FRAMEWORK TO MANAGE LABELS FOR E-ASSESSMENT OF DIAGRAMS

A Thesis submitted for the degree of Doctor of Philosophy

by

AMBIKESH JAYAL

School of Information Systems, Computing & Maths,
Brunel University

March 2010

Abstract

Automatic marking of coursework has many advantages in terms of resource benefits and consistency. Diagrams are quite common in many domains including computer science but marking them automatically is a challenging task. There has been previous research to accomplish this, but results to date have been limited. Much of the meaning of a diagram is contained in the labels and in order to automatically mark the diagrams the labels need to be understood. However the choice of labels used by students in a diagram is largely unrestricted and diversity of labels can be a problem while matching.

This thesis has measured the extent of the diagram label matching problem and proposed and evaluated a configurable extensible framework to solve it. A new hybrid syntax matching algorithm has also been proposed and evaluated. This hybrid approach is based on the multiple existing syntax algorithms.

Experiments were conducted on a corpus of coursework which was large scale, realistic and representative of UK HEI students. The results show that the diagram label matching is a substantial problem and cannot be easily avoided for the e-assessment of diagrams. The results also show that the hybrid approach was better than the three existing syntax algorithms. The results also show that the framework has been effective but only to limited extent and needs to be further refined for the semantic stage.

The framework proposed in this Thesis is configurable and extensible. It can be extended to include other algorithms and set of parameters. The framework uses configuration XML, dynamic loading of classes and two design patterns namely strategy design pattern and facade design pattern. A software prototype implementation of the framework has been developed in order to evaluate it.

Finally this thesis also contributes the corpus of coursework and an open source software implementation of the proposed framework. Since the framework is configurable and extensible, its software implementation can be extended and used by the research community.

Keywords: e-learning, e-assessment, automated marking

Contents

List of Figures

List of Tables

Acronyms

e-learning Electronic Learning

e-assessment Electronic Assessment

XML Extensible Markup Language

UML Unified Modelling Language

CASE Computer Aided Software Engineering

VLE Virtual Learning Environment

XMI XML Metadata Interchange

API Application Programming Interface

NLP Natural Language Processing

HCI Human Computer Interface

DTD Document Type Definition

UK HEI United Kingdom Higher Education Institution

OKI Open Knowledge Initiative

IMS QTI IMS Question Test Interoperability

Acknowledgments

First and foremost I would like to thank Professor Martin Shepperd for providing me the opportunity to pursue PhD research. Without his support this PhD would never had even started. I would also like to thank him for the supervision, support, encouragement and guidance throughout my PhD studies. I have been very fortunate to have him as my supervisor. Apart from the knowledge about the subject, I have also learnt from him the qualities of a good academic. I hope that I will embed some of these qualities in my future academic life. I would also like to thank Dr. Michelle Cartwright and Dr. Steve Counsell for the excellent supervision. I would also like to thank Brunel University for providing me the financial support to pursue PhD.

Secondly, I would like to thank my family and friends (Supayanee Watchararattanawalee, Carolyn Mair, Alex DeWitt, Navonil Mustafee, Yogesh Dwivedi and Banita Lal). I would also like to thank Dr. Kate Dunton from Learning and Teaching Unit, Julie Whittaker and all the support staff at DISC.

Finally, thanks to all those who helped me throughout this thesis.

Dedication

I dedicate this thesis to my dearest mother and father who have brought me up with immense love, care and blessings.

Research publications

The following is a list of four related publications directly arising from this thesis.

- Jayal, A. and Shepperd, M. The Problem of Labels in E-Assessment of Diagrams. *ACM J. Educ. Resour. Comput.* 8, 4 pp1-13, 2009, Citation count=3

- Jayal, A. and M. Shepperd (2009). An improved method for label matching in e-assessment of diagrams. Innovation in Teaching and Learning in Information and Computer Sciences (ITALICS), *Electronic journal of the UK Higher Education Academy*, 8(1), 2009.

- Jayal, A. and Shepperd, M.J. An evaluation of e-learning standards, *5th International Conference on E-Governance*, Hyderabad, India, December 28-30, 2007. Citation count=2

- Jayal, A. Cartwright, M. and Shepperd, M.J. Premark: A System Designed to Organising Course Work for Assessment, *5th International Conference on E-Governance*, Hyderabad, India, December 28-30, 2007.

In addition to the above, I have also published following related e-learning papers.

- Shepperd, M.J. and Jayal, A. Experiences of Introducing Group Projects to Computing Degrees, *35th International Conference on Improving University Teaching (IUT)*, Washington D.C., USA, June 30, 2010.

- Lauria, S., Jayal, A., Tucker, A., Swift, S. Python for Teaching Introductory Programming: a Quantitative Evaluation, *United Kingdom Higher Education Academy 10th Programming Workshop*, University of Brighton, UK, March 30, 2010.

Chapter 1

Introduction

1.1 E-learning

E-learning comprises any type of learning activity that is based upon some electronic media. According to Wentling (Went 00), “E-learning is the acquisition and use of knowledge distributed and facilitated primarily by electronic means”. E-learning includes the use of electronic means for all the aspects of teaching, for example creation and delivery of course content, electronic submission and marking of coursework, delivery of coursework assessment results, interaction with and among students through discussion boards, etc.

E-learning has practical advantages, for example resource benefits in terms of time and effort (Higg 02a). Virtual learning Environments VLE(Chin 03) are software tools for e-learning. A survey conducted by Brown (Brow 08) in which 74 of the 164 UK HEI institutions participated shows that 91% of the participating UK HEI institutions used a VLE. It also shows that the most popular VLE among UK HEIs are Moodle(Cole 05), Blackboard(Blac 10) and WebCT(Clar 02). So e-learning is popular and widely adopted among UK HEIs.

E-learning also has two disadvantages. The first one is that sometimes the e-learning tools impose new processes on lecturers resulting in extra administrative work (Jaya 07a). This may hinder the adoption of these e-learning tools. This problem may be solved by introducing middleware components that sit between the lecturer and the e-learning tool

Educational Context	**Purpose**	**Time**
Diagnostic	To ascertain knowledge level of learner	Before or after learning programme
Formative	To provide feedback to the learner	During learning programme
Summative	To formally grade performance of learner	During or end of learning programme

Table 1.1: Types of Assessment

and lessen the extra administrative work imposed by the e-learning tool. Such a system has been developed by Jayal (Jaya 07a).

The second disadvantage of having many e-learning tools is the issue of interoperability between them. This can be solved by defining standards for e-learning content and components. Recently there have been efforts to define standards for the e-learning content and components in order to make them interoperable and reusable with other e-learning tools. For example the standards Learning Object Metadata (Hodg 05; CETI 08), IMS standards (Cons 10) and SCORM (Lear 04) define the specifications for the e-learning content in order to make them reusable and interoperable. Similarly the OKI standards (Tech 07) define the service interfaces for the various components of e-learning so that implementation of the components can be shared between the different e-learning systems. A study by Jayal (Jaya 07b) shows that although these standards may help interoperability, accessibility and reusability of the e-learning content and e-learning components, they have limited adoption at UK higher education institutions.

1.2 E-assessment

Assessment is an integral part of the learning process. According to Brown (Brow 97a) the assessment word was extracted from the phrase "*ad sedere*" which means to sit down beside and is primarily concerned with providing guidance and feedback to the learner. Assessment can involve various activities, for example preparing questions for the examination, delivering the exam, marking the students answers, providing feedback to the students and detecting plagiarism. It is used at different stages of the learning process as summarized in the Table 1.1 (Khed 05). Before the learning process begins, the diagnostic assessment can be used to ascertain the knowledge of the learner. During the learning process the formative assessment can be used to provide feedback to the learner about their level of understanding of a concept. On completion of the learning process, summative assessment can be used to provide grades to the students.

E-assessment or electronic assessment refers to automating the process of assessment. There has been growing interest within the e-learning community for e-assessment (Brow 97b). E-assessment may involve coursework delivery by the lecturer, coursework submission by the students, coursework marking and presenting feedback to the students. A software tool for e-assessment is known as an e-assessment system. Some of the important features of an effective e-assessment system should be user friendliness, efficiency, reliability, consistency, effective plagiarism detection, interoperability, adherence to e-learning standards for example IMS QTI (Cons 10) and providing timely feedback to students.

1.3 Motivation for research

The e-assessment of coursework brings both pedagogic and practical benefits (Higg 02a). It provides resource benefits in terms of time and effort, consistency in marking and quicker feedback. Additionally, it may also be used for plagiarism detection.

Automatically marking coursework is one of the factors that can help provide feedback quickly to the students. One important area is the e-assessment of diagrammatic coursework as diagrams are commonplace in many subjects such as computer science. Previous re-

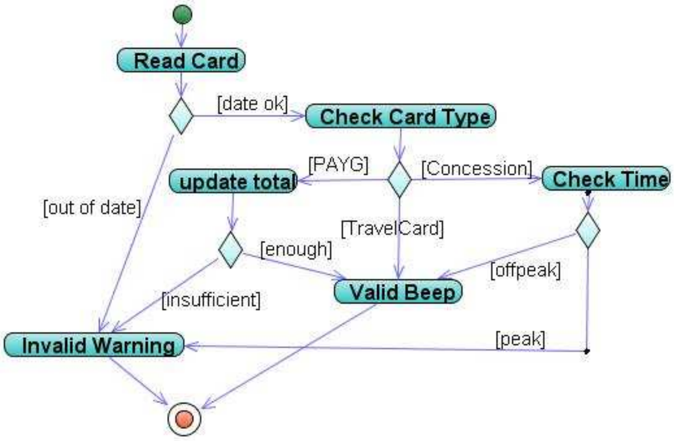


Figure 1.1: Sample diagram

search work to automatically mark coursework has mostly targeted objective type questions (Clar 02; Feng 06), free response text based questions (Vale 03; Pere 05; Kere 05), mathematics based questions (Poll 02; Beev 02) and computer programming questions (Ala 05). However, there is now growing interest in diagram based questions (Hogg 98; Higg 02b; Thom 04; Tsel 05) but results to date have been limited.

Diagrams are difficult to automatically mark because of problems such as the diagram being malformed or possessing missing or extraneous features (Smit 04). Another problem is the diversity of labels used by the students. Typically much of the meaning of the diagram resides in the diagram labels. For example, the sample diagram shown in Figure 1.1 can not be interpreted without meaningful labels. However, the choice of label by the students is largely unrestricted and so there are problems of synonyms, homonyms, abbreviations, misspellings, labels comprising variable numbers of words and so forth. This means a correct student diagram may utilise labels that differ yet are syntactically or semantically equivalent to the labels in the model solution. Previous research has reported label matching to be a problem for e-assessment of diagrams (Jaya 09b; Thom 09; McGe 05).

A systematic review of all the diagram e-assessment systems shows that although there are benefits of e-assessment of diagrams there are only six systems for e-assessment of diagrams, none of which are open source while two of them are commercial[1]. For e-assessment, the structure of a diagram and the labels present in it need to be compared with that of the model solution. The structure of a diagram can be compared using graph theory while the labels can be compared using natural language processing techniques. Existing e-assessment systems for diagrams focus on structure (Tsel 07). This Thesis complements existing work by empirically measuring the problem of diversity of labels used by students and exploring various syntax and semantic techniques to address this problem.

1.4 Research aims

The following are the five research aims of this Thesis.

1. **Empirically measure the extent of the diagram label matching problem.**
   This Thesis will be empirically measuring the extent of the diagram label matching problem.

2. **Propose and evaluate a framework to match diagram labels.**
   This Thesis will propose a framework to match the labels in diagrams and evaluate the proposed framework on a coursework at a UK HEI.

3. **Propose and evaluate a new hybrid syntax matching algorithm.**
   This Thesis will proposes and evaluate a new hybrid syntax matching algorithm based on the multiple existing syntax algorithms.

4. **Provide a corpus of coursework for further experiments**
   Considering the time and effort required to collect coursework for experiments, the corpus of coursework will be made available to the research community for further experiments.

5. **Develop a prototype open source implementation of the framework.**
   This Thesis will provide an open source implementation of the proposed framework

[1] The source code for the non-open source tool developed by Open University (Thom 04) is available informally from the authors.

which can be used by the research community for further implementation and evaluation.

1.5 Thesis roadmap

Chapter 2 of the Thesis describes the existing systems for e-assessment of diagrams. In Chapter 3 a framework to match the diagram labels for e-assessment is proposed. The chapter 3 starts by first discussing the design research methodology used in this Thesis. Then the functional and non-functional requirements of the framework are explained. The six different stages of the framework which fulfil the functional requirements are then explained. Finally, the design of the software prototype implementation of the framework will be explained. This design fulfils the non functional requirements of the framework. In Chapter 4 the corpus of coursework collected for the experiments to evaluate the framework and the results of these experiments is discussed. The software prototype implementation of the framework explained in the Chapter 3 is used to carry out these experiments on coursework from a UK HEI. These results show that the framework is effective, but there is a need for more effective semantic algorithms as the syntax algorithms are effective only to a limited extent. Chapter 5 presents the conclusion of the Thesis, limitations and directions for the future work.

Chapter 2

Literature Review

2.1 Introduction

This chapter explains the techniques for marking the diagrams by existing e-assessment systems and identifies those areas needing further work.

2.2 Existing systems for automatically marking a diagram

To find the existing systems that automatically mark the diagrams, a systematic search of the literature was conducted in May 2008 on five bibliographic databases namely IEEE/IET Electronic Library, ACM Digital Library, ScienceDirect, Scopus and SpringerLink. The main search terms used were “e-assessment”, “eassessment”, “computer aided assessment”, “computer-aided assessment”, “computer assisted assessment” and “computer-assisted assessment”. This retrieved almost 900 papers that were then hand-checked for relevance as a result of which six e-assessment systems for diagrams were identified. The spreadsheet containing details of these papers is available at (Jaya 10). The databases were searched again in February 2010 and no new e-assessment system for diagrams was found. Tables 2.1 and 2.2 summarize the six e-assessment systems for diagrams. The next two sections describe the techniques used by these six systems to mark the diagrams and match the labels in the diagrams.

Reference	**Tool name**	**Source**
eA1	University of Teesside (UK) Automated Student Diagram Assessment System	(Hogg 98)
eA2	Nottingham University (UK) CourseMaster	(Higg 02b) (Higg 06)
eA3	Open University (UK) DEAP Diagram Tool	(Thom 04) (Thom 07b)
eA4	Manchester University (UK) Assess By Computer (ABC)	(Tsel 05)
eA5	Loughborough University (UK) Diagram Tool	(Batm 06)
eA6	Canterbury University (NZ) KERMIT Tool	(Sura 02)

Table 2.1: Tools for e-Assessment of Diagrams

2.3 Approaches for marking diagrams

The following five approaches have been used for automatically marking diagrams in the existing e-assessment tools.

- **Object Oriented Metrics**

  This approach involves calculating the various Object Oriented metrics for cohesion, coupling etc. and using them to mark the diagrams. The marking tool for the Object Oriented diagrams in the CourseMaker system (Higg 02b) uses metrics to mark the diagrams for correct classes, relationship and completeness. This approach is limited to Object Oriented diagrams only and can not be used for other types of diagrams.

- **Graph Isomorphism**

  In this approach a diagram is treated like a graph with the node representing an activity or entity depending on the type of the diagram and the edge representing the relationship between them. Marking is done by searching for graphs or sub graphs in

Ref.	**Marking technique**	**Label similarity technique**	**Label diversity problematic?**
eA1	Object Oriented Metrics	Manual intervention (students map the labels in their diagram to those in the model solution)	Unknown
eA2	Object Oriented Metrics	Unknown	Unknown
eA3	Local Metrics (label, type)	Edit Distance algorithm, synonyms, punctuation, hyphenation and stemming	Yes
eA4	Graph Isomorphism, Local Metrics (label, type)	Edit Distance algorithm	Yes
eA5	human marking	Manual intervention (students are required to choose from a list of labels presented to them)	Unknown
eA6	human marking	Manual intervention (students highlight the text in the problem specification and this highlighted text is then used as a label)	Yes

Table 2.2: e-Assessment Tool Details

the student diagram isomorphic to the model solution. The advantage of this approach is that it is generic and can be applied to wide variety of diagrams. This approach is not scalable because of the computational power required to find isomorphic components (Sedg 03) but this is not a major problem for the e-assessment domain considering the expected size of the the student diagrams (Tsel 05). This approach has been implemented in the ABC system developed by the University of Manchester and is reported to have computationally worked well even for large artificial student diagrams (Tsel 05).

