver notifies the company.
3. Each customer **is known as a member** and is issued a membership card that shows a unique member number
4. Along with the membership number, other details on a customer **must be kept such as a name, address, and date of birth**.

Associated Problems:

A similar theme throughout, as these they are managed individually. When they should be considered as a whole.

Findings:

Prepositions themselves typically indicate a relationship between the object of the preposition and some component contained within the sentence. However, this may not be entirely true when considering some of these cases.

Example three ‘is known as’ indicates that customer is a type of member. Therefore, we would want to remove all references to customer and replace them with member.

Example four ‘must be kept’ indicates a possessive relation between ‘customer’ and the objects of the preposition.

Examples one and two indicate relationships between the object of the preposition and subject of the sentence.

Issue: Linguistic Structure	Type: Has	Average Frequency Count: 2

Examples:

1. The elevator **has** a set of buttons.
2. A language tape **has** a title language (e.g. French), and level (e.g. beginner).
3. A book **has** a title, and authors.
4. For example, matinee screenings usually **have** a lower ticket price than evening screenings, while weekend screenings usually have higher ticket prices.
5. The hotel Bolzano **has** a restaurant, a private car park and a garden.
6. It **has** 15 double rooms and 5 single rooms.
7. All rooms **have** balcony.
8. If it **has** no free vehicles, it does not operate any form of queuing system

Associated Problems:

The main issue raised here is being able to distinguish between what should be modelled and what shouldn't be in terms of being either an attribute or a class within the design.

Furthermore, some of these identified problems could be linked. In example one we have an 'X of Y' combination, as well as the 'has' structure.

Findings:

In UML modelling, a 'has a' relationship indicates an aggregation relationship between classes. However, in the above examples this may not be entirely true.

Examples 2, 3, 4, 7 and 8 all indicate attributes of the subject. Whilst examples 1 and 6 are attributes of the subject, but also imply that they should also be modelled as a class. However, this is understood from experience, not from the actual model itself. Additionally, examples 3 and 5 also demonstrate a combination of attributes and potential classes.

However, considering that the 'has' structure has a greater probability of introducing attributes rather than classes has to be considered.

A possible resolution to this problem could be addressed by considering if any of the following objects have an attribute semantic set. However, there are still problems with this consideration which could result in the creation of erroneous classes.

Issue: Linguistic Structure	**Type: Existential**	**Average Frequency Count: 1**

Examples:

1. **There** are three hotels in the chain.
2. **There** are two kinds of degrees, pass degrees and honours degrees.
3. **There** are seven standard gizmos that must be supported: bumpers (square, circular, and triangular), flippers (left and right), absorbers, and outer walls.
4. **There** are two kinds of screenings: seated and unseated ones.
5. **There** are two types of loan items, language tapes, and books.
6. **There** may also be junior and senior competitions.

Associated Problems:

There are no associated problems with the identification, but rather an observation identified during the discussion process related to this structure.

Findings:

An existential sentence indicates the existence of some element contained within the sentence. This structure therefore identifies a high probability of candidate classes relating to its structure and definition.

Issue: General/Common/Irrelevant Terms/phrases (HFS|DS)	**Average Frequency Count: 2.75**

Examples:

1. Information (communication|communication)
    a. Regarding the films, **information** that is important includes the film's classification (determined by the board of film classification) as well as its duration.
2. Details (cognition|group)
    a. Along with the membership number, other **details** on a customer must be kept such as a name, address, and date of birth.
3. Type (cognition|communication)
    a. There are two **types** of loan items, language tapes and books,
4. Addition (artefact|act)
    a. In **addition** to running the individual meets, the league prepares the schedule of meets for the season, ensures that qualified judges are assigned, registers teams and gymnasts, and publishes seasonal standings.

Associated Problems:

Class creation is decided solely by its semantics and is the primary reason as to why these are created as classes.

Examples two and three both have Highest Frequency Semantics (HFS) of (cognition), but the Dominant Semantics (DS) are within the set of possible candidates and identifies why they are created as classes. On the other hand is this a failing of the dominant semantics? As examples one and four are the opposite, their HFS are within the set of candidates, whereas the dominant semantics for example four's DS is not.

Findings:

Although these terms themselves are irrelevant to the specification, do these terms act as markers that introduce potential design components that have a greater than average chance of inclusion within the final design?

Additionally, consideration and further analysis of the (HFS|DS) semantics may also aid in reducing this type of problem.

The creation of marker word ontology could signify special behaviour for consideration. Furthermore, some of these markers are also tied up with the language structures and consideration of these could also aid this problem.

Issue: Language inconsistency/Synonymous	**Average Frequency Count: 3**

Examples:

1. Original/Concept (Screenings)
   a. Language inconsistency: Ones
2. Original/Concept (Loan Item)
   a. Language inconsistency: item
3. Original/Concept (Passenger)
   a. Language inconsistency: Group, individual, Hotel, Tourist Organisation
4. Original/Concept (Customer)
   a. Language inconsistency: Member

Associated Problems:

The main problem with language inconsistency/synonymous terms is that they create duplicate classes. In addition the creation of these classes would also effect the creation of any operations, as parameters of the operation would be of the language inconsistency type and not the overall concept.

Findings:

To limit the introduction of duplicate classes. The only feasible way to manage the situation is to create (manually) a language inconsistency model. The language inconsistency model would require the developer to read the specification and construct a simple XML file mapping between the concept and its synonymous terms. This defeats the whole purpose of the system by being able to through any specification without any special pre-processing requirements.

Although this creates an initial overhead for the developer in essence, they are creating a domain dictionary to resolve this particular issue. Furthermore, this language inconsistency model/domain dictionary is something that could be possibly reused for similar domains.

Issue: Under-Specification/System Boundary	**Average Frequency Count: 2**

Examples:

1. Bar Code Reader
2. Control Mechanism
3. Driver
4. Call
5. Film Distributor
6. Cinema Management Team
7. Cinema Staff
8. Photograph

Associated Problems:

System boundary classes and probable under-specifications although lead to the creation of classes that can indicate actors within the system, interfacing with external devices or areas that require further investigation. Its not that this is a specific problem, but more of an observation that requires resolution.

Findings:

With regard to class creation, this issue cannot be readily resolved. The classes created have the correct semantics for class creation. However, it is considered helpful to the developer that these have been identified and created. Allowing them the opportunity to consider any relationships or operations that may be created as a result of their

identification for their inclusion within the final design. As such these would typically be left to the developer to make a relevant decision regarding their inclusion within the first-cut design.	
Issue: Common Phrases	**Average Frequency Count: 0.25**
Examples: 1. date of birth 2. date of issue	
Associated Problems: Similarly, they are managed as individual parts rather than being considered as a whole. As such, this allows for the creation of additional classes when something else if specifically meant by these types of phrases. Although the examples here indicate probable attributes rather than classes.	
Findings: Common phrases can have an inferred meaning that is difficult if not impossible for automation to extract from the phrase alone.	

Appendix One - Result Classification Analysis Details

Taxi Specification (with language inconsistency model – see Appendix Four)

Out of all the specifications analysed; the taxi specification is the only one which had a language inconsistency model defined. The taxi specification had two runs one with the language inconsistency model and the other without to determine its effect. As a result it only affects over-specification metric which is demonstrated by a 114% reduction in over-specification. Given this result it allows the clear identification of what is possibly extra information contained within the specification.

Briefly, the taxi specification discusses the operation and functions of a taxi service.

Class Name:	Driver		
Trace:	[p3.1.0] When a vehicle arrives at a pick up location, the driver notifies the company. [p3.1.1] Similarly, when a passenger is dropped off at their destination, the driver notifies the company.		
Presence in Specification:	Both sentences describe an algorithmic process of causality of what should happen when a vehicle either drops off or picks up a passenger.		
Detected by (HFS|DS|AS):	Person	Person	False
System Impact	The inclusion of this class would allow the modelling of a driver, their association between them, a vehicle and the company. Additionally, there is the discovery of a notification operation between the driver and the company.		
Original Classification:	Extra		
New Classification:	Correct		

Class Name:	Call		
Trace:	[p2.1.0] When the company receives a call from an individual, hotel, entertainment venue, or tourist organization, it tries to schedule a vehicle to pick up the fare.		
Presence in Specification:	The sentence identifies an algorithmic process describing the actions that take place as a result of receiving a call.		
Detected by (HFS|DS|AS):	Communication	Communication	False
System Impact	The inclusion of this class within the model would have little impact on the actual final model as it forms part of an operation (recevieCall) and not actually a class.		
Original Classification:	Extra		
New Classification:	Extra		

Library Specification (see Appendix Three)

The library specification details and discusses the operations and features of a library service.

Class Name:	Author		
Trace:	[p2.1.3] A book has a title, and authors.		
Presence in Specification:	This term is in the situation where it is being described as an attribute of a book rather than an actual class.		
Detected by (HFS|DS|AS):	Person	Person	False
System Impact:	By allowing author to exist as a class would be beneficial as an author would be associated with many books and could provide a facility for searches to be run for a particular author.		
Original Classification:	Extra		

New Classification:	Extra

Class Name:	Member		
Trace:	[p1.1.1] Each customer is known as a member and is issued a membership card that shows a unique member number.		
Presence in Specification:	This is being used to indicate that the customer is a type of member through some form of hierarchical relationship.		
Detected by (HFS|DS|AS):	Person	Body	False
System Impact:	The ASA algorithm itself, through its relational processor, identifies that customer is a type of member. As such a member interface and abstraction is created with a concrete type of customer is defined.		
Original Classification:	Extra		
New Classification:	Extra		

Class Name:	Membership		
Trace:	[p3.1.3] If the membership is still valid and the number of items on loan less than 8, the book bar code is read, either via the bar code reader or entered manually.		
Presence in Specification:	This term forms part of a discussion regarding the issuing of loan items and ensures that the customers membership is still valid and the introduction of preconditions prior to a loan.		
Detected by (HFS|DS|AS):	Group	Group	False

System Impact:	The identification has no impact on the system as it is used as a synonymous towards customer/member. If a language inconsistency model had been created then this issue would not have arisen.
Original Classification:	Extra
New Classification:	Extra

Class Name:	Birth		
Trace:	[p1.1.2] Along with the membership number, other details on a customer must be kept such as a name, address, and date of birth.		
Presence in Specification:	Forms part of a discussion regarding information that should be retained for a customer. Furthermore, 'birth' forms part of a statement 'date of birth'		
Detected by (HFS|DS|AS):	Time	Time	False
System Impact:	The inclusion of this class has no benefit towards the system as a whole and should be an attribute.		
Original Classification:	Extra		
New Classification:	Extra		

Class Name:	Date
Trace:	[p1.1.2] Along with the membership number, other details on a customer must be kept such as a name, address, and date of birth.