The ABC system first finds the maximal sub graphs in the student diagram that are isomorphic to the model solution and then produces a list of relabellings required for the vertexes. This approach has two disadvantages. The first is that this approach will not work well if the student diagram has some edges missing. The second is that this approach treats diagrams purely as graphs whereas diagrams can have richer associated information; for example the label in the nodes and edges, type of nodes etc. (Tsel 05). Two diagrams may be topologically equivalent yet have differing semantics. So this approach alone may not be sufficient for marking diagrams. The ABC system (Tsel 05) has used it in combination with the Local Metrics approach explained in the next paragraph.

- **Local Metrics**

  This approach complements the graph isomorphism approach by taking into account the richer information associated with nodes and edges in a diagram; for example the type and label of the nodes. The ABC system uses this approach in conjunction with the graph isomorphism approach (Tsel 05). For each node in the diagram, a local metric object containing five attributes is created[1]. The first attribute is the degree which is equal to the number of the edges to the node. The second attribute is the type of the node which is a string representing the domain specific name of the

[1] According to the developers of ABC system these metrics are subject to change and only discussed informally in the paper (Tsel 05). The exact information on the metrics could not be found as the tool is commercial in nature.

node for example entity or the way node is drawn for example circular, rectangular or diamond shaped. The third attribute is the number and type of the adjacent nodes. The fourth attribute is the number of incident connectors, their types and the label in the connector. The fifth attribute is the label in the node. Each attribute of the local metric object is given a weight between 1 and 4. All the attributes are either number or string labels. The string labels are considered the same if the edit distance between them is more than a certain threshold. The lecturer also has the option to exclude any attribute from the marking process using a check box. The similarity score of two local metrics is the weighted average of the similarity score of each of their attributes.

The ABC system also has a weight manager which can automatically calculate the weights for the attributes of the local metric object. It works by calculating marks using all the possible combinations of weights for each attribute of the local metric object and choosing the weight that maximizes the marks.

The local metric for a node in the student diagram is matched with the local metric for each node in the model solution. The node in the model solution that gives the highest similarity score is considered to be matched and is assigned to the node in the student diagram. This process is repeated for each node in the student diagram but only with the unassigned nodes in the model solution so that a node in the model solution is not matched to more than one node in the student diagram. Since only the unassigned nodes in the model solution are considered for matching the order in which the nodes in the student diagram are selected for matching will effect the nodes in the model solution that they are matched to. The literature available at (Tsel 05; Tsel 07) does not make clear the order in which the nodes in the student diagram are selected up matching[2].

Experiments were carried out using 15 students' coursework from a real exam. The results of these experiments show that, on average the machine marks were lower than the human marks. The average marks awarded by the machine was 57% compared to

[2] The tool is commercial in nature and owned by the Assessment21 Limited.

62% awarded by the human with a standard deviation of 14.7%.

- **GREE**

  Tselonis has introduced a domain independent marking technique called "Dynamically extendable AND/OR trees GREE" (Tsel 07). It has a modular scoring strategy. This technique is the subject of a patent application by Assessment21 Ltd. who have developed the eA4 system.

- **Minimal Meaningful Unit MMU**

  This approach has been adopted by the eA3 system and is based on the concept of meaningful unit (MU) and minimal meaningful unit (MMU) (Thom 04; Thom 07b; Thom 07a). A diagram consists of smaller units which have their own meaning. For example in the Figure 2.1 the state represented by a rectangle with the label "read card" is a unit having it's own meaning; similarly the arrow labelled "date ok" together with the the diamond decision box and the state "check card type" at each end is a unit having it's own meaning. Such smallest units of a diagram which have their own meaning are called minimal meaningful units (MMU). A set of MMUs form a meaningful unit MU. A diagram consists of one or more MUs which in turn consist of one or more MMUs. For example the diagram in the Figure 2.1 has six MMUs of type state, one MMU of type start state, one MMU of type end state and sixteen (nine having labels) MMUs of type arrow.

  This approach takes a raster based image as input and outputs feedback and marks using five stages namely segmentation, assimilation, identification, aggregation and interpretation. The first two stages segmentation and assimilation identify the basic diagram components for example boxes, arrows etc. from the raster based image. The identification stage using the domain knowledge finds all the MMUs in a diagram. All the MMUs in a student diagram are compared with all the MMUs in the model solution to find the matching MMUs. Once the matching MMUs are found in the student diagram, the marks are allocated as per the marking scheme which can allocate different marks to different MMUs in the model solution. An enhancement to this

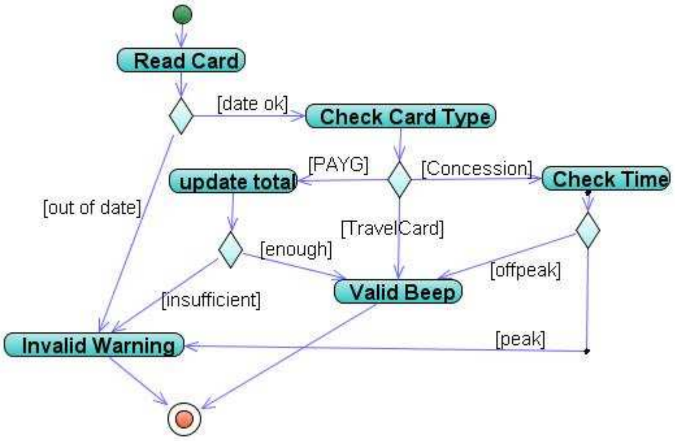


Figure 2.1: Diagram for minimal meaningful unit (MMU)

process is the introduction of the aggregation stage which combines different MMUs to form units with higher level meaning. This aggregation stage has not yet been implemented by this tool (Thom 07b).

The marking approach does some initial lexical processing, removes the stopwords (Wiki 10f) from labels and uses the edit distance algorithm to calculate a numeric similarity measure between them. It also uses a manually prepared list of synonyms. This approach recognises that the edit distance only works well with simple labels. Dealing with complex labels consisting of more than one word requires further processing (Thom 07b). To deal with complex labels, in an evaluation of this tool using the entity relationship diagrams, noun phrases consisting of noun and modifiers were extracted from the entity labels and verb phrases were extracted from the relationship labels. These noun and verb phrases were then used to calculate the similarity score. The approach also uses the structure of the diagram and domain specific rule to find synonyms. For example if there is an entity in the student diagram and a yet unattached entity of the same type in the model solution, then these two should be considered as matched. This approach also recognises the use of hyponyms as synonyms by students for example “prereq course” being used as synonym for “course”.

This eA3 system was first evaluated with a sample of 20 answers from a real exam and then with 11 answers from a mock exam. In the first experiment, the average difference between the machine and human marks was 2.5 (10.5%) with a standard deviation of 1.54 marks. In the second experiment with the mock exam, the average difference between human and machine mark was 12%. The research reports that the main reasons for discrepancies were spelling errors, thesaurus deficiencies and abbreviations. When these deficiencies were manually rectified, the average difference between the human and machine mark was reduced from 12 to 4.73. So this research recognises the problems in matching the labels and shows that by rectifying them manually improves the accuracy of the automatic marker. This research also acknowledges the need for large scale experiments on real coursework in order to determine the accuracy of the automatic marker.

Large scale experiments were then carried out using the eA3 system on 594 entity relationship diagrams drawn in a real but supervised examination (Thom 07b). Out of 594 diagrams, 200 were used as training set to detect bugs and set the weights and thresholds, while the remaining 394 were used for the actual evaluation. The results show that in 91% of diagrams there was less than 7% difference between the human and machine marks while in 69% of the diagrams there was no difference. The system was evaluated again on a set of 30 more complex entity relationship diagrams. The model solution in this experiment was more complex because apart from containing entities (represented by boxes) and relationships (represented by lines) it also contained an additional type of minimal meaningful unit (MMU) for entity supertype-subtype relationship (represented by a box wholly containing another box). The results show that there was no difference between the human and machine mark in 36.7% of the diagrams, while there was a difference of just 7% marks (0.5 marks our of 7) between human and machine mark in about 96.7% of the diagrams. The difference in results of two experiments show that the performance of the system is encouraging, but the algorithms need refining.

- **Graph Transformation Approach**

  Although this approach, like the Graph Isomorphism approach treats the diagram as a graph, it differs in the way it marks the diagrams. In this approach, the student diagram is treated as a starting graph and model solution is treated as the final graph. An attempt is made to transform the student diagram into the model solution. The number of steps required for this transformation is treated as a measure of the distance between the two diagrams and used to mark them. The problem with this approach lies in selecting the transformation steps and the order they should be applied as it can effect the marks. This approach has not been used by any of six existing e-assessment tools found during the literature review conducted during this Thesis.

All of the above approaches of automatically marking the diagrams include matching the labels in student diagram with the labels in the model diagram. The next section explains the different approaches that have been used to match the labels for marking the diagrams.

2.4 Approaches to match labels for marking a diagram

Table 2.2 summarizes the approaches used by the e-assessment systems to match the labels. The systems eA3, eA4 and eA6 acknowledge that label matching is a problem for e-assessment of diagrams. Following are two quotes from the existing literature acknowledging the label matching problem.

The eA6 system reports “It is believed that the naming sometimes causes inconsistencies between student diagram and the referenced phrases” (Sura 02).

The developers of eA4 system McGee et al. (McGe 05) encountered a high degree of variation in the labels used by students for an objective two-word phrase in a technical domain and reports “If this nature and degree of variation is found even for an objective two-word phrase in a technical domain, in an open-book test with no time pressure, it is somewhat alarming to speculate what we may find when we begin to look at less constrained situations, such as (for example) language translation exercises.” (McGe 05).

The systems eA1, eA5 and eA6 use the manual intervention approach to deal with the label matching problem. In the eA1 system, the students can use the label of their choice but in the eA5 and eA6 systems the students are restricted to the labels present in the problem specifications. In the eA1 system, each student is presented with a list of labels within their diagram and the model solution. The students are then required to map the labels in their diagram to those in the model solution. In the eA6 system the students are required to highlight the text in the problem specification and this highlighted text is then used as a label. In the eA5 system the students are presented with a list of labels in the form of a drop down selection box. The labels in this list are all the different noun phrases present in the problem specification. The students are required to choose the labels from this list. The system eA5 classifies all the labels in the student diagram as either directly or indirectly referencing a label in the model solution. A directly referencing label is one where the students have picked up the label directly from the problem specification. However sometimes students pick up two or more labels from the problem specification and merge them into a new label. For example in the problem specification for the case study (see appendix Section 6.5), the student may select "pre-pay card" and "concessions card" and merge them into a single label "non-travel card". Such labels in the student diagram are different from the labels in the model solution and known to be indirectly referencing the labels in the model solution. To deal with the indirectly referenced labels in the student diagram, the eA5 system provides buttons to split and merge labels. The student selects the labels from the list and use these buttons to create new labels.

The systems eA3 and eA4 uses the edit distance algorithm (Wagn 74; Nava 01), manually prepared list of synonyms and abbreviations to deal with the label matching problem. The syntax algorithm used by the systems eA3 and eA4 returns a score based on the edit distance (Wagn 74; Nava 01) between the two labels. If the edit distance is less than a certain threshold value the labels are considered to be similar (Tsel 05). In the eA4 system, all the matched labels having an edit distance less than a threshold are presented to the lecturer in the form of a navigable tree which can then be reviewed and updated by the lecturer (Jone 05).

So the existing e-assessment systems acknowledge the problem of label matching but the extent of the problem has not been measured before. Also only a few techniques have been explored to deal with this problem. The use of algorithms to finding syntactic similarity between labels is limited to edit distance algorithm (Wagn 74; Nava 01) which is used by the system eA3 and eA4. Techniques to find the semantic similarity are limited to manually prepared lists of synonyms and abbreviations used by the system eA3. Also there is no generic framework for matching the labels for e-assessment. So there is a need for the following four things:

- Formally assess the extent of the label matching problem

  The extent of the problem should be empirically measured using real coursework. Towards the end of writing of this Thesis a similar work by Thomas was found at (Thom 09) which explores the label matching problem for the e-assessment of diagrams. The research finds that the label matching is an existing problem and the various values of threshold for similarity index affect the performance of the e-assessment tool.

- Explore syntax algorithms

  Apart from the edit distance algorithm (Nava 01) used by the systems eA3 and eA4, other syntax algorithms for example q-gram (Ukko 92) and simon algorithm (Whit nd) should be explored. The syntax algorithm returns a number between 0 and 1 as the similarity score. If the score is more than a certain threshold the labels are considered to be similar. The threshold value can be important for the matching process so there is also a need for exploring the optimal value of the threshold.

- Explore semantic techniques

  The semantic techniques for example WordNet (Mill 95a), Wu & Palmer (Wu 94), Leacock and Chodorow (Leac 98) etc. should be explored.

- Extensible and configurable framework for label matching

  A generic framework should be developed for matching the labels for e-assessment. This framework should be extensible and configurable.

Chapter 3

Diagram Label Matching Framework

3.1 Introduction

Chapter 2 discussed existing systems to automatically mark the diagrams and the techniques used to match the diagram labels. In this chapter an extensible and configurable framework is proposed to match the labels in a diagram. This chapter starts by discussing the research methodology used in this Thesis and then discusses the non-functional and functional requirements of the proposed framework. It then explains the various stages of the framework linking them to the requirements.

For the purpose of the framework proposed in this Thesis, a word is defined as any continuous sequence of alphanumeric characters while a label is defined as a sequence of words separated by one or more spaces.

Word=(A-Z | a-z | 0-9 | special character)*

Label = (Word (single space)*)*

Towards the end of writing this Thesis I came across a rarely cited technical report by Hart (Hart 94) which proposes an improved algorithm to identify the spelling and word order errors in student coursework. Similar to the framework proposed in this Thesis, it is configurable. For example it can be be configured for extra words using the parameter extraWordOk which if True will judge a student label as correct even if extra words are present. Similarly it can be configured for sensitivity using parameter capFlag which can have the values "exact_case, authors_caps, ignore_case"). This work differs from the framework proposed in this Thesis because it does not evaluate the various syntax algorithms and does not use semantic analysis.

3.2 Research Methodology

This Thesis uses empirical and design research methodologies. First, the empirical research is used to measure the extent of the diagram label matching problem. The result of this experiment shows that the diagram label matching is a substantial problem and cannot be easily avoided for the e-assessment of diagrams. These results will be presented in the next chapter. Having established the problem a system is required to solve it. Since the domain of e-assessment of diagram-based coursework lacks maturity and there are no open source systems for the same a new system needs to be designed and evaluated. The design research methodology (Vais 04; Take 90) suits this need as it involves the development and evaluation of a system. So the design research methodology is used to design a framework for diagram label matching. This framework is then evaluated on coursework from a UK HEI. The results of this evaluation will be presented in the next chapter.

3.3 Requirements of Framework

The following are the functional and non-functional requirements of the framework.

3.3.1 Match labels

The framework should compare the labels in the student diagram with the labels in the model solution and produce a set of matching pairs of labels. This involves removing ambiguities in the student labels and calculating the similarity score with the labels in the model solution. By manually categorising the students coursework used in the experiments in this Thesis, the following kinds of ambiguities were found. The results of this manual categorisation has been presented in the Section 4.4 Chapter 4.

- Differing case, leading, trailing and embedded spaces.
- Special characters, so a label in the student diagram may contain special characters for example underscore, currency symbols etc.
- Misspellings
- Concatenation of words

  Sometimes students concatenate multiple words in a label for example in the label "UpdateTotal". To match the concatenated label to the correct label, it needs to be separated into words.
- Differing number of words and order.

  A label in the student diagram may contain more or less words than the corresponding correct label in the model solution. Also, the words in the student label may be arranged in a different order than the corresponding correct label in the model solution.
- Synonyms and abbreviations.