Presence in Specification:	Forms part of a discussion regarding information that should be retained for a customer. Similarly, forms part of the statement ‘date of birth’.		
Detected by (HFS|DS|AS):	Time	Time	False
System Impact:	The inclusion of this class has no benefit towards the system as a whole and should be an attribute.		
Original Classification:	Extra		
New Classification:	Extra		

Class Name:	Name		
Trace:	[p1.1.2] Along with the membership number, other details on a customer must be kept such as a name, address, and date of birth.		
Presence in Specification:	Forms part of a discussion regarding information that should be retained for a customer.		
Detected by (HFS|DS|AS):	Communication	Communication	False
System Impact:	The inclusion of this class has no benefit towards the system as a whole and should be an attribute.		
Original Classification:	Extra		
New Classification:	Extra		

Class Name:	Update

Trace:	[p4.1.0] The library must support the facility for an item to be searched and for a daily update of records.		
Presence in Specification:	Forms part of a statement *"a daily update of records"* that discusses the facilities that the library system must offer		
Detected by (HFS|DS|AS):	Communication	Communication	False
System Impact:	Has no benefit towards the overall final design and would be best suited as some form of operation contained within the library class.		
Original Classification:	Extra		
New Classification:	Incorrect		

Class Name:	Record		
Trace:	[p4.1.0] The library must support the facility for an item to be searched and for a daily update of records.		
Presence in Specification:	Similarly, forms part of a statement *"a daily update of records"* that discusses the facilities that the library system must offer		
Detected by (HFS|DS|AS):	Communication	Communication	True
System Impact:	Similarly, as update would form part of an operation		
Original Classification:	Extra		
New Classification:	Extra		

Class Name:	Type

Trace:	[p2.1.1] There are two types of loan items, language tapes, and books.		
Presence in Specification:	This appears in a sentence which is discussing the existence of two forms of loan items namely books and language tapes.		
Detected by (HFS|DS|AS):	Cognition	Communication	True
System Impact:	This class identification would have no impact within the system and is completely erroneous.		
Original Classification:	Extra		
New Classification:	Incorrect		

Class Name:	Bar Code Reader		
Trace:	[p3.1.2] When an item is issued the customer's membership number is scanned via a bar code reader or entered manually. [p3.1.3] If the membership is still valid and the number of items on loan less than 8, the book bar code is read, either via the bar code reader or entered manually.		
Presence in Specification:	Forms part of an algorithmic process on how the reader is utilised during the processing of a loan item(s) and customer details.		
Detected by (HFS|DS|AS):	Artefact	Artefact	True
	Communication	Communication	False
	Person	Person	False

System Impact:	This class although standalone identifies the requirement of an interface between two separate systems; the library system and the bar code reader.
Original Classification:	Extra
New Classification:	Extra

Class Name:	Item		
Trace:	[p3.1.0] A customer may borrow up to a maximum of 8 items. [p3.1.1] An item can be borrowed, reserved or renewed to extend a current loan. [p3.1.2] When an item is issued the customer's membership number is scanned via a bar code reader or entered manually. [p3.1.3] If the membership is still valid and the number of items on loan less than 8, the book bar code is read, either via the bar code reader or entered manually. [p3.1.4] If the item can be issued (e.g. not reserved) the item is stamped and then issued. [p4.1.0] The library must support the facility for an item to be searched and for a daily update of records.		
Presence in Specification:	A synonymous towards loan item		
Detected by (HFS|DS|AS):	Communication	Communication	True
System Impact:	The identification has no impact on the system as it is used as a synonym towards loan item. If a language inconsistency model had been created then this issue would not have arisen.		

Original Classification:	Extra
New Classification:	Extra

Class Name:	Detail		
Trace:	[p1.1.2] Along with the membership number, other details on a customer must be kept such as a name, address, and date of birth.		
Presence in Specification:	Highlights the important information that should be retained for a customer.		
Detected by (HFS|DS|AS):	Cognition	Group	False
System Impact:	None; should not be included in the model.		
Original Classification:	Incorrect		
New Classification:	Incorrect		

Lift Specification (see Appendix Six)

To give a short overview, the lift specification discusses the operation and feature of a lift service.

Class Name:	Control Mechanism		
Trace:	[p1.1.1] The elevators and the control mechanism are supplied by a manufacturer.		
Presence in Specification:	This sentence is discussing parts of the system that is supplied by the manufacturer of which is not stated.		
Detected by (HFS|DS|AS):	Attribute	Attribute	True
	Process	Process	True

System Impact:	The inclusion of this class within in the model provides no benefit to the system at all. There are no relationships with any other component contained within the model.
Original Classification:	Extra
New Classification:	Incorrect

Class Name:	Manufacturer		
Trace:	[p1.1.1] The elevators and the control mechanism are supplied by a manufacturer.		
Presence in Specification:	Similarly, as above, identifies who supplies the relevant parts that are required the installation of a lift within a building.		
Detected by (HFS|DS|AS):	Group	Group	False
System Impact:	Similarly, as above, the inclusion of this class serves has no benefit in the final model.		
Original Classification:	Extra		
New Classification:	Incorrect		

Class Name:	Direction
Trace:	[p3.1.2] The buttons are cancelled when an elevator visits the floor and is either travelling the desired direction, or visiting a floor with no requests outstanding. [p4.1.2] All requests for floors within elevators must be serviced eventually, with floors being serviced sequentially in the direction of travel.

Presence in Specification:	Sentence p3.1.2 is discussing the operation of buttons and what happens as a result of the lift travelling a particular direction. Similarly, sentence p4.1.2 discusses the operation of requests and how they should be served.		
Detected by (HFS|DS|AS):	Location	Communication	False
System Impact:	As a class it has no real impact on the overall model and would be best suited as an attribute of the elevator class.		
Original Classification:	Extra		
New Classification:	Extra		

Class Name:	Floor Request Button		
Trace:	[p3.1.3] In the latter case, if both floor request buttons are illuminated, only one should be cancelled.		
Presence in Specification:	Introduces a conditional argument related to the previous sentence [p3.1.2] regarding the cancelation of the floor button illumination if both have been pressed. The decision is based on the direction of lift travel.		
Detected by (HFS|DS|AS):	Artefact	Artefact	True
	Communication	Communication	False
	Artefact	Artefact	True
System Impact:	Introduces a specific type of button and condition that is associated to a particular floor contained within the building.		
Original Classification:	Extra		
New Classification:	Correct		

Class Name:	Destination		
Trace:	[p4.1.0] When an elevator has no requests to service, it should remain at its final destination with its doors closed and await further requests.		
Presence in Specification:	Introduces an algorithmic process regarding what should happen when a lift has no requests to service. That is to remain at its current destination.		
Detected by (HFS|DS|AS):	Location	Location	False
System Impact:	This class has no additional benefit regarding the final model. As such should not be included within the final design.		
Original Classification:	Incorrect		
New Classification:	Incorrect		

Class Name:	Set		
Trace:	[p2.1.0] Each elevator has a set of buttons, one button for each floor.		
Presence in Specification:	This sentence identifies the components contained within the lift. Furthermore, 'Set' is part of the statement 'set of buttons' indicating that the elevator has a set of buttons.		
Detected by (HFS|DS|AS):	Group	Group	True
System Impact:	The inclusion of this class indicates some form of data structure to manage the collection buttons that a lift has.		

	However, it has no real benefit to the final model but its identification should be considered within the system.
Original Classification:	Incorrect
New Classification:	Extra

Cinema Specification (see Appendix Five)

Extra Classifications:

Class Name:	Date
Trace:	[p3.1.0] Screenings are open for ticket sales one week before the date they take place. [p6.1.0] For the system to be able to support the cinema management team it should be able to produce the following kinds of statistics: the number of ticket sales to date per film, the revenue of the ticket sales per film, the percentage of empty seats for each screening for the current or future weeks, the ticket sales and revenue for each screening for the current week, a listing of films ordered by ticket sales or revenue for the current week. [p8.1.2] The cards are valid for six months from the date of issue and each month the customer is charged the monthly subscription.
Presence in Specification:	Date is used indifferent ways throughout the specification. In [p3.1.0] date is referring to when tickets can be sold for a screening. In [p6.1.0] date is referring to statistical information regarding ticket sales and in [p8.1.2] date refers to the validity period of a cinema card.

Detected by (HFS|DS|AS):	Time	Time	False
System Impact:	For two of the cases [p3.1.0, p8.1.2] ‘date’ would be best served as an attribute of a particular class and with [p6.1.0] date would infer that it is part of a calendar application which allows the selection of particular dates to generate relevant statistical information.		
Original Classification:	Extra		
New Classification:	Extra		

Class Name:	Day		
Trace:	[p2.1.0] The cinema operation is organised around a screening schedule, which is a timetable listing the films that will be shown on each screen each day of the week.		
Presence in Specification:	Used to identify a particular day and what films will be shown on that date.		
Detected by (HFS|DS|AS):	Time	Time	False
System Impact:	The inclusion of this class would allow the identification of which film is shown when. However, the class ‘day’ infers that it is part of a separate application. Such as a calendar application which is used in conjunction with the cinema application. This class on its own would have little relevance to the cinema application itself.		
Original Classification:	Extra		
New Classification:	Extra		

Class Name:	Debit Card		
Trace:	[p7.1.0] Ticket sales are handled by cinema staff and payment can be made in three forms: by cash; by credit or debit card; by using cinema membership cards.		
Presence in Specification:	Infers one of three payment types that the cinema can accept.		
Detected by (HFS|DS|AS):	Possession	Possession	False
	Artefact	Communication	True
System Impact:	The inclusion of this class would allow the modelling of this form of payment type along with others that are also present. Furthermore, the human design has a 'CardSale' class that could be synonymous with this class. The modelling of this class could be important in that some debit and credit cards have different information compared to other such as issues numbers. However, this would not require the modelling of a separate class but be handled through attributes of a card class implementation.		
Original Classification:	Extra		
New Classification:	Correct		

Class Name:	Credit
Trace:	[p7.1.0] Ticket sales are handled by cinema staff and payment can be made in three forms: by cash; by credit or debit card; by using cinema membership cards. [p8.1.1] When signing up for a cinema card, the following are required: a photograph of the customer which is taken on the spot and is attached to the card, customer information such as

	name and address, and credit or debit card details for the monthly subscription charge.		
Presence in Specification:	Infers a payment type.		
Detected by (HFS|DS|AS):	Communication	Communication	False
System Impact:	Similarly as previously discussed for 'debit card' Furthermore, the algorithm does not detect that this term is of the same type as 'debit card'. Hence why there is no reference to card for this class detection.		
Original Classification:	Extra		
New Classification:	Correct		

Class Name:	Evening Screening		
Trace:	[p3.1.6] For example, matinee screenings usually have a lower ticket price than evening screenings, while weekend screenings usually have higher ticket prices.		
Presence in Specification:	This is used within the specification to identify relevant ticket prices for showings at different times of the day.		
Detected by (HFS|DS|AS):	Time	Time	false
	Communication	Act	True
System Impact:	The inclusion of this class would allow for the modelling of different types of screenings with different ticket prices. Furthermore, the inclusion of this hierarchical structure creates an area of flexibility that can allow the inclusion of different types of screenings.		