  The student label may contain a synonym or abbreviated form of a label in the model solution. For example the student may use "update amount" instead of "update total" or "expdate" instead of "expiry date".
- Different level of decomposition

  Sometimes the students provide a more detailed answer than the model solution and so there is a subsumption relationship between one label in the model solution and a set of many labels in the student answer. The human expert will give marks for those lower level labels in the student answer which when composed add up to form a label in the model solution. For example the label "check balance" in the student diagram is a detailed level of the label "update total" in the model solution as balance is checked while updating the total [1].

[1] The categorisation can be subjective and may differ from person to person. The categorisation process

The work in this Thesis handles all the above mentioned label ambiguities except the last one namely “Different level of decomposition”. Some of the ways to handle this imprecision are discussed in Chapter 5 of this Thesis.

3.3.2 Configurable and extensible

The aim of this framework is not to propose any specific algorithm or a set of parameters, rather it is to propose a sequence of steps for matching the labels in a student diagram to those in the model solution. According to Johnson (John 88), a framework plays the role of the main program in coordinating and sequencing application activity and consists of an abstract class for each major component. So the framework in this Thesis should only define a main program for sequencing the steps and a set of interfaces for various components. The user should then use the framework by providing the concrete implementations of these interfaces. To enable the framework to be used ‘out of the box’ by new users the framework should define default implementations of the interfaces and provide default values for the set of parameters. In order to evaluate the framework, it was implemented using a set of specific algorithms and parameters. These specific algorithms and parameters are embedded as the default behaviour of the framework. However, this Thesis does not compare the various algorithms and parameters available for each step of the framework except for the syntax stage.

So the non-functional requirements of the framework are that it should be configurable, extensible, generic by design and not confined to an algorithm, an implementation of an algorithm or a set of parameters. It should also be easy to use. The users should be able to configure the framework according to their needs or availability of algorithms. There can be multiple algorithms for a task. For example the spellings can be corrected using different spell checking algorithms for example Hunspell (Neme 10) or ASpell (Atki 04). Different algorithms may work best in different situations and users should be able to choose the particular algorithms to be used. Also, on occasions users may have access to better commercial

carried out in this Thesis was reviewed by two members of academic staff at the Brunel University. The results of this categorisation has been explained in Chapter 4 and non-open source algorithms which they should be able to use within the framework. So the framework should be independent of any particular algorithm or implementation detail of algorithms and the user should be able to configure the framework to select algorithms of their choice. The user should also be able to configure the framework for various parameters like threshold, list of special characters, abbreviations and stopwords. The framework should be available as a jar file with Java documentations for it's application programming interface (API) so that it can be integrated with other applications.

3.4 Framework

The framework proposed in this Thesis is driven by a simple high level design rationale which is to first disambiguate the labels and then to calculate the syntactic and semantic similarity. Figure 3.1 shows this proposed framework. Table 3.1 lists the five stages of the framework and Table 3.2 summarizes the input and output of these five stages. The *first* stage disambiguates the labels to produce a set of cleaned labels that are used for the subsequent stages. The *second* stage runs the syntax algorithms on these cleaned labels to produce a matrix for the syntactic similarity index. The *third* stage uses WordNet to produce a list of synonyms in the form of a synonym XML file. The *fourth* stage uses this synonym XML file and re-executes the *second* syntax stage to produce a matrix of the combined similarity matrix. The final *fifth* stage analyses this combined similarity matrix and marks the labels in the student's diagrams as correct if a match can be found in the model diagram and incorrect if a match can not be found in the model diagram. Only the first three stages of this framework namely pre-processing, syntactic matching and semantic matching have been evaluated in this Thesis, the results of which are presented in the next chapter. The remaining two stages namely combined similarity and analysis have been included in the framework so as to make it complete.

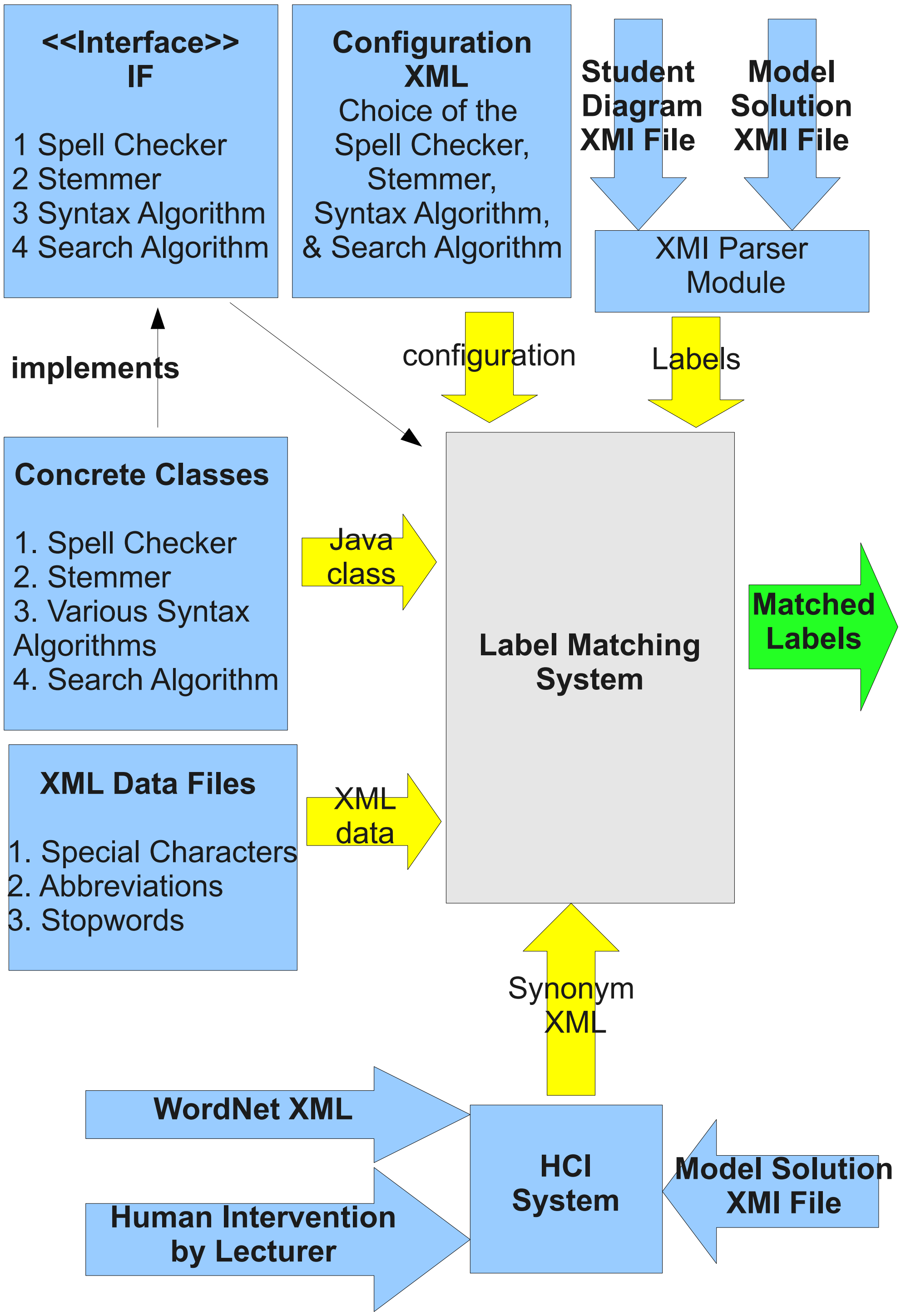


Figure 3.1: DiagramLabelMatchingFramework

Reference	**Stage Name**	**Main Purpose**
S1	Pre-processing (6 Sub Stages)	Disambiguate the labels
S2	Syntactic Matching	Calculate Syntax Similarity
S3	Semantic Matching	Produce synonym XML file
S4	Combined Similarity	Return stage S2 using synonym XML file
S5	Analysis	Best Match Selection

Table 3.1: Framework Main Stages

Stage Name	**Input**	**Output**
Pre-processing (6 Sub Stages)	Labels	Processed labels
Syntactic Matching	Processed labels	Syntactic similarity matrix
Semantic Matching	Processed labels	Synonym XML file
Combined Similarity	synonym XML file	Similarity matrix
Analysis	similarity matrix	List of matching pair of labels

Table 3.2: Framework Stage Input Output

Reference	**Sub Stage Name**
S1.1	Lowercase And trimming
S1.2	Special Character Replacement
S1.3	Abbreviation Expansion
S1.4	Auto Correct
S1.5	Removing Stopwords
S1.6	Stemming

Table 3.3: Sub-stages of Pre Processing Stage

3.4.1 Pre processing stage

The pre-processing stage comprises of the measures carried out before any actual similarity matching is done and is analogous to the data cleaning which is a standard and a well researched approach in data analysis. It takes the raw labels as input and produces a set of processed labels which are used in the subsequent stages. The purpose of this stage is to remove the superficial imprecision in the labels so that they can be better matched against the correct labels in the model solution. This stage removes the imprecision caused by using different case, space, special character, abbreviation, spelling mistake, stopword and different derived word from the same root word. Table 3.3 lists all the sub-stages of the pre-processing stage. Following are the details of each sub stage.

- Lowercase and trimming

  This stage involves converting all the labels to lowercase and removing the leading and trailing spaces.

- Special Character Replacement

  This stage involves replacing or removing the special characters, for example ′ (single quote), “_” (underscore) and “&” (ampersand). Some of the special characters may be used to separate the words in a label for example “update_total”, so they should be replaced by a single space “ ” while others for example ′ (single quote) should be removed. Also some of the special characters may have specific meaning in a particular domain and removing or replacing them may change the meaning of the label. To overcome this problem a list of domain specific special characters should be used. The user can modify the list of special characters using the configuration XML file. For the purpose of evaluating the framework the results of which are explained in the next chapter the list of special characters that was used is presented in the Appendix Section 6.2.1. The framework allows flexibility to modify this list.

- Abbreviation and Acronym Expansion

  Sometimes students use abbreviations and acronyms in the labels, for example “msg” for “message” and “paygcard” for “Pay As You Go Card”. Expanding the abbreviation is necessary to match the labels, but it is a difficult task. It can not be solved by using a dictionary as only some of the dictionaries register some of the commonly used abbreviations as informal words (Tera 04). For example the abbreviation “calc” is not present in the WordNet(Mill 95b) but is registered as an informal word for “calculation” in the Dictionary.com (Dict 10). Also the abbreviations can be ambiguous. For example the abbreviation “exp” may mean “exponential” or it may also mean “expiry”. Expansion becomes more difficult if the student has appended the abbreviation with some other word for example in “check expdate” because the combined words need to be separated before doing expansion. Terada (Tera 04) has done work on automatic expansion of abbreviations by using context and character information but it is still an ongoing research problem.

  This stage involves replacing the abbreviations and acronyms with their expanded forms using a list of abbreviations for example the list available at (LLC 10b; Univ 96). Also the free REST style (Fiel 02) Application Programming Interface (API) has been provided at (LLC 10a) to retrieve the expanded form of abbreviations in the form of XML.

  For the purpose of evaluating the framework, the results of which are explained in the next chapter the list of abbreviations available at (LLC 10b) was used. The framework allows flexibility to use a different list of abbreviations.

- Auto Correct

  This stage involves replacing the misspelled words in a label with their correct spelling using a spell checker. There are different spell checkers available for example Hunspell (Neme 10) which is used by Open Office (Corp 10a), Microsoft office spell checker, GNU Aspell (Atki 04) etc. For the purpose of evaluating the framework the results of which are explained in the next chapter the Hunspell spell checker (Neme 10) was used as its implementation was easily available in Java (Fran 09). The framework allows flexibility to use any spell checker. Also the spell checker often suggests multiple words having different meanings for a supposedly misspelled word and it needs to be determined which one among the multiple suggested words should be selected to replace the supposedly misspelled word. Sometimes, but not always, the first word suggested by the spell checker is the correct one, meaning a human expert will choose it to replace the misspelled word. For example for the misspelled word "ceck", the spell checker suggests words "check", "neck", "deck" and "peck" in order. In this case the first word "check" is the correct one. However in another example for the misspelled word "vaildiate", the spell checker suggests words "vacillate", "radiately", "validate" and "repudiate" in order. In this case the third word "validate" suggested by the hunspell is the correct one.

  For the experiments in this Thesis the first word suggested by the spell checker was used to replace the supposedly misspelled word. The evaluation results as explained in the next chapter show that selecting the first suggested word gives reasonable performance in terms of accuracy. Also the auto correct stage sometimes introduces special characters. For example it flags "offpeak" as a spelling error and auto corrects it to "off-peak". Similarly it flags "todays" as a spelling error and auto corrects it to "today's". It does not have any negative effect but introduces special characters like minus sign and single quotes. So the stage S1.2 which involves replacing special characters needs to be rerun after the auto correct stage to remove any special characters introduced by it.

- Removing Stopwords

  This stage involves removing the stopwords which are very common words like “is”, “an”, “the”, “to” that are not relevant for matching. A list of English language stop words can be found in google (Doyl nd). For the purpose of evaluating the framework, the results of which are explained in the next chapter the list of stopwords that was used is presented in the Appendix Section 6.2.2. The framework allows flexibility to modify this list.

- Stemming

  This stage deals with the different forms of the same root word. It involves replacing the words in a label with their root words (Wiki 10e) using a stemmer. For example the word “warning” is replaced by its root word “warn”. There are different stemming algorithms available for example Lovins(Lovi 68), Paice(Paic 90), Porter(Port 06), S-Removal(Harm 91) etc. Smirnov(Smir 08) presents an overview of various stemming algorithms. Frakes(Frak 03) has done an evaluation of these four stemming algorithms. This evaluation shows that Paice is strongest followed by Lovins, Porter and S-Removal in order. The strength of a stemmer is defined as how much it changes words. This evaluation also shows that the amount of over stemming is zero in S-removal followed in increasing order by Porter, Lovins and Paice. So there is tradeoff between over and under correction. In the proposed framework, over stemming is a significant drawback because the stemming is a part of the first stage and its output is fed into the subsequent stages, so the over stemming will have a multiple negative effect on the subsequent stages. Also stemming should be applied after running all the other stages of pre-processing stage because applying it early on can negatively effect the performance of other stages. For the purpose of evaluating the framework the results of which are explained in the next chapter the Paice stemming algorithm (Paic 90) was used as its implementation source code was easily available in Java (Neil 00). But the framework allows flexibility to use any stemming algorithm.

3.4.2 Syntax stage

The syntax stage proposes a new hybrid syntax algorithm known as "Combined Hybrid Syntactic Algorithm" and uses it to calculate the syntactic similarity between two labels. This hybrid algorithm is explained in the next subsection and it uses four existing syntax algorithms namely edit distance, Q-gram, simon and soundex. The framework allows flexibility to configure the hybrid algorithm for the choice of existing syntax algorithms used. This can be done by making changes in the configuration XML file and the sample code to do this has been presented in the Appendix Section 6.8. The results, as explained in the next chapter show that it works better than the existing syntax algorithms on the corpus of coursework used in this Thesis.

The approach behind the combined syntax algorithm is that since previous research shows that none of the existing syntax algorithm dominates and is a clear best (Chri 06; Cohe 03), use the results from various syntax algorithms to calculate the syntactic similarity. This is analogous to searching for the best solution from a search space which consists of various syntax similarity scores returned by the different syntax algorithms. The search algorithm used in the combined syntax algorithm in this Thesis is simply to choose the maximum value. The maximum value is selected in order to maximise the chance of finding the match for a label. However the framework is configurable to select a different search algorithm for example average, median or mode of all the values returned by the individual syntax algorithms.

Combined Hybrid Syntactic Algorithm

This algorithm combines the existing word-to-word syntax matching algorithms into a label-to-label syntax matching algorithm. It takes as input two labels, the first being the label in the model solution and the second being the label in the student diagram. It then runs the various syntax algorithms inputting one word from the first label and one word from the second label. This is done for each pair of words in the cartesian product of word pairs between words in first and second labels and a similarity index for each pair of words is obtained. After that it selects the matching pair of words based on the threshold value and the maximum value of similarity index. While selecting the matching pairs, it removes the duplicates because one word in the first label can have only one corresponding matching word in the second label. Once the matching pair of words has been selected, it then calculates the overall similarity between the labels by taking the sum of the similarity scores of all the matched pair of words and dividing it by the total number of words present in the first label, which is the label in the model solution. The extra words present in the student diagram are handled by checking them against the negation words list which is a list of words like "no, don't" that negate the meaning of label. If any of the extra words present in the student label are not a negation word then they do not impact the similarity score of the labels, otherwise the similarity score of the labels is set to zero. Also, since this algorithm operates on a word by word basis, differing orders of the words in the labels do not impact the similarity score of the labels. Hence this algorithm is independent of the word order and insertion of extra words. The following steps explain the algorithm.

1. Input two labels L_i and L_j. The first label L_i denotes the label in the model solution while the second label L_j denotes the label in the student diagram. Also input the negation words list which is a list of words like "no, don't" that negate the meaning of label.

2. Extract words[2] from each label, for example L_i has words $w_{i1}, w_{i2}..., w_{in_i}$ and L_j has words $w_{j1}, w_{j2}..., w_{jn_j}$ where the number of words in the label L_i is n_i and number of words in the label L_j is n_j.

3. Prepare the cartesian product pairs between the words in the first and the second labels, so we have
$(w_{i1}, w_{j1}), (w_{i1}, w_{j2}), ..., (w_{in_i}, w_{jn_j})$

4. For each pair of words, calculate the Syntactic Similarity Index (SynSI) using the following formula:

[2] Recall from the definition of word and label in the beginning of this chapter that a label consists of a sequence of words separated by one or more spaces

$SynSI(w_i, w_j)_{csa} = \max_{p=1}^{5} SynSI(w_i, w_j)_{algo_p}$

where p is defined as:

(a) Exact Match algorithm [3] ($p = 1$)

(b) Levenshtein Distance algorithm ($p = 2$)

(c) Q gram algorithm ($p = 3$)

(d) Simon algorithm ($p = 4$)

(e) Soundex algorithm ($p = 5$)

The Combined Hybrid Syntactic Algorithm is denoted by *csa*.

5. For each word w_k in the first label L_i, select the word w_l in the second label L_j such that the pair has maximum possible value for $SynSI(w_k, w_l)_{csa}$. Remove the duplicates and recalculate because one word in the first label can have only one corresponding matching word in the second label. If one word in first label has the same value of similarity index with two words in the other label, then select one randomly.

6. Read the word similarity threshold value represented by $T(SynSI)$ from the configuration XML file. The value of this threshold used for experiments in this Thesis is 0.6.

7. For the word pairs for which $SynSI(w_i, w_j)_{csa} < T(SynSI)$, set the $SynSI(w_i, w_j)_{csa}$ to 0. This is because if the value is less than the threshold, the words are considered to be non matching. The remaining pairs are considered to be matched and these will be used to calculate the similarity index between the labels.

8. Each word in the second label L_j that has not been matched to any word in the first label L_i is termed an extra word. Check such extra words in the second label L_j against negation word list[4]. If any of the extra word is present in the negation words list, then

[3] The plain match algorithm is kept for completeness. Also it can be used to optimise the Combined Hybrid Syntactic algorithm by not executing any other individual syntax algorithm if a match is found using the plain match algorithm.

[4] This step has not been implemented in the software prototype used for evaluating the framework.

set the similarity index between labels L_i and L_j denoted by $SynSI(L_i, L_j)_{csa}$ to zero. Otherwise, calculate the similarity index between labels L_i and L_j by taking the mean of the similarity index vales of the matched pair of words using the following formula: $SynSI(L_i, L_j)_{csa} = \frac{\sum_{i=0}^{n_i} SynSI(w_i, w_j)}{n_i}$ where n_i is the number of words in the first label L_i. Note that if no match is found for a particular word in the first label or if the similarity index value is less than the threshold, then its similarity index value is set to zero and hence the overall similarity index of the label is reduced.

9. Return the SynSI value calculated above as the syntactic similarity index between the labels L_i and L_j.

10. Two labels are considered to be matched if the syntactic similarity index between them is more than the value of the label similarity threshold mentioned in the configuration XML file. Experiments were carried out using the label similarity threshold value of 0.5 through 1. The results of these experiments are explained in the next chapter.

This algorithm is not symmetric in the sense that the similarity score between label1 and label2 may be different from the similarity score between label2 and label1. The reason for this is that the first label is considered to be the label from model solution and the value of the number of words in it is used to calculate the final similarity score.

Example of the Combined Hybrid Syntactic Algorithm

Considering the label in the model solution “invalid warn” and the label in the student diagram “not valid”, the value syntactic similarity index between them calculated using the combined hybrid syntactic algorithm is 0.40. Table 3.4 shows the various intermediate values while calculating the final score. Before applying the threshold the matching pair of words are (not, invalid, 0.14) and (valid, invalid, 0.80). After applying the threshold value of 0.6 the first pair of words (not, invalid, 0.14) is discarded as its similarity score is less than the threshold. The only remaining matching pair of words is (valid, invalid, 0.80) and it will be used to calculate the similarity index between the labels. As the number of words in the correct label “invalid warn” is two so the similarity score between the matching pair is

Word1	Word2	Exact	Simon	Levenshtein	Soundex	Q-gram	Max	Match
not	invalid	0.0	0.0	0.14	0.0	0.0	0.14	Selected but discarded as 0.14 less is than threshold 0.6
not	warn	0.0	0.0	0.0	0.0	0.0	0.0	Not selected as it is not maximum
valid	invalid	0.0	0.80	0.71	0.0	0.63	0.80	Yes
valid	warn	0.0	0.0	0.20	0.0	0.0	0.20	Not selected as it is not maximum

Table 3.4: Combined Syntax Algorithm Example

divided by two. Accordingly, the overall similarity index between the labels “not valid” and “invalid warn” is equal to 0.40 (0.80 divided by the number of words in the first label “not valid”, =0.80/2).

Design rationale

The syntax algorithm used by the existing e-assessment tools retrieved from the systematic search described in Chapter 2 uses the edit distance algorithm to find the syntactic similarity between labels. Three widely known existing syntax algorithms are edit distance (Nava 01), Q-gram(Ukko 92) and Simon(Whit nd) algorithm. Previous research has shown that there is no single best syntax algorithm available (Chri 06; Cohe 03). Each of these existing syntax algorithms have their own strengths and weaknesses. According to Christen (Chri 06), “Experimental results on different real data sets have shown that there is no single best technique available. The characteristics of the name data to be matched, as well as computational requirements, have to be considered when selecting a name matching technique” . Below are the characteristics of a syntax matching algorithm for the e-assessment domain. These characteristics are identified by analysing the corpus of coursework used in this thesis.

- Reflection of word based lexical similarity

  The similarity score of two labels should be monotonic to the number of common words between them. This characteristic gives rise to the following two requirements.

  - Word Order Independent

    The algorithm should be word order independent. This applies to the cases where the student label contains no extra words and has all the words contained in the correct label but at different positions. For example:
    Student Label="time check", Correct Label="check time"
    Student Label="card reader", Correct Label="read card"
    Student Label="cardreader", Correct Label="read card"

  - Insertion of extra non-stopwords

    The student should not be penalised for extra words in the label unless the extra word negates the meaning. So for example the following labels should be judged as matched.
    Student Label="invalid audible warning", Correct Label="invalid warning"
    Student Label="invalid beep warning", Correct Label="invalid warning".

    Sometimes the extra word inserted negates the meaning of the label for example "check time" and "do not check time". Although we did not find any such case in the corpus of coursework used in this Thesis, such a scenario is possible. So a list of negation words for example "no, don't" should be used while calculating the similarity. The insertion of any extra word not present in this list should have no bearing on the similarity index of labels but the insertion of a extra word from the negation word list should impact the similarity score by setting it to zero.

- Similar Sound

  Sometimes a student may spell the word according to it's sound for example "updat" instead of "update" as both have similar sounds. The algorithm should be able to deal with this.

Reference	**Syntax Algorithm**	**Source**
A1	Exact Match	None
A2	Soundex	(Wiki 10d; Russ 18)
A3	Levenshtein Distance	(Nava 01)
A4	Q-gram	(Ukko 92)
A5	Simon Algorithm	(Whit nd)
A6	AlgoMax	(Jaya 09a)
A7	Combined Syntactic Algorithm	(Jaya 09a)

Table 3.5: Syntax Matching Algorithm

- Time Complexity and Memory

  Generally the approximate matching algorithms in the NLP domain are led by the objective of reducing the time complexity and the memory requirement. But in the e-assessment domain the execution time and memory are not major constraints because the number of labels in the model solution and the student diagram is generally limited. As explained in the next chapter the corpus of coursework of 160 students used in this Thesis has a total of 773 different labels, averaging 5 unique labels per student diagram. So multiple syntax algorithms can be used simultaneously to calculate the syntactic similarity for e-assessment of diagrams because the execution time and memory are not major constraints.

Existing syntax algorithms

Following are the existing syntax matching algorithms.

- Exact Match Algorithm

  This simply compares the two labels and produces a value 1 if an exact match is found otherwise produces a value 0.

- Levenshtein Distance Algorithm

  The Levenshtein distance between two strings is calculated by counting the the minimum number of insertions, deletions, or substitutions of a single character that is needed to transform one string into the other (Wagn 74; Nava 01). The following algorithm is used by (Chap 06) for calculating similarity using Levenshtein distance.

  - Set cost of one insertion or deletion or substitution of a single character = 1
  - Calculate minimum number of operations required to transform one string into the another string.
  - $Similarity = 1 - \frac{costofminimumtransformationoperations}{lengthoflongerstring}$

  Following are two examples of calculating similarity using Levenshtein distance algorithm.

  String1=valid
  String2= invalid
  Minimum operations required= Two insertions (insert ‘i’ and ‘n’ in the first string)
  Cost of minimum transformation operations= 2
  Length of longer string=7
  Similarity=1 - (2/7)=1-0.29=0.71

  String1=valid
  String2= warn
  Minimum operations required= Three insertions and One deletion (substitute ‘v’ with ‘w’ in the first string, substitute ‘l’ with ‘r’ in the first string, substitute ‘i’ with ‘n’ in the first string, delete ‘d’ from the first string)
  Cost of minimum transformation operations= 4
  Length of longer string=5
  Similarity=1 - (4/5)=1-0.8=0.2

- Q-gram Algorithm

  This algorithm calculates similarity between two strings based upon the number of common Q-grams between them (Ukko 92). The implementation of this algorithm available at (Chap 06) is used in this thesis for evaluating the framework. This implementation uses trigrams (q=3). It also appends the characters "##" at the beginning and end of each string so that the starting and ending characters of each string also count equally towards the similarity score as the middle characters.

- Simon Algorithm

  This algorithm uses the number of common adjacent character pairs contained between two strings as a measure of their similarity (Whit nd). The formula for calculating the similarity score used by this algorithm is as follows. $similarityindex = \frac{2*numCommonPairs}{(numPairsStr1+numPairsStr2)}$ where numPairsStr1 is the number of adjacent character pairs contained in the first string, numPairsStr2 is the number of adjacent character pairs contained in the second string and numCommonPairs is the number of common adjacent character pairs between first and second strings. This algorithm[5] is the same as the Dice coefficient (Hill 06).

- Soundex Algorithm

  This uses the similarity in sound produced by two labels as a measure of similarity between them and is based on the Soundex Algorithm (Wiki 10d; Russ 18). So for example "Update" and "Updat" will have a Soundex similarity score of 1 whilst "go" and "come" will have a Soundex similarity score of 0. This algorithm has some variations like RefinedSoundex, Metaphone and DoubleMetaphone (Foun 09).

The aim of experiments carried out using this stage of the framework is to determine which syntax algorithm is best suited for e-assessment of diagrams. The results of these experiments are explained in the next chapter.

[5] According to an informal email conversation with the author of this algorithm Simon White, he had reinvented it without knowing about the Dice coefficient at that time.

Label1	Label2	A1	A2	A3	A4	A5
invalid audible warning	invalid warning	0	1	0.75	0.88	0.9
invalid beep warning	invalid warning	0	1	0.71	0.79	0.86
card reader	read card	0	0	0.33	0.45	1
time check	check time	0	0	0.11	0.36	1
read card	card reader reads card information	0	0	0.35	0.36	0.52

Table 3.6: Syntax Matching Example

Alternative Option: Use the various existing syntax algorithms individually

This option means individually using the various existing syntax algorithms explained in the previous section. The existing e-assessment tools use this approach by using just the edit distance algorithm. Table 3.6 shows the similarity score calculated using the syntax algorithms for five sets of labels present in the corpus of student coursework used for experiments in this Thesis. The details of the corpus of coursework are explained in the next chapter and the simmetrics library (Chap 06) has been used to execute the algorithms [6]. All the five sets of labels present in the Table 3.6 have been judged as correct by the human marker and hence the machine should judge them the same as well. But as can be seen from this table, none of the five syntax algorithms performs well consistently and fulfils all the requirements. To overcome this limitation the combined hybrid syntax algorithm explained in section 3.4.2 was proposed.

Design Rationale: Combined Hybrid Syntactic Algorithm

As demonstrated in the previous section using various examples, every syntax algorithm works best in some situations but none works best in all the situations. In other words, none of the syntax algorithms dominates and each has its strengths and weaknesses. So it is

[6] The soundex algorithm implementation in the simmetrics library (Chap 06) takes into account just the first word of the labels. So the labels "invalid audible warning" and "invalid warning" have soundex similarity of 1 as both start with the common word "invalid".

proposed to use multiple syntax algorithms in a hybrid way because using just one of them means missing out the strength of others. This mean that two labels should be considered as similar if any one of the syntax algorithms identifies them as similar. One way to use multiple syntax algorithms is to combine them using the AlgoMax hybrid algorithm. This AlgoMax hybrid algorithm takes, as input, two labels and returns a syntactic similarity index between 0 and 1 with 1 meaning a perfect match. It runs the various syntax algorithms for example edit distance, Q-gram etc., thus obtaining different similarity scores. It then simply returns the maximum value. Although the AlgoMax Hybrid Algorithm does combine the various syntax algorithms it is still not independent of word order and insertion of extra words (which are the requirements identified in the previous section), because it applies the various syntax algorithm on the label as a whole and not on individual words. So it is modified by applying the various syntax algorithms at the word level rather than at the label level. This means that the input to the syntax algorithm should be words rather than whole labels. The syntactic similarity between words should then be combined to calculate the syntactic similarity between the labels. This new modified algorithm explained in section 3.4.2 is known as the "Combined Hybrid Syntactic Algorithm". Similar to the AlgoMax algorithm, it uses different syntax algorithms and returns a syntactic similarity index between 0 and 1. But it differs from the AlgoMax algorithm in the way it runs these syntax algorithms and processes the results. While AlgoMax runs the various syntax algorithms inputting the two complete labels as they are, the "Combined Hybrid Syntactic Algorithm" runs the various syntax algorithms inputting one word from the first label and one word from the second label. Please see section 3.4.2 for details of this algorithm.

There are two design decisions to be made for this stage. The first is to decide which of the individual syntax algorithms to use and the second is to decide how to combine the similarity scores from the various syntax algorithms. The framework is plug and play based, so the users can choose this at runtime by providing the concrete class for the syntax algorithms and adding entries for them in the configuration XML file. To carry out experiments, for the first decision we have chosen the five existing syntax algorithms explained in the previous section. For the second decision, we have decided to select the maximum value out of the similarity indexes by various syntax algorithms.

Summary For Syntax Stage

Overall the syntax stage takes as input the processed labels from the pre-processing stage, runs the various syntax matching algorithms and outputs a similarity matrix for the labels in the student and model solution diagrams. However, the student may often use a label that is syntactically different but semantically the same as the correct label. For example a student may use “update amount” in place of “update total” or “invalid beep” in place of “invalid warning”. The syntax matching stage is limited to the syntactically similar labels and would fail to deal with such labels that are syntactically different but semantically same. The next stage deals with such labels.