Original Classification:	Extra		
New Classification:	Correct		

Class Name:	Weekend Screening		
Trace:	[p3.1.6] For example, matinee screenings usually have a lower ticket price than evening screenings, while weekend screenings usually have higher ticket prices.		
Presence in Specification:	See evening screening discussion		
Detected by (HFS|DS|AS):	Time	Time	False
	Communication	Act	True
System Impact:	See evening screening discussion		
Original Classification:	Extra		
New Classification:	Correct		

Class Name:	Matinee Screening		
Trace:	[p3.1.6] For example, matinee screenings usually have a lower ticket price than evening screenings, while weekend screenings usually have higher ticket prices.		
Presence in Specification:	See evening screening discussion		
Detected by (HFS|DS|AS):	Communication	Communication	False
	Communication	Act	True

System Impact:	See evening screening discussion
Original Classification:	Extra
New Classification:	Correct

Class Name:	Board		
Trace:	[p5.1.0] Regarding the films, information that is important includes the film's classification (determined by the board of film classification) as well as its duration. [p5.1.1] This information is important as it affects the scheduling process and the allocation of films to screens.		
Presence in Specification:	This class identification demonstrates where the film’s classification has been obtained		
Detected by (HFS|DS|AS):	Group	Artefact	True
System Impact:	This class is not relevant to the final class design.		
Original Classification:	Extra		
New Classification:	Incorrect		

Class Name:	Information
Trace:	[p5.1.0] Regarding the films, information that is important includes the film's classification (determined by the board of film classification) as well as its duration. [p5.1.1] This information is important as it affects the scheduling process and the allocation of films to screens.

Presence in Specification:	Used as a synonym for the attributes classification and duration related to a film.		
Detected by (HFS|DS|AS):	Communication	Communication	False
System Impact:	This identification has no benefit towards the final design.		
Original Classification:	Extra		
New Classification:	Incorrect		

Class Name:	Cinema Membership Card		
Trace:	[p7.1.0] Ticket sales are handled by cinema staff and payment can be made in three forms: by cash; by credit or debit card; by using cinema membership cards.		
Presence in Specification:	As with debit and credit cards, 'cinema membership card' is also identified as a payment type. Furthermore, this term is also a synonym toward 'cinema card'.		
Detected by (HFS|DS|AS):	Communication	Communication	True
	Group	Group	False
	Artefact	Communication	True
System Impact:	Allows for the modelling of cinema members		
Original Classification:	Extra		
New Classification:	Correct		

Class Name:	Film Distributor

Trace:	[p1.1.0] The cinema leases films for screening from film distributors.		
Presence in Specification:	This identifies the source of where the films are obtained from.		
Detected by (HFS|DS|AS):	Communication	Artefact	True
	Person	Person	True
System Impact:	The inclusion of this class would allow the modelling of where films are obtained from and their contact details. Due to the hierarchical structure that is developed as a result of its identification would also allow for the inclusion of other distributors not discussed within the specification. Its impact on the model as a whole is insignificant in the grand scheme of what the specification is expressing.		
Original Classification:	Extra		
New Classification:	Extra		

Class Name:	Listing
Trace:	[p6.1.0] For the system to be able to support the cinema management team it should be able to produce the following kinds of statistics: the number of ticket sales to date per film, the revenue of the ticket sales per film, the percentage of empty seats for each screening for the current or future weeks, the ticket sales and revenue for each screening for the current week, a **listing** of films ordered by ticket sales or revenue for the current week.
Presence in Specification:	Listing in the specification is expressing an ordering of films with regard to some constraint.

Detected by (HFS|DS|AS):	Communication	Communication	False
System Impact:	The inclusion of this class has little benefit to the overall system. It is highly unlikely that the focus of this application would require a 'listing' class.		
Original Classification:	Extra		
New Classification:	Incorrect		

Class Name:	Cinema Management Team		
Trace:	[p6.1.0] For the system to be able to support the cinema management team it should be able to produce the following kinds of statistics: the number of ticket sales to date per film, the revenue of the ticket sales per film, the percentage of empty seats for each screening for the current or future weeks, the ticket sales and revenue for each screening for the current week, a listing of films ordered by ticket sales or revenue for the current week.		
Presence in Specification:	Identifying a user of the system that can obtain some relevant statistics regarding films.		
Detected by (HFS|DS|AS):	Communication	Communication	True
	Act	Act	False
	Group	Group	False
System Impact:	The inclusion of this class within the system would be beneficial as it would allow the modelling of this type of user.		
Original Classification:	Extra		
New Classification:	Extra		

Class Name:	Management Team		
Trace:	[p6.1.1] It should also allow the management team to enter the new screening schedule and make changes to the current screening schedule.		
Presence in Specification:	As before a synonym for 'Cinema Management Team', but also refers to particular privileges that the manager of the cinema has regarding access to the computer system.		
Detected by (HFS|DS|AS):	Act	Act	False
	Group	Group	False
System Impact:	As before,		
Original Classification:	Extra		
New Classification:	Extra		

Class Name:	Cinema Staff		
Trace:	[p7.1.0] Ticket sales are handled by cinema staff and payment can be made in three forms: by cash; by credit or debit card; by using cinema membership cards.		
Presence in Specification:	Identifies a user of the system.		
Detected by (HFS|DS|AS):	Communication	Communication	False
	Group	Group	True
System Impact:	The inclusion of staff members will allow a clear separation between the identification of this and the management team.		

	Furthermore, being able to track who has handled a ticket sale or more is important to the operation of the cinema.
Original Classification:	Extra
New Classification:	Extra

Class Name:	Month		
Trace:	[p4.1.1] According to this scheme every subscribed customer pays a monthly subscription, which allows them to buy a fixed number of tickets for any screening during the month. [p8.1.2] The cards are valid for six months from the date of issue and each month the customer is charged the monthly subscription.		
Presence in Specification:	Month is used in three ways; firstly it is used to refer to a monthly subscription free, the number of tickets that can be purchased during a month and the validity period of the cinema membership card. (where the term 'card' in [p8.1.2] is being used to refer to cinema membership card')		
Detected by (HFS|DS|AS):	Time	Time	False
System Impact:	The inclusion of month as a class has no benefit towards the final design. In most cases the term 'month' is best suited to being an attribute rather than class or some operation that is performed once a month.		
Original Classification:	Extra		
New Classification:	Extra		

Class Name:	Card		
Trace:	[p8.1.0] Cinema cards are personal (i.e. only the person named on the card can use it) and they are limited to a maximum of four tickets per screening. [p8.1.1] When signing up for a cinema card, the following are required: a photograph of the customer which is taken on the spot and is attached to the card, customer information such as name and address, and credit or debit card details for the monthly subscription charge. [p8.1.2] The cards are valid for six months from the date of issue and each month the customer is charged the monthly subscription.		
Presence in Specification:	This term is being used as a synonym toward 'cinema card' as such would be resolved by the language inconsistency model.		
Detected by (HFS|DS|AS):	Artefact	Communication	True
System Impact:	Cinema cards has already been identified as part of the final design therefore its inclusion is important		
Original Classification:	Extra		
New Classification:	Correct		

Class Name:	Person
Trace:	[p8.1.0] Cinema cards are personal (i.e. only the person named on the card can use it) and they are limited to a maximum of four tickets per screening.

Presence in Specification:	Is used to define the term ‘personal’, that it is only one particular person that can actually use the card, that is its owner.		
Detected by (HFS|DS|AS):	Body	Body	False
System Impact:	This class is as a particular type of customer who pays a subscription charge to the cinema. The inclusion of this class would be incorrect		
Original Classification:	Extra		
New Classification:	Extra		

Class Name:	Student		
Trace:	[p3.1.4] There are a number of different types of tickets associated with each screening, which include normal tickets, concessionary tickets for students and senior citizens, discounted family tickets, etc.		
Presence in Specification:	Student a particular type of customer is being referred to as some whom can receive a special type.		
Detected by (HFS|DS|AS):	Person	Person	False
System Impact:	The inclusion of this type would allow the modelling of a particular type of customer of which a particular type of ticket is available for. Furthermore, the specification stops short at defining any more information regarding ticket types and what else maybe on offer.		
Original Classification:	Extra		

New Classification:	Extra

Class Name:	Citizen		
Trace:	[p3.1.4] There are a number of different types of tickets associated with each screening, which include normal tickets, concessionary tickets for students and senior citizens, discounted family tickets, etc.		
Presence in Specification:	Similarly, as previous, a kind of customer.		
Detected by (HFS|DS|AS):	Person	Person	False
System Impact:	Similarly, as previous defined for 'Student'		
Original Classification:	Extra		
New Classification:	Extra		

Class Name:	Photograph		
Trace:	[p8.1.1] When signing up for a cinema card, the following are required: a photograph of the customer which is taken on the spot and is attached to the card, customer information such as name and address, and credit or debit card details for the monthly subscription charge.		
Presence in Specification:	The photograph of the customer is used as a visual identification as their ownership of a cinema card		
Detected by (HFS|DS|AS):	Artefact	Artefact	True

System Impact:	This has no additional benefit to the final design as it would if it were to be included. This class if created would more than probably be a data class which holds a reference to the actual picture held on file.
Original Classification:	Extra
New Classification:	Incorrect

Class Name:	Place		
Trace:	[p3.1.0] Screenings are open for ticket sales one week before the date they take place.		
Presence in Specification:	Place is referred to in the specification as a particular time period in which tickets can be purchased for a particular screening.		
Detected by (HFS|DS|AS):	Location	Location	False
System Impact:	The inclusion of this within final model has no benefit to the final design nor does it have any benefit as an attribute of any class.		
Original Classification:	Extra		
New Classification:	Incorrect		

Class Name:	Release Period
Trace:	[p2.1.2] During its release period a particular film can be shown on a number of different screens.