3.4.3 Semantic stage

As explained in the previous section, the syntax matching stage would fail to match labels that are syntactically different but semantically the same. This stage complements the previous stage by enhancing the capacity of the framework to match such labels. This is done by producing a list of synonyms for each word in the model solution and adding them to a synonym XML file. The synonyms are produced from the WordNet (Mill 95a) using a human computer interface (HCI) system used by the lecturer. WordNet (Mill 95a) lists the different senses in which a word can be cognitively used and the corresponding synonyms for each of these senses. The HCI system presents the various senses of a word and allows the lecturer to manually select the senses in which the word has been used in the model solution. The system then picks up the synonyms and hypernyms[7] for the selected senses from the WordNet and generates the synonym XML file. This synonym XML file is then fed into the

[7] The semantic analysis carried out in this thesis was done using human expert. The human expert treated the hypernyms as synonyms for example “chec hours” and “check time”. So for the purpose of this Thesis, hypernyms are treated as synonyms. However this may be an issue in other domains. For example if the model solution has both a hypernyn and the corresponding synonym as separate labels. This requires further work and is an area for future research.

next stage namely combined similarity stage.

The next chapter explains the evaluation of the semantic stage. The HCI system used to carry out this evaluation was openoffice spreadsheet (Corp 10a). The spreadsheet was used because it was easily available and sufficient for evaluation purposes considering that the number of words in the model solution was small. However more efficient HCI like navigable tree based expandable check box nodes would help the lecturer to choose the synonyms more quickly. Such a tree based HCI has been used by Assess By Computer system to allow the lecturer to review and update the matched labels (Jone 05; Tsel 05). Below is an example of the synonym XML file. The document type definition (DTD) schema of the synonym.xml file is presented in the appendix.

```
<synonymset name="example synonym xml">
<word description="update total" value="total">
<!--Sense #1-->
<synset description="" wordnetdatabseid="04353803" selectedbylecturer="yes">
<sense value="the whole amount">
</sense>
<synonymn value="sum, total, totality, aggregate">
</synonymn>
<partofspeech value="noun">
</partofspeech>
<examplesentence value="">
</examplesentence>
</synset>
<!--Sense #2-->
<synset description="" wordnetdatabseid="05861067" selectedbylecturer="yes">
<sense value="a quantity obtained by the addition of a group of numbers">
</sense>
<synonymn value="sum, amount, total">
```

```
</synonymn>
<partofspeech value="noun">
</partofspeech>
<examplesentence value="">
</examplesentence>
</synset>
&lt;!--Sense #3-->
<synset description="" wordnetdatabseid="00515380" selectedbylecturer="no">
<sense value="constituting the full quantity or extent; complete">
</sense>
<synonymn value="entire, full, total">
</synonymn>
<partofspeech value="adjective">
</partofspeech>
<examplesentence value="an entire town devastated by an earthquake">
</examplesentence>
</synset>
</word>
</synonymset>
```

This example synonym xml shown above finds the synonyms of the word “total” as used in the model solution label “update total”. It uses WordNet and shows three different senses of the word “total” and the synonyms corresponding to each of these three senses. For each sense it also has an XML attribute “selectedbylecturer=yes/no” which shows whether the lecturer agrees if the word “total” was used in this sense in the model solution label “update total”. As can be seen in this synonym xml, the value of parameter selectedbylecturer is equal to “yes” for the first two senses and “no” for the third sense, which means that according to the lecturer the valid synonyms of the word “total” as used in the label “update total” are those corresponding to the first two senses. Hence the valid synonyms of word “total” are “amount, sum, total, totality, aggregate”. After this synonym xml is fed into the framework, the labels “update amount” and “update total” would be judged as matched since the word “amount” has been listed as a synonym for the word “total”. On the other hand the labels “update full” and “update total” would not be judged as matched since the word “full” has not been listed as a synonym for the word “total”.

Design Rationale

As seen in the figure 3.2 there are two design decisions to be made during this stage. In step one the source for the semantic similarity is to be decided. The two available options for this are a linear dictionary like (Dict 10) and WordNet (Mill 95a). The linear dictionary is easy to use but does not take into account various semantic relations. WordNet is a cognitive science based collection of English words and the various senses in which they can be used (Mill 95a; Wiki 10h). Its latest 2.1 version has a total of 155327 words. A sense can be defined as the way in which a word can be used. For example the word “beep” can be used in three different senses. First as a noun meaning “a short high tone produced as a signal or warning”, second as a verb meaning “make a loud noise” and third as a verb again meaning “call, summon, or alert with a beeper”. So in the WordNet the word “beep” has three senses. For each of these senses WordNet also defines a set of synonymous words that can used interchangeably to represent that particular sense. So the first noun sense of the word “beep” has synonym “bleep”, while it’s second verb sense has synonyms “honk, blare, claxon, toot”. In other words it can be said that, to represent the noun sense “a short high tone produced as a signal or warning”, the word “beep” or “bleep” can be used. Similarly to represent the verb sense “make a loud noise” the words “honk” or “blare” or “beep” or “claxon” or “toot” can be used. Also the different senses can be related to each other. For example the sense “Sound (make a certain noise or sound)” is a hyerpnym or more generic form of “Beep (make a loud noise)”, “building” is a holonym of “window” and “full (complete in extent or degree and in every particular)” is similar to another sense “complete (having every necessary or normal part or component or step)”. So each sense has pointers relating it to other senses. This combination of a sense, the synonymous words that can be used interchangeably to represent that sense and the pointers relating that sense to other senses

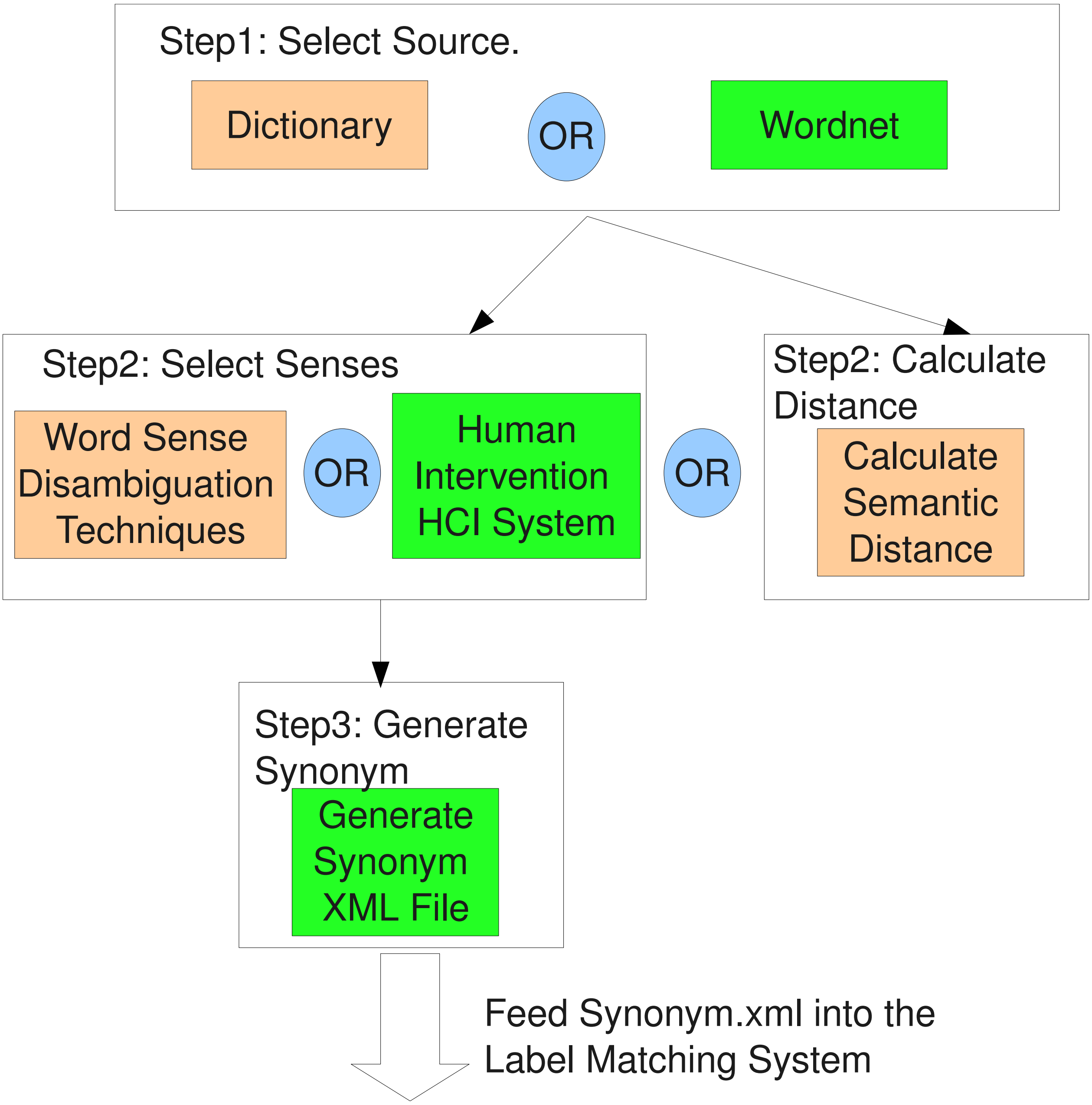

Step1: Select Source.
Dictionary
OR
Wordnet
Step2: Select Senses
Word Sense Disambiguation Techniques
OR
Human Intervention HCI System
OR
Step2: Calculate Distance
Calculate Semantic Distance
Step3: Generate Synonym
Generate Synonym XML File
Feed Synonym.xml into the Label Matching System

Figure 3.2: Semantic Stage Design Rationale

forms a Synset. WordNet is basically a collection of these Synsets. The various synsets are related to each other using different kinds of pointers or relationships for example hypernym, hyponym, holonym, meronym etc. It is important to note that the various relationships are between the various senses of two words and not between the words themselves. Various applications like OpenOffice (Corp 01) and Memidex dictionary (Memi 09) use WordNet to generate the thesaurus. Hence the proposed framework intends to use WordNet.

The second design decision is the way to use the WordNet to find the semantic similarity. There are three available options for this, the first is to use semantic distance algorithms to find the semantic similarity, the second is to use automatic word sense disambiguation to find the senses and synonyms and the last is to select the synonyms using manual intervention by the lecturer using HCI system. These three options are explained below along with the reason to choose or discard them. The proposed framework uses the third option of manual intervention by the lecturer using HCI system.

The first option is to use the semantic algorithms to calculate the similarity index between two words using the WordNet (Mill 95a; Fell 98). A summary of semantic algorithms is provided at (Pede 08; Pede 04). There is a distinction between semantically related and semantically similar (Patw 03; Sala 09). Semantically related takes into account all kinds of relationships for example synonyms, antonyms, hypernym, hyponym, holonym, meronym etc. So even two antonyms will be related to each other. On the other hand semantically similar takes into account only hyponymy and hypernymy relationships. These are applicable only to nouns and verbs as adjectives and adverbs don't have hyponymy and hypernymy relationships. So "semantically related" is a superset containing a subset "semantically similar". Some of the algorithms to calculate semantic relatedness are Hirst & St-Onge (Hirs 98), Lesk (Lesk 86), Adapted Lesk (Bane 02). Some of the algorithms to calculate semantic similarity are Wu & Palmer (Wu 94), Lin (Lin 98) and Path length (Pede 04), Leacock and Chodorow (Leac 98), Jiang & Conrath (Jian 97) and Resnik (Resn 95).

These algorithms use the shortest path between the two words and their Information Content to calculate the similarity index but differ on the actual formula that they apply. The Information Content (IC) of a word is a measure of the amount of information it represents.

There are different algorithms to calculate the value of IC. For example, the algorithm by Nuno Seco et al. (Seco 04b) use the number of hyponyms a word has to calculate its IC (Seco 04b). If a word has more hyponyms then that means the word is more generic and represents less specific information and hence should have low value of IC. So the IC value for a top root node will be the minimum while that for a leaf node will be the maximum. Another algorithm to calculate the semantic similarity is the combined text-to-text semantic similarity algorithm (Corl 05). It is the only semantic matching algorithm that is focussed on a group of words rather than the individual words. Some of the software systems that calculate the semantic similarity between two words using different algorithms are (Pede 09a; Seco 04a; Veks 07). Budanitsky has carried out experiments using WordNet to compare five different semantic distance algorithms and found Jiang and Conrath algorithm to be best (Buda 01).

The advantage of this approach is that it is easy to use and full automation is possible as it returns a similarity score. The disadvantage of this approach is its blackbox nature. The algorithms to calculate the semantic distance between two words are based on the shortest path between them. The WNConnect software (Fong 05) shows the shortest distance between two words in a graphical form. But this shortest path might have in between senses representing nodes which might be different from the sense in which the word was originally used in the label in the model solution. So this approach is blackbox in nature because the lecturer may or may not agree with the path taken to calculate this semantic distance. Because of the blackbox nature this approach is discarded.

Having discarded the approach to automatically calculate the semantic distance using the semantic algorithms, a second approach involves finding the synonyms from WordNet using the various sense. A word can be used in many senses and the task of resolving the sense in which a word has been used in a particular sentence is termed as word sense disambiguation (Sanf 98). As already discussed that the WordNet lists the various senses in which a word can be used and the synonyms corresponding to each of these senses. So the first step is to select the senses present in the WordNet which match the sense in which the word was used in the label in the model solution. This can either be done manually by the lecturer using an HCI system or automatically using word sense disambiguation techniques. For example the word “fare” may mean “eat well” as in the sentence “They fared sumptuously” or it may mean “the sum charged for riding in a public conveyance”. The technique of automatic word sense disambiguation uses the neighbouring words to determine the sense in which a word is used. For example the sentence “Different fares are charged for peak and off peak services” contains the word “fare” alongwith “charge” and so it can be assumed that the word “fare” is used for “sum or amount” and not for “eat well”. But word sense disambiguation is an ongoing research problem (Pede 09b) and it becomes more complex for words in diagram labels as the diagram labels consist of few words. Since it is difficult to apply the word sense disambiguation techniques to diagram labels, so manual intervention by lecturer using an HCI system is preferred the details of which have already been explained in the beginning of the Section 3.4.3 of this chapter.

The advantage of manual intervention by the lecturer to select the senses over the previous approach of automatically calculating the similarity index using semantic algorithms is its white box nature. The user has control over the sense that should be used to find the synonyms. The manual intervention has the disadvantage that it will involve lecturer time, but this time can be reduced by using the efficient HCI systems for example the tree based selection. Moreover it is a one time task and needs to be done only for the labels in the model solution which are generally not many. Also, the economies of scale will make this approach more beneficial if the same coursework is used multiple times.

3.4.4 Combined similarity stage

This stage calculates the combined similarity index (ComSI) value for each pair of labels by using the synonym XML file mentioned in the previous section and reexecuting the syntax stage algorithm.

3.4.5 Analysis stage

This is the last stage of the framework and for each label in the student diagram it outputs the corresponding matching label in the model solution or null if none is found. The input to this stage is the cartesian product set of the labels in the model solution and the student diagram along with the value of similarity index for each of these pairs. The following steps explain the method used by this stage to select the matching pair of labels.

- Input the similarity index matrix for the cartesian product of labels in the model solution and the student diagram. If there are n and m labels in the model solution and the student diagram respectively then this will be a n × m matrix.

- For each Label L_i in the model solution (each row in the similarity index matrix), select the corresponding label L_j (represented in columns) from the student diagram such that the pair has maximum value for the similarity index $ComSI(L_i, L_j)$.

- So if there are n labels in the model solution, n pairs will be selected each consisting of one label from model solution and one label from the student diagram. If the number of labels in the student diagram (m) is less than the number of labels in the model solution (n), then for some labels in the model solution the corresponding label from student diagram will be null. Also In case of duplicates where one label from the student diagram matches more than one label from the model solution, select the one with the higher similarity score or any one randomly if similarity scores are equal and then recalculate the matching pairs. After recalculating the matching pairs again check for duplicates and recalculate if duplicate exists. This step of recalculating the matching pair should be repeated until each label in the student diagram matches only one label of the model solution.

- Read the threshold value represented by $T(ComSI)$ from the configuration XML file. The value of this threshold used for experiments in this Thesis is 0.6.

- Discard the pair of labels for which $ComSI(L_i, L_j) < T(ComSI)$

- The remaining pairs of labels are considered to be matched. Each student diagram label L_j present in these matching pairs is considered to be correct and the corresponding label L_i from model solution is returned. The rest of the student diagram labels not present in the matching pairs are considered as incorrect and null is returned for them.

	La1	La2	La3	La4
L1	0.40	0.10	0.90	0.82
L2	0.20	0.30	0.95	0.80
L3	0.25	0.30	0.70	0.81

Table 3.7: Similarity matrix

The following example explains the analysis stage. The model solution in this example has three labels namely (L1, L2 L3) while the student diagram has four labels namely (La1, La2, La3, La4). The similarity matrix for the cartesian product of labels in the model solution and the student diagram is shown in the Table 3.7. The maximum value of the similarity index in each row has been highlighted. The matching pair of labels are (L1,La3), (L2,La3) and (L3,La4). But two labels in the model solution can't match the same label in the student diagram which is the case here as both the labels L1 and L2 in the model solution are matched with the same label La3 in the student diagram. Since the pair (L2, La3, 0.95) has higher similarity index value than (L1, La3, 0.90) so L2 is matched with La3 and a second maximum is chosen for L1 which is La4 with a value of 0.82. Now the matching pairs becomes (L1,La4), (L2,La3) and (L3,La4). But the labels L1 and L3 are matched with the same label La4 so the pairs need to be recalculated. Since the pair (L1, La4, 0.82) has higher similarity index value than (L3, La4, 0.81) L1 is matched with La4 and a second maximum is chosen for L3 which is La3 with a value of 0.70. But La3 is already matched with L1 with similarity index value higher than 0.70 so a third maximum is choose for L3 which is La2. So the matching pairs of labels are (L1, La4), (L2, La3) and (L3, La2). Now the threshold value of 0.6 is applied and the pairs having similarity index less than the threshold value are discarded. The intermediate output of the analysis stage before applying the threshold is as follows.

L1 La4 0.82

L2 La3 0.95

L3 La2 0.30

The final output of analysis stage after applying the threshold of 0.6, is

L1 La4 0.82

L2 La3 0.95

L3 None

Since the pair (L3, La2) has similarity index value less than the threshold it is discarded. So the final matching pairs are (L1, La4), (L2, La3) and (L3, null). The labels L1 and L2 in the model solution match the labels La4 and La3 in the student diagram while there is no match in the student diagram for the label L3 in the model solution. So the labels La3 and La4 in the student diagram are flagged as correct while the labels La1 and La2 are flagged as incorrect.

The algorithm used in the analysis stage always selects the local maximum in each row. However sometimes selecting the second maximum from a row may lead to an overall better match. For example, in the Table 3.7 if the value for (L2,La4) is changed from 0.80 to 0.94, then the same match as before is obtained which is (L1 La4 0.82), (L2 La3 0.95) and (L3 La2 0.30). The overall similarity score for this match is 2.07 (0.82+0.95+0.3). But selecting second maximum for the L2 (selecting 0.94 instead of 0.95) would result in the matches (L1,La3, 0.90), (L2,La4, 0.94) and (L3, La2, 0.30). This is a better match because the overall similarity score is 2.14 (0.90+0.94+0.3) which is more than overall similarity score obtained by selecting the local maximum for each row. So there can be a tradeoff between local maximum in each row and the overall similarity score. This issues requires further work and is an area for future work.

3.5 Design patterns and Software prototype

As explained before there are two non-functional requirements of the framework. Firstly the framework should be configurable, extensible, generic by design and not confined to an algorithm, an implementation of an algorithm or a set of parameters. This will allow the framework to be configured and extended by others. Secondly the framework while remaining

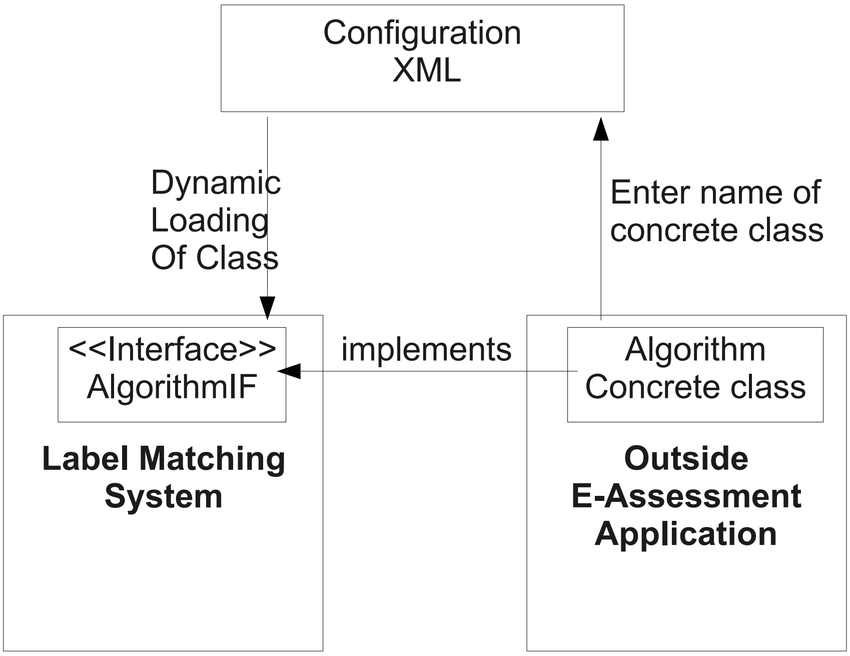


Figure 3.3: Strategy design pattern and dynamic loading

extensible at the same time should be simple to use and embed into other applications without much effort. To fulfil these two requirements configuration XML (W3C 08), dynamic loading of classes (Micr 99b; Micr 09) and two design patterns namely the strategy design pattern (Wiki 10g) and the facade design pattern (Wiki 10b), are used in the framework. These are explained below.

3.5.1 Strategy design pattern and dynamic loading

For the first requirement the Strategy design pattern (Wiki 10g) is used. According to Gamma et.al (Gamm 95), “The strategy pattern defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it. The strategy pattern is useful for situations where it is necessary to dynamically swap the algorithms used in an application”. The strategy pattern suits the requirement except that it requires the concrete implementations of the algorithm interface to be defined at compile time. This limitation of the strategy pattern was overcome by removing the name of the concrete class from the framework code and using the dynamic loading of classes (Micr 99b; Micr 09). The name of the concrete class for an algorithm is mentioned in the configucation XML file while the actual concrete class is provided either as a Java Archive JAR file (Corp 10b) or as a Java class file (Micr 99a). The framework gets the name of the concrete class of the algorithm from the configuration XML file and accesses the class using the Java class path (Micr 04). The client can create a new concrete implementation of the algorithm interface and add its name in the configuration XML file. Figure 3.3 shows this process.

The interface for algorithms being used during the various stages is defined. The users should create concrete classes for the algorithm of their choice by implementing these interfaces. The framework uses these interfaces, a configuration file containing names of the concrete classes and a dynamic binding mechanism to load at runtime the externally provided concrete implementations of the various algorithms. All the concrete implementation of the algorithms are outside the framework.

Users can extend the framework by writing a concrete implementation for any algorithm and adding the entry in the configuration file. So, for example, the following steps need to be carried out in order to use the Porter stemming algorithm (Port 06) in the framework. The actual code to do so has been provided in the Appendix Section 6.7.

- Write the concrete class for the Porter stemming algorithm. This class should implement the StemmingInterface which has been defined in the framework and contains the abstract method getStem(String srt1).

- Compile the file and place the class file inside the framework.

- Add the name of this class to the configuration XML file. The framework then automatically reads the name of the concrete class from the configuration xml file, loads it, and calls its methods at runtime.

3.5.2 Facade design pattern

Facade design pattern has been used to provide a simple interface to the diagram label matching framework (Wiki 10b). The facade class consists of following method which the user can call.

```
//string[0][0]=label in student diagram
//string[0][1]=matching label in the model solution, this is null if none is found
//string[0][2]=similarity index
String[][] getMatchedLabels(String xmlStudentDiagram, String xmlModelSolution)
```

3.5.3 Software Tool

An open source software tool to automatically mark the diagrams has been developed using the framework proposed in the Thesis. The Java source code for this prototype implementation is shown in Appendix 7. The Java code, Java documentation javadoc and netbeans project folder of the prototype has also been published at (Jaya 10) so that it is easy to download and reuse.

This tool, called "DiagramAssessmentTool.jar", consists of the following four components. Each component is a separate jar file. The software also has a configuration XML file which can be used to customize various algorithms and parameters.

- UMLDiagramXMIAPI.jar
  This component extracts the labels from UML diagrams in XMI format.

- GenericLabelMatcher.jar
  This component matches the labels.

- GenericLabelMatcherConcreteClasses.jar
  This component has the concrete implementations of the algorithms used by the GenericLabelMatcher component.

- GenericLabelMatcherInterface.jar

This component has the interface for the algorithms used by the GenericLabelMatcher component. The user needs this component containing interfaces in order to compile the concrete implementation of the algorithms.

The design of this software prototype has the following properties.

- XML Based

  The prototype is based on the XML format. The input to the framework is a diagram expressed in XMI format (OMG 07b) which is a standard format to express UML diagrams. The prototype can be extended to accept diagrams as input in various formats for example JPEG, GIF, SVG etc. (Mian 99) but it is a topic for further work and will be discussed in Chapter 5.3.

- Configurable for various algorithms and parameters

  The prototype has a main controller class and a set of interfaces for the algorithms. It is independent of any concrete implementation class for any of the algorithms. The concrete algorithm classes are outside the prototype system and the users can provide their own implementations of the interfaces for the algorithms and plug them into the system. The users can also customize the various parameters like special characters, stopwords etc. All this is done via a configuration XML file, an example of which is presented in the Appendix Section 6.3.

3.6 Summary

In this chapter a framework to match the diagram labels for e-assessment has been proposed. The chapter started by first discussing the design research methodology used in this Thesis. Then the functional and non-functional requirements of the framework were explained. The six different stages of the framework were then explained which fulfil the functional requirements. The design of the software prototype implementation of the framework was then explained. This design fulfils the non-functional requirements of the framework.

The functional requirement of the framework is that it should compare the labels in the student diagram with the labels in the model solution and produce a set of matching pairs of labels. The proposed framework has five stages explained in Section 3.4 of this chapter which fulfils this requirement. The *first* stage disambiguates the labels to produce a set of cleaned labels that are used for the subsequent stages. The *second* stage runs the syntax algorithms on these cleaned labels to produce a matrix for the syntactic similarity index. The *third* stage uses WordNet to produce a list of synonyms in the form of a synonym XML file. The *fourth* stage uses this synonym XML file and re-executes the *second* syntax stage to produce a matrix for the combined similarity matrix. The final *fifth* stage analyses this combined similarity matrix and marks the labels in the students' diagrams as correct if a match can be found in the model diagram and incorrect otherwise.

Note that the fourth stage of the framework reruns the second syntax stage but with the synonyms XML file. This makes the second syntax stage of the framework redundant. However the second stage is kept as a separate stage in the framework for two reasons. Firstly it is the sequence in which the framework was evaluated, first the syntax algorithms were run alone, and then further stages were added to enhance the label matching process. Secondly keeping the second and fourth stage separate allows the user to see the effect of syntax algorithms and semantic stage separately.

The non-functional requirements of the framework are that it should be configurable, extensible, generic by design and not confined to an algorithm, an implementation of an algorithm or a set of parameters. The software prototype design explained in the Section 3.5 of this chapter fulfils this requirement. The framework is a set of interfaces for various components and a configuration XML file which has the entries for the names of the concrete classes and value of various parameters. The framework provides concrete classes and parameters for the default behaviour so that it can be used straight out of the box by the users but the user is able to override these configurations by modifying the configuration XML file and define the algorithms and parameter values of their choice.

In order to evaluate the proposed framework a software prototype was implemented. This prototype was used to carry out experiments on a coursework at a UK HEI. The next chapter will discuss the results of these experiments.

Chapter 4

Results and Discussion

4.1 Introduction

Having proposed a configurable and extensible framework to match the labels in the previous chapter, in this chapter the results of experiments carried out using this framework are explained. The first section of this chapter explains the data collection process for the corpus of student coursework that was collected to carry out experiments in this Thesis. The experiments were then carried out using this corpus to first empirically measure the scale of the label matching problem and then to evaluate the effectiveness of the proposed framework to solve this problem. The second section explains the results of the experiments to measure the scale of the label matching problem. These results show that the problem of labels is substantial and cannot be easily avoided for the e-assessment of diagrams. These results also indicate a set of syntax *and* semantic algorithms will be required to solve this problem. The third section explains the categorisation of the data carried out manually. Here each label in the student coursework has been assigned to a category depending upon whether it would require just the syntax algorithms, just the semantic algorithms, or a combination of both in order to be matched to the correct label in the model solution. These results provide an upper bound for the performance of a perfect label matching process over the corpus of coursework used in this Thesis. The fourth section presents the analysis of the performance of two pre-processing stages namely auto correct and abbreviation expansion. These results show that the auto correct stage was quite effective but the abbreviation expansion stage did not perform well. The fifth section discusses the results of four syntax algorithms. These results show that the hybrid syntax algorithm explained in Section 3.4.2 of the previous chapter performs comparatively better than the three existing syntax algorithms. The last section analyses the results of using WordNet for semantic matching which shows that WordNet is only marginally effective for finding synonyms.

4.2 Data collection

Data was collected from coursework for second year Computer Science undergraduates at Brunel University. The coursework and the model solution provided by the lecturer are presented in the Appendix Section 6.5. The coursework description consisted of three paragraphs of text explaining the requirements for a bus travel card system and the students were required to draw a UML activity diagram (OMG 07a) for this problem. They were free to draw the diagrams either at home or in the labs over a period of a month. The students created a project for the UML activity diagram (OMG 07a) in the Borland Architect CASE tool (Toge 08) and submitted the complete project folder as a single compressed file through WebCT Virtual Learning Environment (VLE) (Clar 02). Initially 193 compressed student coursework files each containning an UML activity diagram were received. But unfortunately some of the compressed files could not be opened because they were corrupted and some did not contain a UML activity diagram. After removing all such files, 160 student coursework files each containing a UML activity diagram were obtained.

These files were then uncompressed and opened in Borland Architect CASE tool (Toge 08). The UML XMI (OMG 07b) (Fran 03) files were then extracted using the Borland Architect export utility for each of the student coursework projects. A Java program was written to parse these XMI files to extract the labels present in the diagrams. The framework explained in the previous chapter takes these UML XMI files (OMG 07b) as input.

This corpus of student coursework was large scale with 160 students and realistic because students did it in a natural setting for a real undergraduate module assessment. The

Item	**Count**
Total number of student diagrams	160
Total number of labels in all the student diagrams	2013
Mean number of labels per student diagram	12.58
Number of labels in the model solution	8
Mean number of words per label in the student diagram	3.06
Mean number of words per label in the model solution	1.88

Table 4.1: Basic Data

coursework created by the lecturer and the submissions by students were not affected by the experiments carried out in this thesis. The lecturer did not know beforehand that the coursework would be used for experiments and so the coursework creation was unbiased. Nor were the students told of this experiment beforehand. After the students had submitted the coursework there was an opportunity to use it for experiments carried out in this Thesis. Also 90% of the students in this module were UK home students and therefore had sufficient level of proficiency in the English language. So this corpus of student work was chosen for experiments as it was large scale, realistic and representative of the UK HEI home students. Table 4.1 summarises the raw data that was used in this Thesis for experiments.

4.3 Proliferation of synonym experiment results

4.3.1 Corpus of coursework

The corpus of coursework has been published in the following three formats at (Jaya 10).

- Borland Architect CASE tool project(Toge 08)

  This is the original format in which students submitted their coursework.