Presence in Specification:	This is basically stating what can happen for a given release period of a film.		
Detected by (HFS|DS|AS):	Artefact	Act	True
	Time	Time	False
System Impact:	The term is more suited to being an attribute of a film since it provides no additional benefit as being a class.		
Original Classification:	Extra		
New Classification:	Extra		

Class Name:	Screening
Trace:	[p1.1.0] The cinema leases films for screening from film distributors. [p2.1.2] During its release period a particular film can be shown on a number of different screens. [p3.1.1] There are two kinds of screenings: seated and unseated ones. [p3.1.2] The main difference between the two is that for seated screenings the customer is allocated a particular seat, while for unseated screenings no specific seat is allocated. [p3.1.3] For each screening the total number of tickets sold should not exceed the seating capacity for that screen. [p3.1.4] There are a number of different types of tickets associated with each screening, which include normal tickets, concessionary tickets for students and senior citizens, discounted family tickets, etc.

	[p3.1.5] The price of each type of ticket may be different for each screening. [p4.1.1] According to this scheme every subscribed customer pays a monthly subscription, which allows them to buy a fixed number of tickets for any screening during the month. [p6.1.0] For the system to be able to support the cinema management team it should be able to produce the following kinds of statistics: the number of ticket sales to date per film, the revenue of the ticket sales per film, the percentage of empty seats for each screening for the current or future weeks, the ticket sales and revenue for each screening for the current week, a listing of films ordered by ticket sales or revenue for the current week. [p7.1.1] In the case where the sale is for a seated screening the customer should be able to select the seats they most prefer from those that are available. [p8.1.0] Cinema cards are personal (i.e. only the person named on the card can use it) and they are limited to a maximum of four tickets per screening.		
Presence in Specification:	This term is used throughout the specification to refer to different aspects of the system. Some giving specific information regarding a screening others regarding information to be obtained about a screening.		
Detected by (HFS|DS|AS):	Communication	Act	True
System Impact:	This class is actually in the human model but under a different term of 'showing'. Within the human model the term 'showing' must have been considered a better descriptive name rather than 'screening'.		

Original Classification:	Extra
New Classification:	Correct

Class Name:	Seat		
Trace:	[p3.1.2] The main difference between the two is that for seated screenings the customer is allocated a particular seat, while for unseated screenings no specific seat is allocated. [p6.1.0] For the system to be able to support the cinema management team it should be able to produce the following kinds of statistics: the number of ticket sales to date per film, the revenue of the ticket sales per film, the percentage of empty seats for each screening for the current or future weeks, the ticket sales and revenue for each screening for the current week, a listing of films ordered by ticket sales or revenue for the current week. [p7.1.1] In the case where the sale is for a seated screening the customer should be able to select the seats they most prefer from those that are available.		
Presence in Specification:	It is utilised in two-ways, one where it allows a customer to book a seat and the other that can be used to identify some statistics related to a particular screening.		
Detected by (HFS|DS|AS):	Location	Artefact	True
System Impact:	This is already defined within the human model as a particular type of screening and also has an attribute to cover the discussion in [p6.1.0]. Furthermore, a seat for a particular screen may have more information related to it than is discussed in the specification.		

Original Classification:	Extra
New Classification:	Correct

Class Name:	Subscription Charge		
Trace:	[p8.1.1] When signing up for a cinema card, the following are required: a photograph of the customer which is taken on the spot and is attached to the card, customer information such as name and address, and credit or debit card details for the monthly subscription charge.		
Presence in Specification:	This only identifies that there is a subscription charge but does not provide any other additional information regarding this. Only that a credit or debit card details are required but never expanded upon.		
Detected by (HFS|DS|AS):	Possession	Act	False
	Communication	Communication	True
System Impact:	The inclusion of this class would allow the management of the subscription charges for many customers over many time periods.		
Original Classification:	Extra		
New Classification:	Extra		

Class Name:	Cinema
Trace:	[p1.1.0] The cinema leases films for screening from film distributors.

	[p1.1.2] The cinema may lease more than one copy of films that are very popular. [p4.1.0] The cinema wishes to operate a customer cinema card scheme.		
Presence in Specification:	The term cinema is used to demonstrate relationships between films.		
Detected by (HFS|DS|AS):	Communication	Communication	False
System Impact:	The inclusion of this class would only server to identify a cinema within a chain of cinemas, which is not explored any further within the specification. From the specification, it is apparent that it refers only to the operation of a cinema not many. As such, its inclusion would have no additional benefit to the final model.		
Original Classification:	Extra		
New Classification:	Extra		

Class Name:	Type
Trace:	[p3.1.4] There are a number of different types of tickets associated with each screening, which include normal tickets, concessionary tickets for students and senior citizens, discounted family tickets, etc. [p3.1.5] The price of each type of ticket may be different for each screening.
Presence in Specification:	The term ‘type’ is used to refer/introduce different types of tickets. Such as, normal, concessionary and family tickets.

Detected by (HFS|DS|AS):	Cognition	Communication	True
System Impact:	The inclusion of the class 'type' clearly has no reason to be included in the design.		
Original Classification:	Extra		
New Classification:	Incorrect		

Class Name:	Week		
Trace:	[p2.1.0] The cinema operation is organised around a screening schedule, which is a timetable listing the films that will be shown on each screen each day of the week. [p2.1.1] This screening schedule is different every week. [p3.1.0] Screenings are open for ticket sales one week before the date they take place. [p6.1.0] For the system to be able to support the cinema management team it should be able to produce the following kinds of statistics: the number of ticket sales to date per film, the revenue of the ticket sales per film, the percentage of empty seats for each screening for the current or future weeks, the ticket sales and revenue for each screening for the current week, a listing of films ordered by ticket sales or revenue for the current week.		
Presence in Specification:	This is referring to how the cinema is operated, when some action can take place and the kinds of statistics that could be generated		
Detected by (HFS|DS|AS):	Time	Time	False

System Impact:	The introduction of the class week would have no benefit on the overall final design. In most cases it appears that week is an attribute of some other class or that it is part of a separate application which manages date which other classes take advantage of.
Original Classification:	Extra
New Classification:	Extra

Class Name:	Copy		
Trace:	[p1.1.1] Each lease is for one copy of the film. [p1.1.2] The cinema may lease more than one copy of films that are very popular. [p2.1.3] The same film cannot be shown on more than one screen at a time unless there are multiple copies.		
Presence in Specification:	Indicates what a copy is and identifies that a cinema can have multiple copies of a film.		
Detected by (HFS|DS|AS):	Communication	Communication	True
System Impact:	This should be an attribute rather than a class. As it has no benefit to the final model.		
Original Classification:	Extra		
New Classification:	Extra		

Class Name:	Debit Card Detail

Trace:	[p8.1.1] When signing up for a cinema card, the following are required: a photograph of the customer which is taken on the spot and is attached to the card, customer information such as name and address, and credit or debit card details for the monthly subscription charge.		
Presence in Specification:	Refers to the relevant credit or debit card details but does actually state what that information is. Furthermore, this noun phrase although correct in its identification, the head of the noun 'detail' has no additional m		
Detected by (HFS|DS|AS):	Possession	Possession	False
	Artefact	Communication	True
	Cognition	Group	False
System Impact:	The inclusion of this class as is 'debit card detail' would be incorrect and should be considered as a synonym toward 'Card Sale' that has already been defined in the human model.		
Original Classification:	Incorrect		
New Classification:	Extra		

Class Name:	Following
Trace:	[p8.1.1] When signing up for a cinema card, the following are required: a photograph of the customer which is taken on the spot and is attached to the card, customer information such as name and address, and credit or debit card details for the monthly subscription charge.
Presence in Specification:	Details what important information is required for the setting up of a cinema card

Detected by (HFS|DS|AS):	Group	Group	False
System Impact:	Has no benefit in any way towards the final design.		
Original Classification:	Incorrect		
New Classification:	Incorrect		

Class Name:	I		
Trace:	[p8.1.0] Cinema cards are personal (i.e. only the person named on the card can use it) and they are limited to a maximum of four tickets per screening.		
Presence in Specification:	A miss-identification stems from (i.e.).		
Detected by (HFS|DS|AS):	Substance	Substance	False
System Impact:	None – Should not be included		
Original Classification:	Incorrect		
New Classification:	Incorrect		

Class Name:	Issue
Trace:	[p8.1.2] The cards are valid for six months from the date of issue and each month the customer is charged the monthly subscription.
Presence in Specification:	Forms part of the statement ‘date of issue’ indicating a start period for some type of card.

Detected by (HFS|DS|AS):	Cognition	Cognition	True
System Impact:	Would be best suited as an attribute rather than a class.		
Original Classification:	Incorrect		
New Classification:	Extra		

Class Name:	Name		
Trace:	[p8.1.1] When signing up for a cinema card, the following are required: a photograph of the customer which is taken on the spot and is attached to the card, customer information such as name and address, and credit or debit card details for the monthly subscription charge.		
Presence in Specification:	Takes form as an attribute of a customer		
Detected by (HFS|DS|AS):	Communication	Communication	False
System Impact:	Would have no benefit to the final model, should be an attribute of customer.		
Original Classification:	Incorrect		
New Classification:	Extra		

Class Name:	Number
Trace:	[p2.1.2] During its release period a particular film can be shown on a number of different screens.

	[p3.1.3] (algorithmic) For each screening the total number of tickets sold should not exceed the seating capacity for that screen. [p3.1.4] (Existential) There are a number of different types of tickets associated with each screening, which include normal tickets, concessionary tickets for students and senior citizens, discounted family tickets, etc. [p4.1.1] According to this scheme every subscribed customer pays a monthly subscription, which allows them to buy a fixed number of tickets for any screening during the month. [p6.1.0] (algorithmic) For the system to be able to support the cinema management team it should be able to produce the following kinds of statistics: the number of ticket sales to date per film, the revenue of the ticket sales per film, the percentage of empty seats for each screening for the current or future weeks, the ticket sales and revenue for each screening for the current week, a listing of films ordered by ticket sales or revenue for the current week.		
Presence in Specification:	In some cases the term number is part of some statement 'number-of'. This appears to take the role of either indicating multiplicity or an attribute of a particular class.		
Detected by (HFS|DS|AS):	Attribute	Communication	True
System Impact:	As a class this has no benefit to the overall design. However, it does exhibit multiplicity or attributive qualities.		
Original Classification:	Incorrect		
New Classification:	Incorrect		

Appendix Two - Results Data:

Lift Specification Results:

Table 1 Lift Specification Results

Ideal Solution (automated)	**Automated Solution (Original-Spec)**	**Classification**	**Automated Solution (Mod-Spec)**	**Classification**
Button	Button	Correct	Button	Correct
Door	Door	Correct		Missing
Elevator	Elevator	Correct	Elevator	Correct
Elevator System		Missing		Missing
Floor	Floor	Correct	Floor	Correct
Request	Request	Correct	Request	Correct
Building		Missing	Building	Correct
Down elevator		Missing		Missing
Up elevator		Missing		Missing
Waiting time		Missing		Missing
User		Missing		Missing
Down button		Missing		Missing
Up button		Missing		Missing
Illumination		Missing		Missing
	Destination	Extra	Destination	Extra
	Control Mechanism	Extra		Extra
	Manufacturer	Extra		Extra
	Set	Incorrect		Extra
	Floor Request Button	Extra		Extra
	Direction	Extra	Direction	Extra
			Ground	Incorrect
			UserPress	Incorrect