- XMI Format

  The XMI XML file corresponding to each student diagram was manually extracted

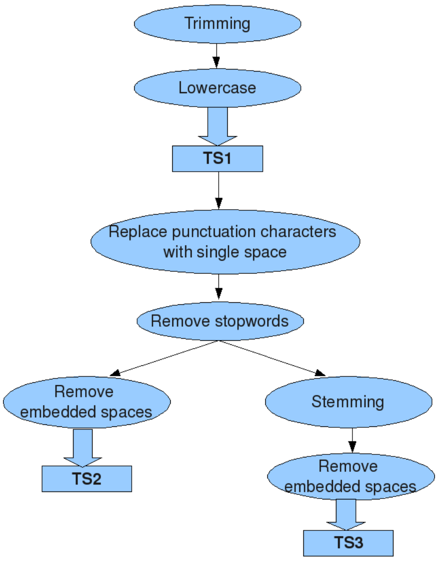


Figure 4.1: Text Transformation Processing

Text Transformation	**Before**	**After**
Trimming	“ Update Balance ”	“Update Balance”
Lowercase	“Update Balance”	“update balance”
Replace punctuation characters with single space	“update_balance”	“update balance”
Remove stopwords	“display the charge”	“display charge”
Remove embedded spaces	“process card”	“processcard”
Stemming	“processing”	“process”

Table 4.2: Text Transformations

Text Transformation Sequence	**Count**	**%**
Total number of labels	2013	100%
(Do Nothing) Total number of unique labels	773	38.4%
(TS1) Case and space trimming	638	31.7%
(TS2) Punctuation and stop words	571	28.4%
(TS3) Stemming	537	26.7%
Total number of unique correct label synonyms from TS3	358	17.8%

Table 4.3: Text Transformation Impact upon Label Count

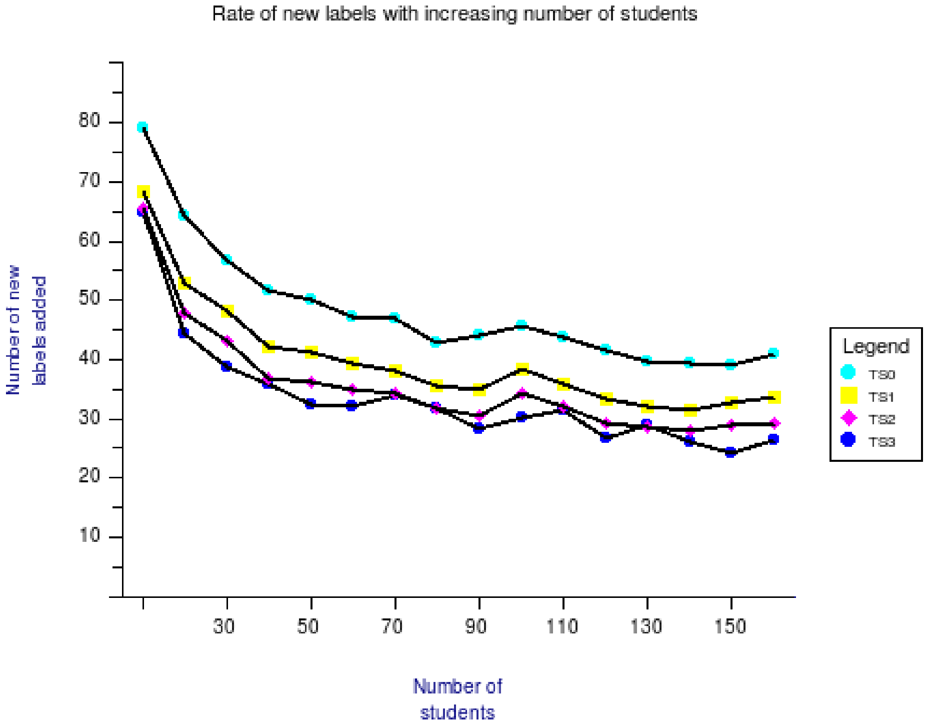


Figure 4.2: Rates of New Labels with Increasing Numbers of Student Coursework

using the Borland Architect CASE tool XMI export utility. All these XMI files have been published as a compressed zip file.

- JPEG PDF Format

  To make it easy to see all the student diagrams, the utility PreMark (Jaya 07a) was used to generate a single PDF file containing all the student diagrams in JPEG format arranged according to student identification number. Another PDF file was also generated using the (Jaya 07a) containing all the student diagrams in JPEG format arranged according to their file size. These two PDF files have also been published as part of the corpus.

To empirically measure the problem of diversity of labels used by the students, first the effect of basic text manipulation techniques in reducing the number of unique labels was assessed. Then the impact of scale was explored by measuring the number of new unique labels added per 10 students. From the dataset of 160 student coursework diagrams explained in the previous section all the labels were extracted using a Java program that parses the XMI files. Three sequences of text processing were then applied to these labels. These sequences are summarised in Fig. 4.1 where the ellipses denote specific text transformations such as trimming. However, there are ordering issues so the combination of transformations are referred to as transformation sequences (TS) and these are denoted by rectangles. Following are the three TS applied to the dataset. Table 4.3 gives examples of each individual text transformation.

- Do nothing: no processing of the labels extracted from the diagrams.
- Transformation Sequence 1 (TS1): this involves trimming the labels and converting them to lowercase, so for example, the terms “Update Balance ” and “Update Balance” would be transformed into “update balance”.
- Transformation Sequence 2 (TS2): this involved first replacing the punctuation characters like underscore with a single space, then removing the stopwords and then finally

removing the embedded spaces[1]. Stopwords are very common words like "to" and "the" that can be ignored whilst comparing labels (Wiki 10f). So "update_balance" is first converted to "update balance" and then to "updatebalance", "display the charge" is first converted to "display charge" and then to "displaycharge" and "process card" is converted to "processcard".

- Transformation Sequence 3 (TS3): this differs from TS2 in that the stemming text processing must be performed prior to removing embedded spaces. This is because the stemming algorithm which reduces a word to its root form cannot deal with concatenated words hence embedded spaces are essential to delineate each word.

The results for the effect of the transformation sequences and the impact of scale are explained in the following two subsections. These results show that the diagram label matching is a substantial problem and cannot be easily avoided for the e-assessment of diagrams. The finding implies that a set of better syntax and semantic similarity algorithms would be required to solve the problem.

4.3.2 Effect of basic text manipulation techniques

Table 4.3 indicates the impact of the basic text manipulation techniques to reduce the number of labels. As can be seen, the most effective of these strategies is TS3 which includes word stemming. This has the positive effect of reducing the number of unique labels by almost 73%, however, in practice this still leaves us with 537 unique labels which has a considerable impact if these must be examined manually. Unfortunately this task can not be ignored since 358 labels out of those produced by TS3 transformation have been judged as correct

[1] Embedded spaces were removed for the following reason. We needed to identify synonyms but the student labels did not always contain spaces between words, for example "invalidbeep". To avoid treating "invalidbeep" and "invalid beep" as different labels we removed the embedded spaces. Another solution would be to split "invalidbeep" into two separate words but for this we would require the automatic correction of words using a spell checker. Unfortunately spell checkers are not always accurate and can lead to over-correction, for example the Open Office spell checker auto corrects the label "cardreader" to "car dreader". Hence we decided to remove the embedded spaces from all labels.

by the human marker. So in the absence of further automation the human marker would have to deal with a total of 537 labels which can be time consuming.

4.3.3 Impact of scale

To explore the economies of scale factor the cumulative effect of adding ten new diagrams at a time on the number of new unique labels added was measured. These sets of ten new diagrams were randomly selected without replacement from the pool of 160 diagrams. Since the order in which the sets of 10 diagrams were selected from the pool of all diagrams might be influential the randomisation and cumulative analysis was repeated 30 times. The results as shown in the line plot in Figure 4.2 indicate the number of new unique labels added per 10 students so as to present the effect of increasing the number of students. The three line plots represent three levels of text processing. As can be seen in the Figure 4.2 the number of new unique labels added tends to decrease as the number of students is increased. This is not particularly surprising since increases in the numbers of label collisions is expected, i.e. picking a non-unique label. However, there also appears to be a tendency to flatten out from about 90 students onwards. The disconcerting issue here is that even after Level 2 text processing there is little evidence that new unique labels are being added at a rate of less than 30 per set of 10 additional student diagrams. Nor does this rate appear to be declining. So overall the cumulative growth of synonyms only shows a limited tendency to reduce at the margin despite using a range of text processing techniques.

4.4 Manual categorisation of data

Having established the scale of the label matching problem in the previous section, it is clear that a set of syntax and semantic algorithms is required to solve the problem. The human marker has marked all the student labels as correct or incorrect. If this marking were to be done automatically using various syntax and semantic algorithms, then some of the correct labels in the student diagram might be matched using just the syntax algorithm, but some would require a combination of syntax and semantic algorithms and some might require semantic algorithms alone. In order to find the percentage of such labels in each of these categories, manual categorisation of all the correct labels was done. This finds the upper bound for the performance of the best syntax and semantic algorithm.

The manual categorisation was carried out by myself for all the student labels that were marked as correct by the human marker. This categorisation was then reviewed by two members of the academic staff at Brunel University, one of them being the original human marker who had categorised each label as correct or incorrect. The manual categorisation can be subjective and may differ from person to person, so it is important to explain the process followed for categorisation as clearly as possible. This has been done by first listing a set of transformation rules and then listing the manner in which these transformation rules were applied to categorise each correct label. The following 15 transformation rules were used during the categorisation process. These transformation rules have not been implemented automatically but were carried out manually for the experiments. Also the transformations T9 and T10 are high level and difficult to automate, but have been included to clearly explain the categorisation process.

Label cleaning Transformations

- T1: convertToLowercase:(label) → label

  This transformation converts the label into lowercase.

- T2: breakIntoWord:(label) → label

  Sometimes a label contain two or more separate words but in a concatenated form. This transformation breaks them into separate words. For example "validbeep" is broken down into "valid beep".

- T3: correctSpelling:(label) → label

- T4: removeStopword:(listOfStopword, label) → label

- T5: removeSpecialCharacter(listOfSpecialCharacter, label) → label

- T6: removeDigits:(label) → label

  For example the labels “valid-beep0” and “valid-beep1” are converted into “valid-beep”

- T7: expandAbbreviation(listofAbbreviation, label) → label

  Semantic Plus Syntax Transformation

- T8: replaceWordBySynonymWord(Synonym, word) → word

  This is a word level transformation and is applied repeatedly on each word of the label. This means replacing individual words in a label with the corresponding synonym, hypernym or hyponym word. For example “update amount” is replaced by “update total”, “amount” being synonym of “total”. The hypernym of a word is its more generic form or a super category. For example, with “process card type” and “check card type”, “process” is a hypernym of “check”. The hyponym of a word is its less generic form or a sub category. For example, with “check hours” and “check time”, “hour” is a hyponym of “time”.

  Semantic Transformations

- T9: replaceLabelBySynonymLabel(listOfSynonymLabel, label) → label

  This transformation replaces the label with a synonym label. For example the label “warning beep” is replaced by the label “invalid warning”.

- T10: replaceDecomposedLabelByHigherLevelLabel(listOfDecomposedLabels, label) → label

  Sometimes a label in the student diagram is in a decomposed form (differing level of decomposition) of a target label in the model solution. This transformation replaces such decomposed labels with the the corresponding target label. For example the label “display reason for error” is replaced by the label “invalid warning”.

- T11: replaceBySurroundingStateMatch(surroundingLabelsOfTargetLabel, label) → label

  Sometimes surrounding or nearby labels are used to match labels in a diagram. Surrounding labels are defined as the immediately adjacent labels. This transformation deals with such case. If all or most of the labels surrounding a label in the student diagram are same as those surrounding a label (known as target label) in the model solution, then this transformation replaces the label by the target label.

- T12: replaceByType(label, targetLabel) → label

  Sometimes there is only one start state in the student diagram and only one in the model solution. A student may have used a different label for this start state but the human expert will match it to the start state in the model solution as both can contain only one start state. This transformation takes care of such situations by replacing the label in student diagram with the target label in the model solution if both have the same type and a maximum of one is allowed in a diagram.

  Syntax Transformations

- T13: removeSurplusWord(label * targetLabel) → label

  The surplus words are the words that are present in the label but not present in the target label. This transformation removes all such surplus words from the label.

- T14: arrangeWordOrder(label, targetLabel) → label

  This transformation arranges the words in the label according to those in the targetlabel. So if label = "time check" and targetLabel = "check time" then this transformation reverses the word order in the label and returns "check time".

  Result Transformation

- T15: equal(label, targetLabel) → boolean

  This transformation returns true if two labels are exactly same, else it returns false.

4.4.1 Categorisation process

The manual categorisation was carried out by applying the above mentioned transformation rules in the following manner. Tables 4.4 and 4.5 summarize the main categories and subcategories respectively.

- Cleaned Label: Input label L and apply transformations T1 through T7. Store the output label as L′. This is the cleaned label on which all further transformations will be applied.

- Syntax Only: Apply T13 through T15 on L′. If true is returned then categorise the label L as “Syntax Only”.
- Semantic Plus Syntax: Apply T8 on each word of L′ and then on its output apply T13 through T15. If true is returned then categorise the label L as “Semantic Plus Syntax” with the sub category as “Sem_Synonym”.
- Semantic Only: Apply T9 on L′ and then on its output apply T15. If true is returned then categorise the label L as “Semantic Only” with the sub category as “Sem_Synonym”.
- Semantic Only: Apply T10 on L′ and then on it’s output apply T15. If true is returned then categorise the label L as “Semantic Only” with the sub category as “Sem_DifDec” (Differing level of decomposition).
- Semantic Only: Apply T11 on L′ and then on its output apply T15. If true is returned then categorise the label L as “Semantic Only” with the sub category as “Sem_Sur”.
- Semantic Only: Apply T12 on L′ and then on its output apply T15. If true is returned then categorise the label L as “Semantic Only” with the sub category as “Sem_Typ”.

The labels categorised as “Syntax only” can be matched using just the syntax algorithms. The labels categorised as “Semantic Plus Syntax” would require a set of semantic and syntax algorithms to be matched while the labels categorised as “Semantic only” would require semantic algorithms to be matched. The labels categorised as “Syntax only” are syntactically similar to the correct label. For example the labels “ceck card type” and “check card type” were categorised as “Syntax Only” because they are syntactically very similar as they just differ in the spelling of the word “check”.

The labels categorised as “Semantic Plus Syntax” are also syntactically similar to the correct label but to a lesser extent than those categorised as “Syntax only” because they also have words that are semantically same but syntactically different than those in correct label. For example the labels “identify card type” and “check card type” were categorised as “Semantic Plus Syntax” because although they have the common part “card type”, but the first label also has the word “identify” which is semantically the same but syntactically different from “check”.

The third category of labels is “Semantic only”. Such labels are syntactically different to the correct label but have the same semantic meaning. For example the labels “validate” and “check card type” have no syntactic similarity but the same semantic meaning.

All the labels categorised either as “Semantic Plus Syntax” or “Semantic Only” have been further divided into four types depending upon the reason for their semantic similarity. The first subcategory is “Sem_Synonym” which means that the label in the student answer and in the model solution are either synonyms or hypernyms or hyponyms for example the labels “identify card type” and “check card type”. The second subcategory is “Sem_DifDec” which means that the student label is a more detailed form of the label in the model solution. The reason for having this subcategory is because sometimes the students provide more detailed answers than the model solution and so there is a subsumption relationship between one label in the model solution and a set of many labels in the student answer. The human expert will give marks for those lower level labels in the student answer which when composed add up to form a label in the model solution. For example the labels “validate” and “check card type” are subcategorised as “Sem_DifDec” because “validate” is a more detailed form of “check card type” as validation is done while checking the card type[2]. The third subcategory is “Sem_Sur” which means that the label in the student answer and in the model solution have the same set of surrounding labels. For example the labels “card type” and “check card type” were matched by the human expert and subcategorised as “Sem_Sur” as both had the same surrounding labels. An issue in matching labels on the basis of surrounding labels is that sometimes only a proportion of the surrounding labels may match because some labels are missing or additional labels are included. This was not an issue during categorisation as

[2] The categorisation can be subjective and may differ from person to person. It was reviewed by two members of the academic staff at Brunel University.