Table 1a Lift Specification Results - Original Specification

Correct	**Incorrect**	**Missing**	**Extra**
5	1	9	5

$$recall = \frac{5}{5+9} = 0.36$$

$$precision = \frac{5}{5+1} = 0.83$$

$$over - specification = \frac{5}{5+9} = 0.36$$

New Metrics Application:

$$precision = \frac{5}{11+9} = 0.25$$

Table 1b Lift Specification Results - Modified Specification			
Correct	**Incorrect**	**Missing**	**Extra**
5	2	9	6

$$recall = \frac{5}{5+9} = 0.36$$

$$precision = \frac{5}{5+2} = 0.71$$

$$over - specification = \frac{6}{5+9} = 0.42$$

New Metrics Application:

$$precision = \frac{5}{9+9} = 0.28$$

Library Specification Results:

Table 2 Library Specification Results				
Ideal Solution	**Automated Solution**	**Classification**	**CM Builder Results**	**Classification**
Book	Book	Correct	Book	Correct
Customer	Customer	Correct	Customer	Correct
Language Tape	Language Tape	Correct	Language Tape	Correct
Library	Library	Correct	Library	Correct
Loan Item	Loan item	Correct	Loan item	Correct
Section	Section	Correct	Section	Correct
Member Card	Membership Card	Correct		Missing
	Author	Extra	Membership number	
	Member	Extra	Bar code reader	Incorrect
	Membership	Extra	Item	Extra
	Birth	Extra	Member	Extra
	Name	Extra	Loan	Extra
	Record	Extra	Subject section	Extra
	Date	Extra	Someone	Extra
	Type	Extra		
	Update	Extra		
	Bar Code Reader	Extra		
	Detail	Incorrect		
	Item	Extra		

Table 2a Library Specification Results			
Correct	**Incorrect**	**Missing**	**Extra**
7	1	0	11

$$recall = \frac{7}{7+0} = 1$$

$$precision = \frac{7}{7+1} = 0.875$$

$$over - specificat\ ion = \frac{11}{7+1} = 1.375$$

New Metrics Application:

$$precision = \frac{7}{19+0} = .037$$

Table 2b CM Builder Library Specification Results			
Correct	**Incorrect**	**Missing**	**Extra**
6	1	1	5

$$recall = \frac{6}{6+1} = 0.86$$

$$precision = \frac{6}{6+1} = 0.86$$

$$over - specification = \frac{5}{6+1} = 0.71$$

New Metrics Application:

$$precision = \frac{6}{13+1} = .042$$

Cinema Specification Results:

Table 3 Cinema Specification Results

Ideal Solution	Automated Solution	Classification
Film	Film	Correct
Screen	Screen	Correct
Weekly Showing Schedule	Timetable	Correct
Cinema Card	Cinema Card	Correct
Customer	Customer	Correct
Family Ticket		Missing
Ticket	Ticket	Correct
Sale		Missing
Card Sale		Missing
Seated Showing		Missing
Unseated Showing		Missing
Cinema Card Sale		Missing
	Date	Extra
	Day	Extra
	Debit Card	Extra
	DebitCardDetail	Incorrect
	Evening Screening	Extra
	Cinema Membership Card	Extra
	Board	Extra
	Film Distributor	Extra
	Following	Incorrect
	I	Incorrect
	Issue	Incorrect
	Listing	Extra
	Management Team	Extra
	Month	Extra
	Name	Incorrect
	Person	Extra
	Photograph	Extra
	Place	Extra
	Release Period	Extra
	Card	Extra
	Screening	Extra
	Seat	Extra
	Matinee Screening	Extra
	Student	Extra
	Subscription Charge	Extra
	Cinema Staff	Extra
	Cinema	Extra
	Type	Extra
	Week	Extra
	Weekend Screening	Extra
	Cinema Management Team	Extra
	Information	Incorrect

	Citizen	Extra
	Copy	Extra
	Credit	Extra
	Type	Incorrect
	Number	Incorrect

Table 3a Cinema Specification Results			
Correct	**Incorrect**	**Missing**	**Extra**
6	8	6	29

NOTE: need to change extra to 29 counted 'week', 'cinema' & 'cinema membership card' twice.

$$recall = \frac{6}{6+6} = 0.5$$

$$precision = \frac{6}{6+8} = 0.43$$

$$over - specificat\ ion = \frac{29}{6+6} = 2.42$$

New Metrics Application:

$$precision = \frac{6}{46+6} = 0.11$$

Taxi Specification Results:

Table 4 Taxi Specification Results				
Ideal Solution	**Automated Solution (without language inconsistency model)**	**Classification**	**Automated Solution (with language inconsistency model)**	**Classification**
Taxi Company	Company	Correct	Company	Correct
Passenger	Passenger	Correct	Passenger	Correct
Location	Location	Correct	Location	Correct
Shuttle	Shuttle	Correct	Shuttle	Correct
Taxi	Taxi	Correct	Taxi	Correct
Vehicle	Vehicle	Correct	Vehicle	Correct
	Driver	Extra	Driver	Extra
	Pick	Extra		
	Call	Extra	Call	Extra
	Destination	Extra		
	Entertainment Venue	Extra		
	Group	Extra		
	Hotel	Extra		
	Individual	Extra		

Table 4a Taxi Specification Results – (Without Language inconsistency Model)			
Correct	**Incorrect**	**Missing**	**Extra**
6	0	0	8

$$recall = \frac{6}{6+0} = 1$$

$$precision = \frac{6}{6+0} = 1$$

$$over - specification = \frac{8}{6+0} = 1$$

New Metrics Application:

$$precision = \frac{6}{14+0} = 0.43$$

Table 4b Taxi Specification Results - (With Language inconsistency Model)			
Correct	**Incorrect**	**Missing**	**Extra**
6	0	0	2

$$recall = \frac{6}{6+0} = 1$$

$$precision = \frac{6}{6+0} = 1$$

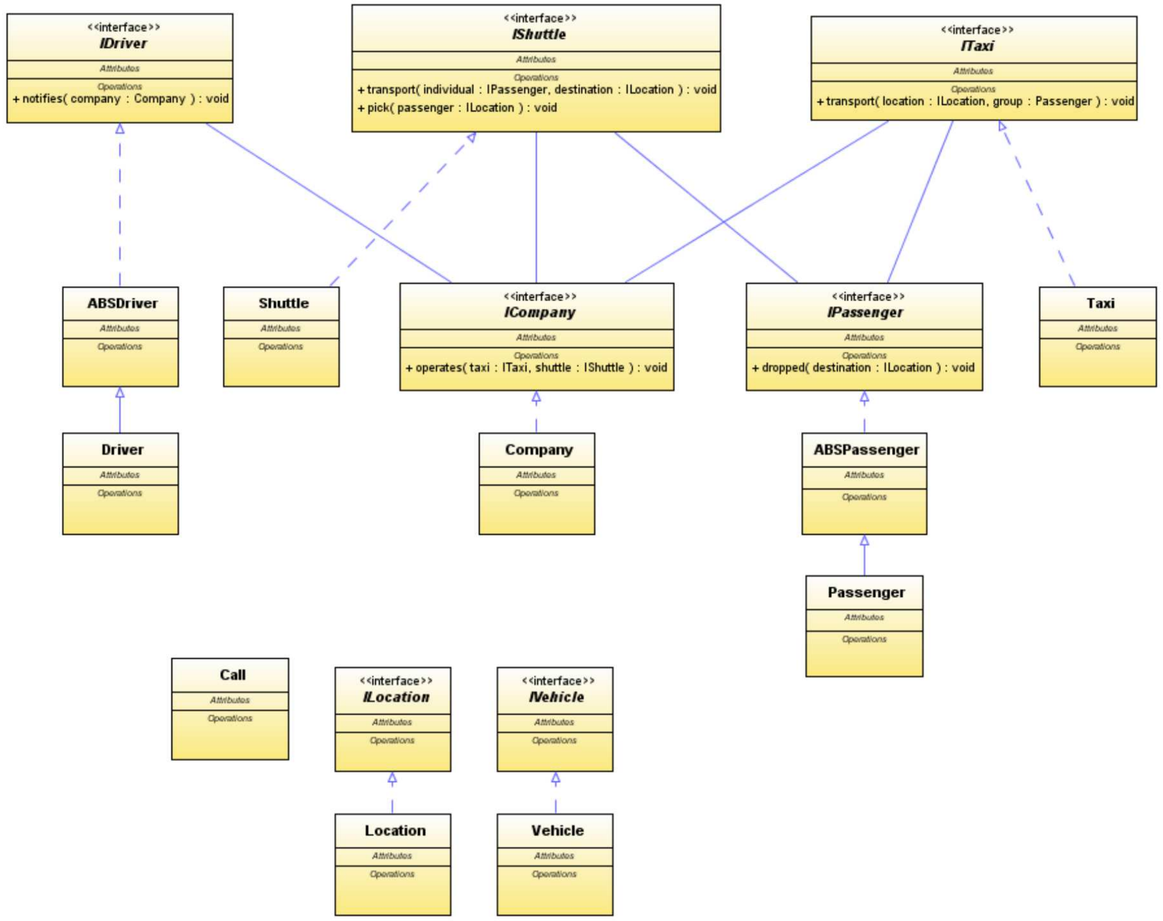
$$over - specificat\ ion = \frac{1}{6+0} = 0.16$$

New Metrics Application:

$$precision = \frac{6}{8+0} = 0.75$$

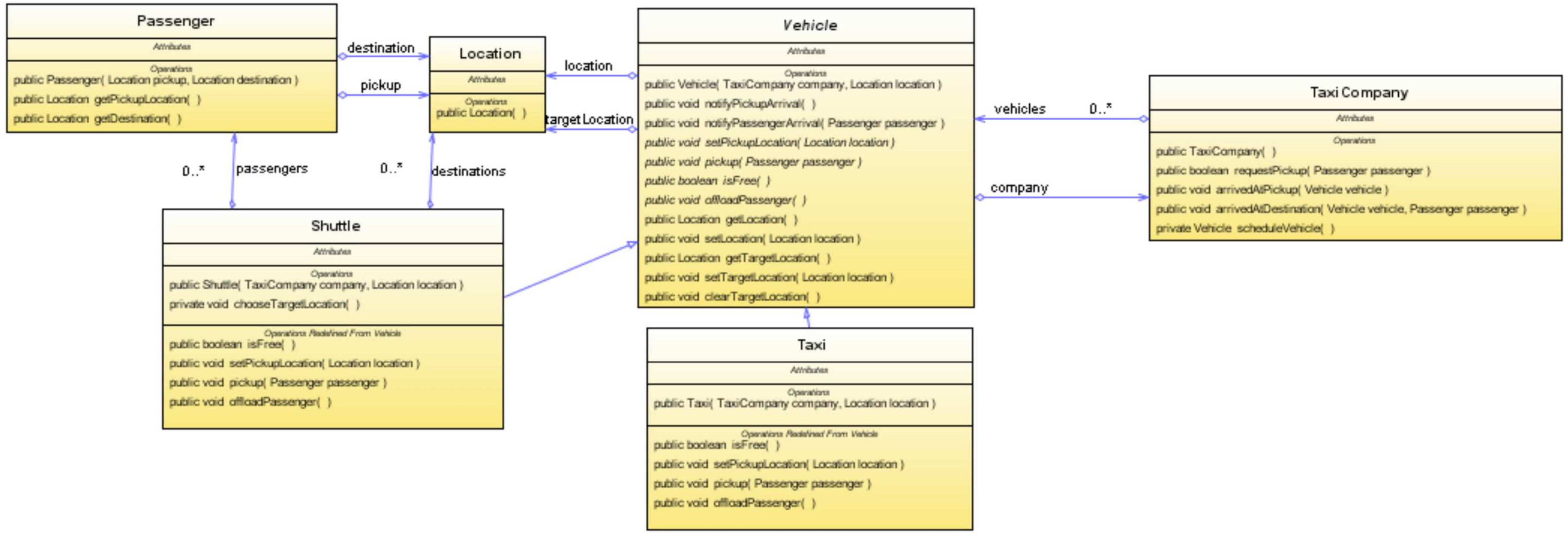
Appendix Three: Taxi Models

Taxi Model (Automated):



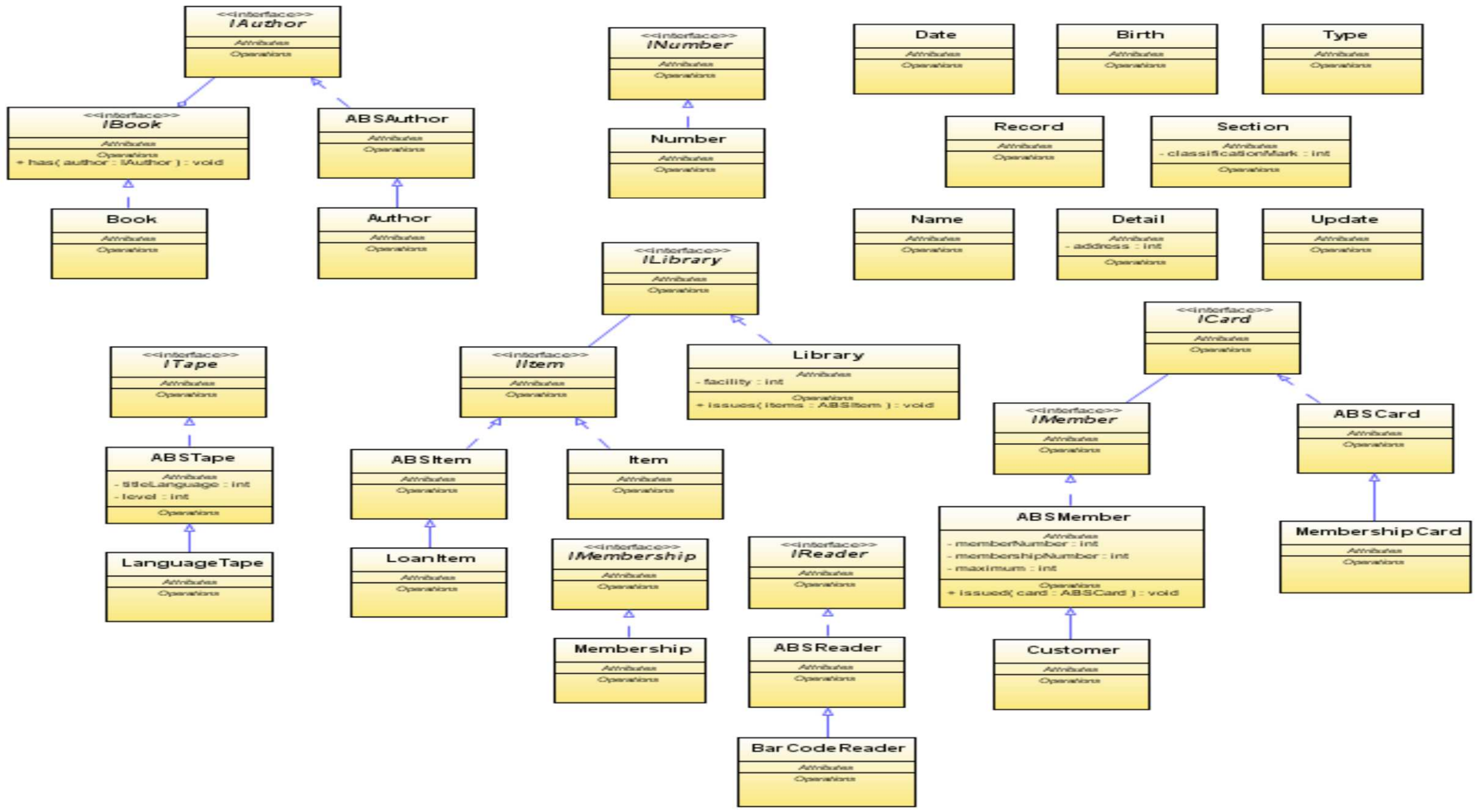
<<interface>> IDriver
Attributes
Operations
+ notifies(company : Company) : void
<<interface>> IShuttle
Attributes
Operations
+ transport(individual : IPassenger, destination : ILocation) : void
+ pick(passenger : ILocation) : void
<<interface>> ITaxi
Attributes
Operations
+ transport(location : ILocation, group : Passenger) : void
ABSDriver
Shuttle
<<interface>> ICompany
+ operates(taxi : ITaxi, shuttle : IShuttle) : void
<<interface>> IPassenger
+ dropped(destination : ILocation) : void
Taxi
Driver
Company
ABSPassenger
Passenger
Call
<<interface>> ILocation
<<interface>> IVehicle
Location
Vehicle

Taxi Model (Human):

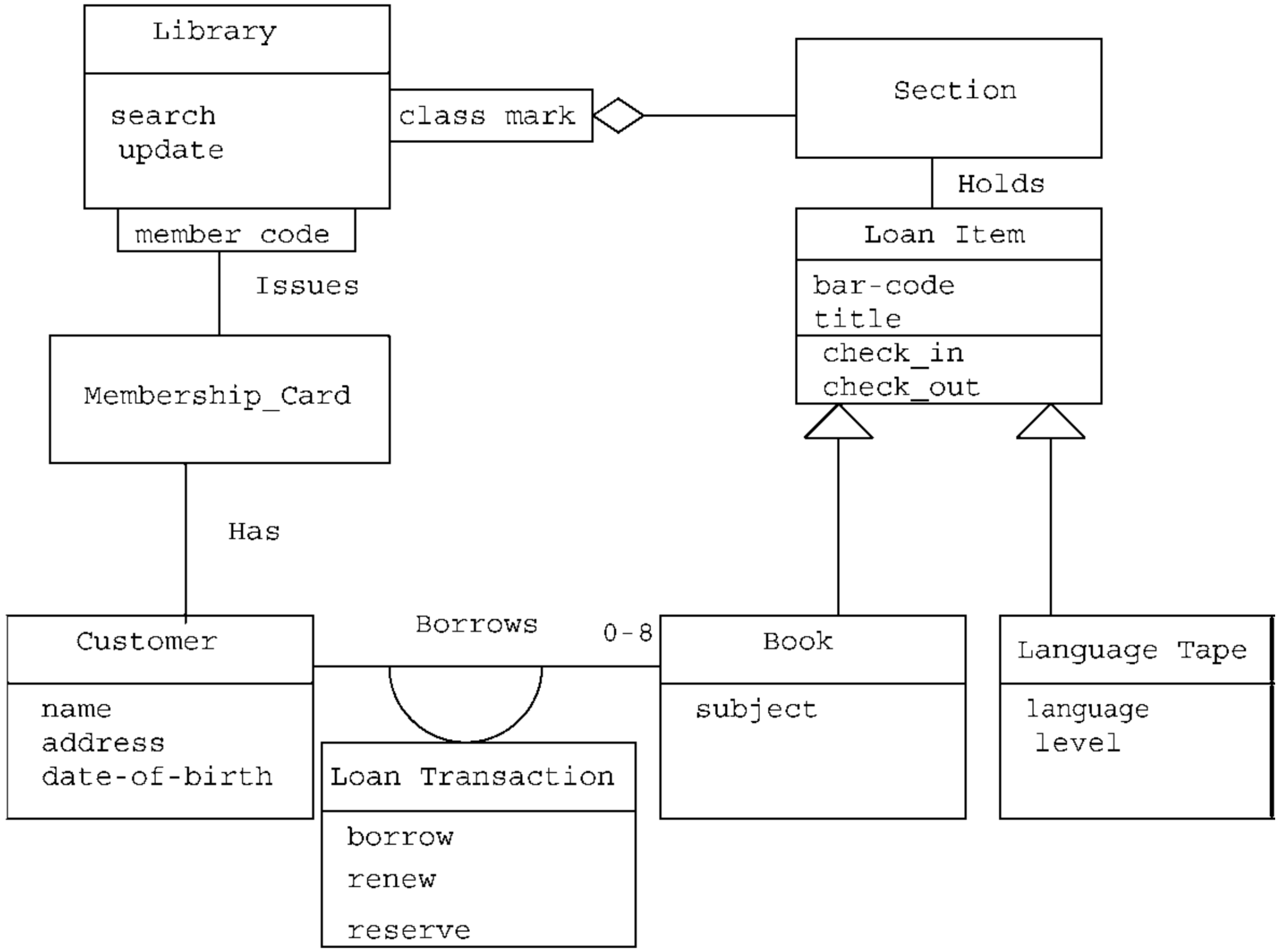

Passenger
Attributes
Operations
public Passenger(Location pickup, Location destination)
public Location getPickupLocation( )
public Location getDestination( )
destination
pickup
Location
Attributes
Operations
public Location( )
location
targetLocation
Vehicle
Attributes
Operations
public Vehicle(TaxiCompany company, Location location)
public void notifyPickupArrival( )
public void notifyPassengerArrival(Passenger passenger)
public void setPickupLocation(Location location)
public void pickup(Passenger passenger)
public boolean isFree( )
public void offloadPassenger( )
public Location getLocation( )
public void setLocation(Location location)
public Location getTargetLocation( )
public void setTargetLocation(Location location)
public void clearTargetLocation( )
vehicles
0..*
company
Taxi Company
Attributes
Operations
public TaxiCompany( )
public boolean requestPickup(Passenger passenger)
public void arrivedAtPickup(Vehicle vehicle)
public void arrivedAtDestination(Vehicle vehicle, Passenger passenger)
private Vehicle scheduleVehicle( )
0..*
passengers
0..*
destinations
Shuttle
Attributes
Operations
public Shuttle(TaxiCompany company, Location location)
private void chooseTargetLocation( )
Operations Redefined From Vehicle
public boolean isFree( )
public void setPickupLocation(Location location)
public void pickup(Passenger passenger)
public void offloadPassenger( )
Taxi
Attributes
Operations
public Taxi(TaxiCompany company, Location location)
Operations Redefined From Vehicle
public boolean isFree( )
public void setPickupLocation(Location location)
public void pickup(Passenger passenger)
public void offloadPassenger( )

Appendix Four: Library Models

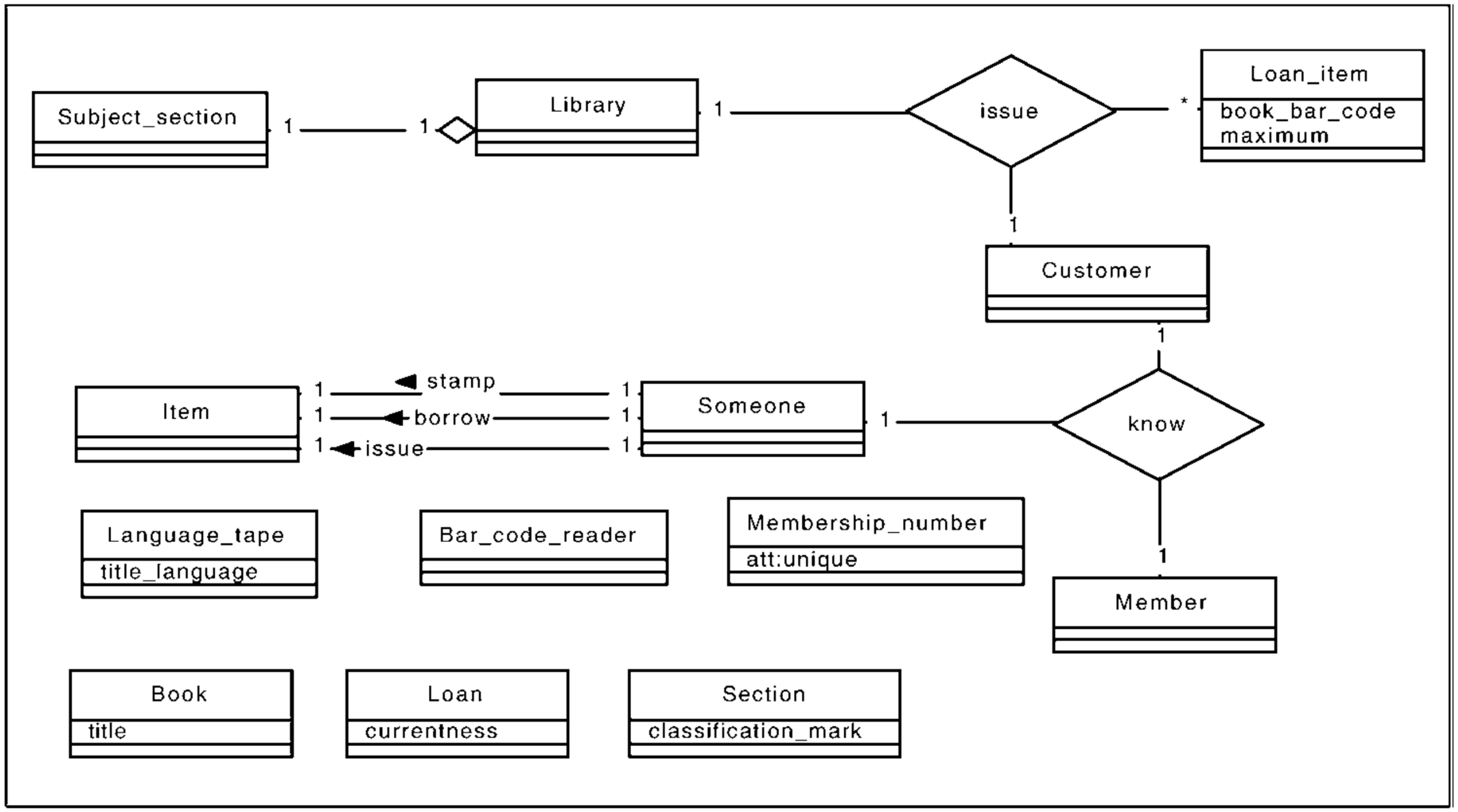
Library Model (Automation):



Library Model (Human Model):



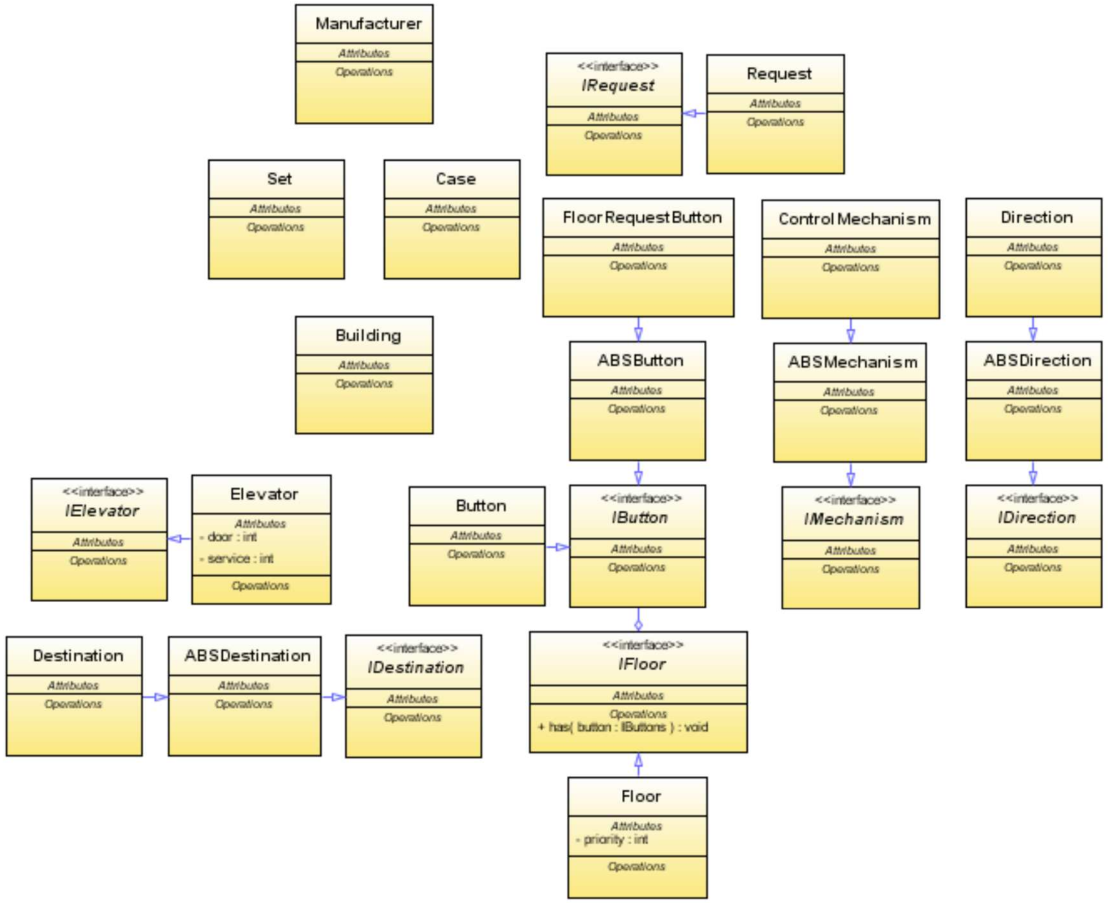
Library Model (CM-Builder):



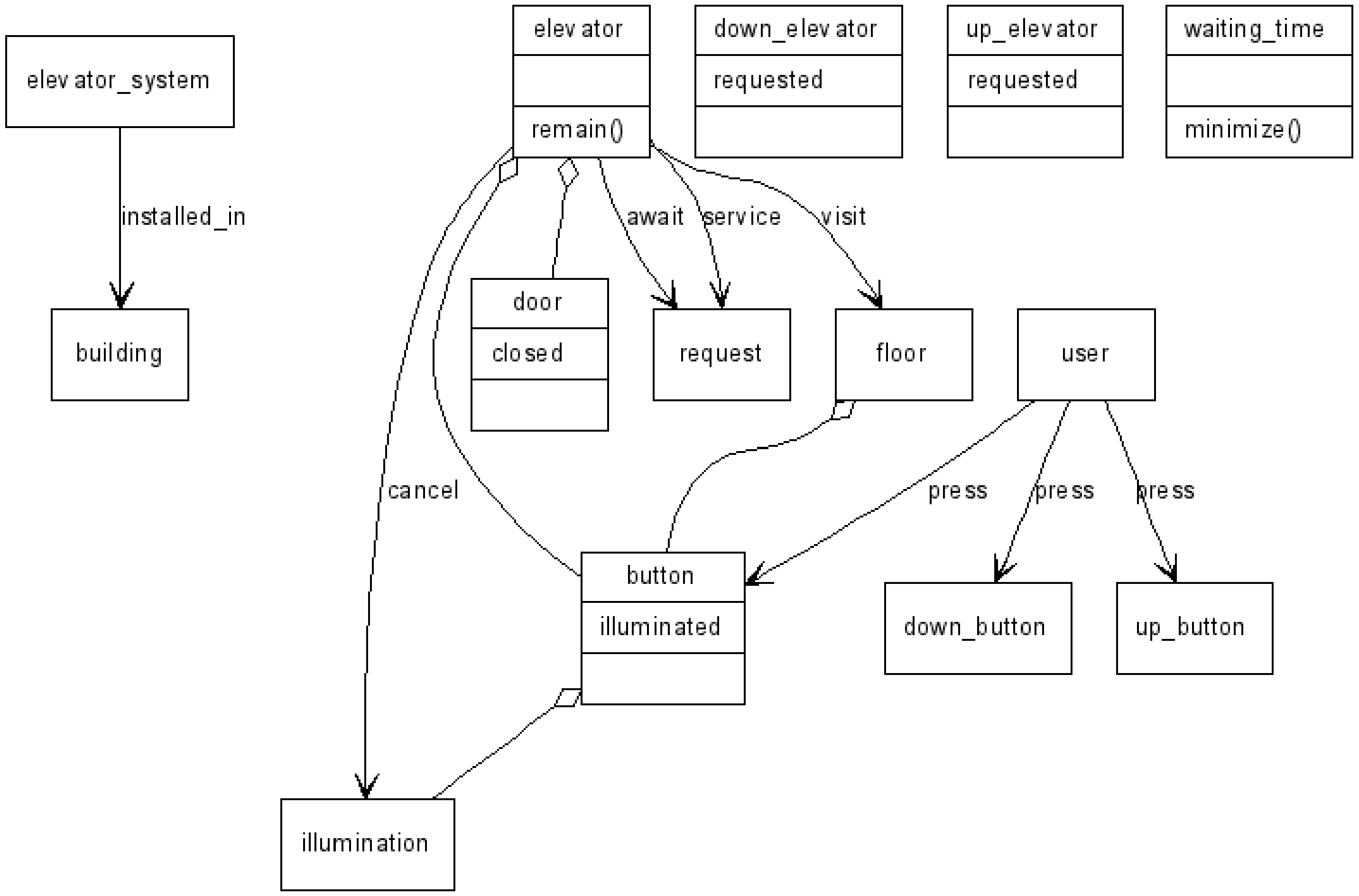
```
CM-Builder, CLD - CLD00001.DAT
Library1 HARMAIN1          7-Dec-99 17:23:49
Page 1 of 1
```

Appendix Five: Lift Models

Lift Model (Automation):

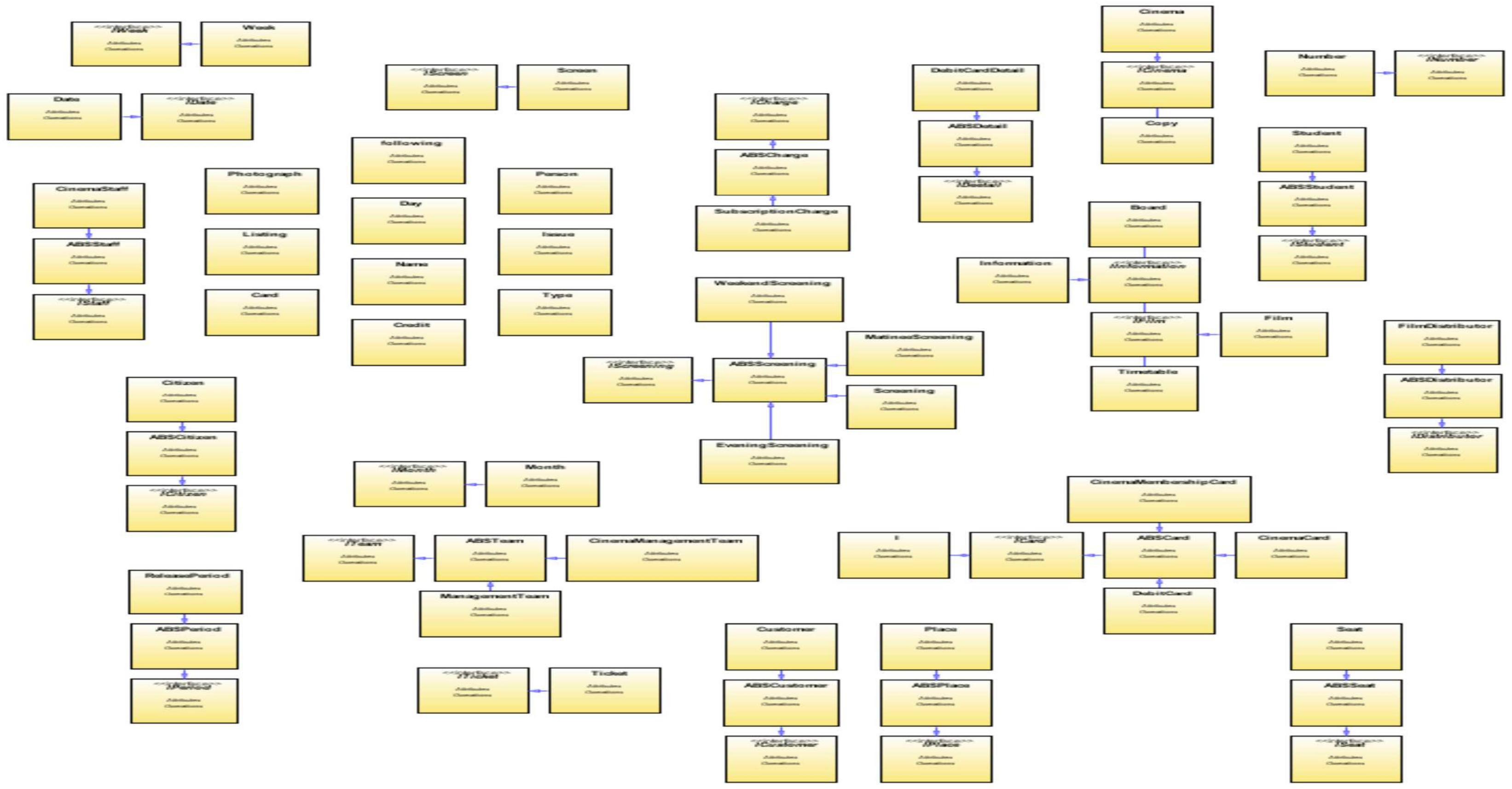


Lift Model (Drowser Automation):

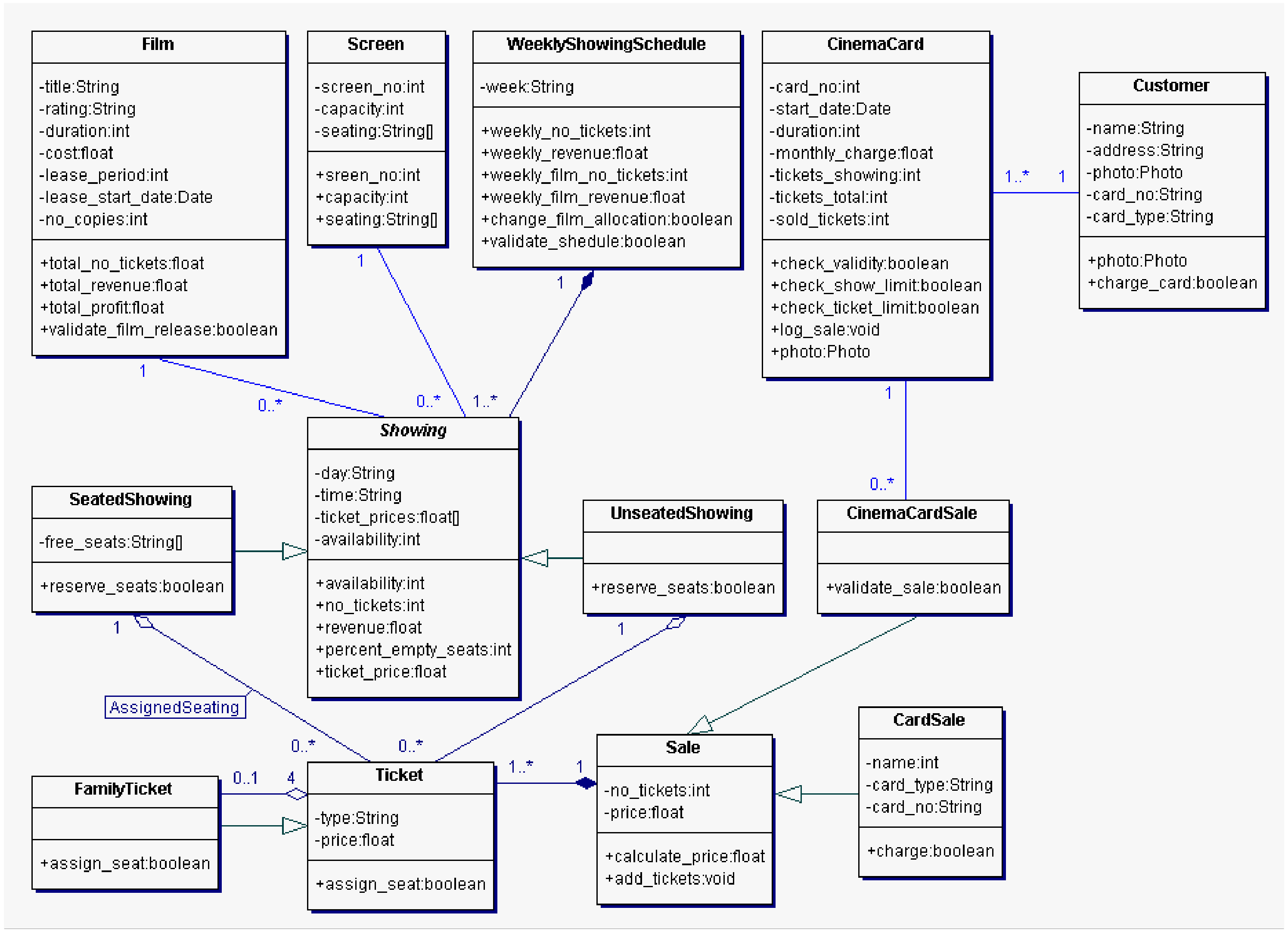


Appendix Six: Cinema Models

Cinema Model (Automation):



Cinema Model (Human):



Appendix Seven: Frequency of Occurrence

Frequency of Occurrence:					
Specification / **Type**	**Library**	**Cinema**	**Taxi**	**Lift**	**Average Count**
'X of Y'	7	21	1	2	**7.75**
Boundary/Under-Specification	1	6	1	0	**2**
Synonym	2	3	7	0	**3**
Common Phrase	1		0	0	**0.25**
'Has'	2	2	1	3	**2**
VP && PP	8	14	2	4	**7**
Existential	1	3	0	0	**1**
General - Introductory - Irrelevant Terms	2	7	0	2	**2.75**

References:

Craig Larman, editor. Applying UML and Patterns: An introduction to Object-Oriented Analysis and Design. Prentice-Hall PTR, 1998

J. Martin and James Odell. Object-Oriented Methods: A Foundation. Prentice-Hall, Englewood Clifs, New Jesey. 1995

Bertrand Meyer. Object-Oriented Software Construction. ISE Inc., second edition, 1997.