Main Category	**Example**
Syntax only	"ceck card type" and "check card type"
Semantic Plus Syntax	"identify card type" and "check card type"
Semantic only	"validate" and "check card type"

Table 4.4: Main Categories

it was done manually and human judgement was used to resolve it. The last subcategory is "Sem_Typ" which means that the label in the student answer and in the model solution have the same state type. The reason for having this subcategory is because sometimes there is only one start state in the student solution and only one in the model solution. The student may have used different labels for this start state but the human expert will match it to the start state in the model solution as both can contain only one start state. For example the labels "cockle cards" and "startstate" were matched by the human expert and subcategorised as "Sem_Typ" as both were start states.

4.4.2 Categorisation results

Table 4.6 summarizes the result of the manual categorisation process. As shown in this table, only 14.7% of the total correct labels can be matched using syntax algorithms alone, while an additional 7% of correct labels would be identified by a syntax algorithm if combined with a semantic algorithm. So overall the syntax algorithms would be useful in matching 21.7% or just over a fifth of the correct labels. This implies the importance of semantic algorithms in matching labels without which e-assessment of diagrams containing labels will be difficult.

Table 4.7 shows the manual categorisation for different kinds of semantic relationships. As can been seen in this table the semantic relationship "Differing level of decomposition" dominates by accounting for 63% of the correct labels, while the semantic relationship "Syn-

Semantic Subcategory	**Description**	**Example**
Sem_Synonym	Synonym or Hypernym	“identify card type” and “check card type”
Sem_DifDec	Differing Level of Decomposition	“validate” and “check card type”
Sem_Sur	Match because of the Surrounding states	“card type” and “check card type”
Sem_Typ	State Type Match	“cockle cards” and “start-state”

Table 4.5: Semantic Subcategories

Ref	Count	%
Total Correct	429	100%
Syntax Only	63	14.7%
Semantic Plus Syntax	30	7%
Semantic Only	336	78.3%

Table 4.6: Manual Categorisation Result Summary

Ref	Count	%
Total Correct	429	100%
Syntax Only	63	14.7%
Sem_Synonym	64	15%
Sem_Sur	28	6.5%
Sem_Typ	4	0.9%
Sem_DifDec	270	63%

Table 4.7: Manual Categorisation Result Detailed

Reference	Cause Of Misspelling
S	Actual Spelling mistake
C	Concatenation
C and S	Concatenation and Actual Spelling mistake
A	Abbreviation
C and A	Concatenation and Abbreviation
N	Not a spelling mistake

Table 4.8: Cause of misspelling

onym" comes second by accounting for 15% of the correct labels. This Thesis only analyses syntax algorithm and semantic algorithm to find synonyms which combined accounts for 29.7% of the correct labels. The remaining 70.3% of the correct labels require other semantic algorithms e.g. to deal with the differing level of decomposition, which are important but out of scope of this Thesis and are a topic for further work.

4.5 Preprocessing stage results

The following are the results from the pre-processing stage of spell check and abbreviation expansion.

4.5.1 Spell check results

The spell checker can flag a word as misspelled for six main reasons as summarized in Table 4.8. For example, it may be an actual spelling mistake as in "ceck", a concatenation of words or an abbreviation or sometimes the spell checker can flag a correctly spelled word as misspelled word for example "offpeak". Also the spell checker often suggests multiple words having different meanings for a supposedly misspelled word and the first word suggested by

Ref Type	**Student Label**	**Misspelled word**	**First suggested word by spell checker**	**Correct by human**
S	ceck card type	ceck	check	Yes
C	acceptedbeep	acceptedbeep	accepted beep	Yes
C and S	cheakdate	cheakdate	cheapskate	No
A	display error msg	msg	mg	No
C and A	check expdate	expdate	exudate	No
N	offpeak	offpeak	off-peak	Yes

Table 4.9: Examples of misspelling

	Count	%
Auto-corrected labels matching human judgement	88	86
Auto-corrected labels not matching human judgement	14	14
Total number of labels detected by spell checker	102	100

Table 4.10: Results of Auto Correct Pre-processing Stage

Type Of Spell Problem	Number of Labels	Num Auto Correct word Match Human Judgement	Num Auto Correct word Does not Match Human Judgement
S	34	31	3
C	51	45	6
C and S	2	0	2
A	3	1	2
C and A	1	0	1
N	11	11	0
TOTAL	102	88	14

Table 4.11: Detailed Analysis of Auto Correct Pre-processing Stage

the spell checker may or may not be the correct one[3]. Table 4.9 shows an example of the six types of misspellings flagged by the spell checker, the first word suggested by it and if the first word suggested is the correct one as judged by a human expert.

As mentioned in the previous chapter 3 on the framework the Hunspell spell checker (Neme 10) was used to carry out the experiments in this Thesis. The spell checker was run on all the labels and the words flagged as misspelled were replaced by the first word suggested by the spell checker. The words flagged as misspelled by the spell checker were manually categorised into one of the six cases of misspelling as mentioned in Table 4.8. Also the first word suggested by the spell checker for each of the supposedly misspelled words was manually checked for correctness, meaning if it should be chosen to replace the misspelled word. The results of these two analyses are presented in Table 4.10 and Table 4.11.

As shown in Table 4.10 the spell checker flagged a total of 102 labels as containing misspelled words. After these misspelled words were replaced by the first word suggested by the spell checker and manually checked for correctness, 88 out of 102 labels matched the

[3] The correct word for a misspelled word means that a human expert will choose it to replace the misspelled word.

human judgement. This means that the spell checker did not work well in just 14 out of 102 labels.

As shown in Table 4.11, 11 labels flagged by the spell checker as misspellings were manually categorised as “N” meaning they had no spelling mistakes. However the first word suggested by the spell checker for all of these 11 words matched the human judgement as it had no negative effect. For example the spell checker flags “offpeak” as a misspelled word and suggests “off-peak” as the first word to replace it. Similarly it flags “todays” as a misspelled word and suggests “today’s” as the first word to replace it. The reason for no negative effect is that the first suggested word only introduced special characters which can easily be removed by rerunning the stage S1.2 explained in Section 3.4.1 of Chapter 3 which involves removing the special characters. Also Table 4.11 shows that 34 labels flagged by the spell checker as misspellings were manually categorised as “S” and 2 labels were manually categorised as “C and S” meaning they had actual spelling mistakes. So a total of 36 labels had actual spelling mistakes out of which 31 were successfully corrected by the spell checker as they matched the human judgement. Also Table 4.11 shows that 51 labels flagged by the spell checker as misspellings were categorised as “C” meaning they had no spelling mistakes but were actually concatenations of correctly spelled words. The spell checker successfully corrected 45 out of these 51 labels as they matched the human judgement. Also Table 4.11 shows that 4 labels flagged by the spell checker as misspellings were categorised either as “A” or “C and A” meaning they had no spelling mistakes but were actually abbreviations. The spell checker successfully corrected only 1 out of these 4 labels.

So the spell checker worked well for the cases where there was an actual spelling mistake (in 31 out of 36 such cases). It also worked well where there was no spelling mistake but the words were concatenated (in 45 out of 51 such cases). The spell checker did flag some of the correctly spelled words as misspelled but it had no negative effect in any of the such cases as it only introduced special characters which can easily be removed by rerunning the stage S1.2 explained in Section 3.4.1 of Chapter 3. The spell checker did not seem to work well in cases where there were no spelling mistakes but abbreviations. However this had only a minor effect as there were only 4 such cases. So using a spell checker to detect spelling

Abbreviated Word	Expanded form by human expert	Number of Unique Labels Abbreviated Word Occurs In	abbreviation found in abbreviation.com	Order in which correct expanded form found in abbreviation.com
msg	message	1	Yes	First
exp	expiry	3	Yes	Not found
calc	calculate	1	Yes	Fifth
paygcard	prepay card	1	No	Not found

Table 4.12: Result of Abbreviation Expansion Pre-processing Stage

mistakes and replacing them with the first word suggested by the spell checker had positive effect in 77 cases (actual spelling mistake, concatenation, abbreviation), was neutral in 11 cases (no actual spelling mistake) and had a negative effect only in 14 cases. This shows that the auto correct stage can be quite useful.

4.5.2 Abbreviation expansion results

As explained in the previous section the spell checker flags some of the abbreviations as spelling mistakes for example “msg” and “exp”. The results explained in the previous section show that the spell checker does not work well for abbreviations. So further experiments were carried out to find if the abbreviation expansion stage would be effective in handling abbreviations. In order to evaluate this, the words detected by the spell checker as spelling mistakes were manually checked to see if they were abbreviations. All such abbreviations were then put in the abbreviation expander and the first suggested abbreviation was manually checked against the human expert judgement for correctness. Table 4.12 shows these results. As can be seen in this table, a total of four abbreviations were found out of which only

Ref	Algorithm
A	Combined Hybrid Syntactic Algorithm
B	Simon Algorithm
C	Levenshtein Distance Algorithm
D	Q-gram Algorithm

Table 4.13: Syntax Algorithms

Label1	**Label2**	**Processed Label1**	**Processed Label2**	**A**	**B**	**C**	**D**
“valid bleep”	“valid beep”	“valid bleep”	“valid beep”	1	0.8	0.9	0.8
“card reader”	“read card”	“card read”	“read card”	1	1	0.3	0.45
“beep for validity”	“valid beep”	“beep valid”	“valid beep”	1	1	0	0.42
“time check”	“check time”	“tim check”	“check tim”	1	1	0.11	0.36
“cardreader”	“read card”	“card read”	“read card”	1	1	0.3	0.45

Table 4.14: Syntax Distance

one of the abbreviations could be resolved through the abbreviation expander. The correct expanded form according to the human expert of the remaining three abbreviations was either not found or if found then not in first place. These results show that there is a need for better abbreviation expander.

4.6 Syntax stage results

After executing the pre-processing algorithms mentioned in the previous section, a set of cleaned labels were collected. As a result of the pre-processing stage, some of the labels became duplicate. For example the labels “invalid beep” and “invalid_beep” were both converted to the label “invalid beep”. The duplicate labels were removed and syntax analysis was carried on the unique cleaned labels. The same process was done for the labels in the model solution. We evaluated four syntax matching algorithms which are listed in Table 4.13. The syntax algorithms were executed on the cartesian product of all these unique student labels and the labels in the model solution. Algorithms B, C, D are widely known and well established algorithms. Algorithm A has been proposed in this Thesis and its details can be seen in the previous framework chapter. These algorithms take two labels as input and return a similarity index value between 0 and 1, 0 being no match and 1 being a perfect match. Table 4.14 shows the similarity distance between some of the labels for the algorithms B,C and D. The similarity index for the labels in Table 4.14 according to algorithm A is 1. If the similarity index value for two labels is more than a certain threshold value they are considered matched. One of the limitations of this is that two or more labels belonging to the same student diagram may match to the same label in the model diagram. This Thesis does not address this issue but it can be handled by taking into account the surrounding labels. Also it will be less of a problem with small diagrams such as the one considered in our dataset.

The threshold value will have an effect on the accuracy of the algorithm. There are two important decisions to be made, to select the best syntax algorithm and set the optimum threshold value. The following sections consider these two questions.

True positives are the labels that are correct according to the human and also correct according to the syntax algorithm. False positives are the labels that are incorrect according to the human but are correct according to the syntax algorithm. Precision is one measures to find the accuracy of an algorithm and can be defined as the fraction of the actually correct values among all the values identified by an algorithm. Recall is one measures to find the effectiveness of an algorithm in identifying maximum number of correct values and can be defined as the fraction of total correct values identified by an algorithm. FScore combines the precision and recall and produces a single value (Wiki 10c; Wiki 10a). The formulae for precision, recall and FScore are as follows:

Precision =(True Positive)/(True Positive + False Positive)
Recall=(True Positive)/(True Positive + False Negative)

Algo (Precision)	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**
A	0.37	0.44	0.5	0.5	0.54	0.62
B	0.35	0.44	0.58	0.71	1	1
C	0.31	0.36	0.45	0.4	1	1
D	0.45	0.6	0.67	1	1	1
B ∩ C ∩ D	0.42	0.56	0.67	1	1	1
B ∪ C ∪ D	0.34	0.38	0.51	0.69	1	1
A ∩ B ∩ C ∩ D	0.42	0.59	0.8	1	1	1
A ∪ B ∪ C ∪ D	0.34	0.35	0.45	0.49	0.54	0.62
A - (B ∪ C ∪ D)	0.36	0.19	0.35	0.37	0.5	0.59
(B ∪ C ∪ D) - A	0.06	0.24	0.3	0.4	1	1
A - (B ∩ C ∩ D)	0.35	0.39	0.44	0.49	0.54	0.62
(B ∩ C ∩ D) - A	1	0.33	0	1	1	1

Table 4.15: Precision For All Threshold

FScore=2*Precision*Recall/(Precision + Recall)

Syntax Results

Tables 6.2, 6.4, 6.6, 6.8 in the Appendix Section 6.5.1 show the count of true positives, false positives, false negatives and true negatives respectively for the threshold values of 0.5, 0.6, 0.7, 0.8, 0.9 and 1.0. Tables 4.15, 4.16, 4.17 have been derived from these tables and show the precision, recall and FScore respectively for the threshold values of 0.5, 0.6, 0.7, 0.8, 0.9 and 1.0. Note that Table 6.2 shows that Algorithm A shows a steep rise in the number of true positives at a threshold of 0.6. This is because the internal threshold for algorithm A is 0.6. While producing these tables, the values found by the exact match algorithm were ignored so that their net effect could be evaluated. So for example in Table 6.2 the count for 0.9 threshold for the edit distance algorithm is 1, which means that the edit distance found one extra match at 0.9 threshold apart from those found by the exact match algorithm. This implies in terms of true positives, at the 0.9 threshold the edit distance algorithm is

Algo (Recall)	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**
A	0.36	0.1	0.08	0.08	0.08	0.08
B	0.17	0.11	0.07	0.04	0.01	0.01
C	0.14	0.08	0.03	0.01	0	0
D	0.1	0.05	0.03	0.01	0	0
B ∩ **C** ∩ **D**	0.09	0.04	0.02	0	0	0
B ∪ **C** ∪ **D**	0.21	0.13	0.07	0.05	0.01	0.01
A ∩ **B** ∩ **C** ∩ **D**	0.09	0.03	0.02	0	0	0
A ∪ **B** ∪ **C** ∪ **D**	0.36	0.15	0.1	0.09	0.08	0.08
A - (B ∪ **C** ∪ **D)**	0.15	0.01	0.03	0.04	0.07	0.07
(B ∪ **C** ∪ **D) - A**	0.01	0.05	0.02	0.01	0	0
A - (B ∩ **C** ∩ **D)**	0.27	0.06	0.06	0.08	0.08	0.08
(B ∩ **C** ∩ **D) - A**	0	0	0	0	0	0

Table 4.16: Recall For All Threshold

Algo (FScore)	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**
A	0.36	0.16	0.14	0.14	0.14	0.14
B	0.23	0.18	0.12	0.08	0.02	0.02
C	0.19	0.13	0.05	0.01	0.01	0
D	0.17	0.1	0.05	0.01	0	0
B ∩ **C** ∩ **D**	0.14	0.07	0.04	0.01	0	0
B ∪ **C** ∪ **D**	0.26	0.2	0.12	0.09	0.03	0.02
A ∩ **B** ∩ **C** ∩ **D**	0.14	0.06	0.04	0.01	0	0
A ∪ **B** ∪ **C** ∪ **D**	0.35	0.21	0.16	0.15	0.14	0.14

Table 4.17: FScore For All Threshold

only slightly better than the exact match algorithm. But at 0.7 threshold, the edit distance algorithm is significantly better than the exact match algorithm in terms of true positives.

All the values in the syntax results tables have been rounded to two decimal places, but while calculating the values based on these values the actual value has been used rather than the rounded value. For example in Table 4.16 the recall for algorithm C for threshold value of 0.9 and 1.0 is shown to be 0. But actually the recall value for 0.9 threshold is 0.002551 and that for a 1.0 threshold the value is 0.0. Hence in Table 4.17, the FScore for 0.9 threshold value is not equal to zero although the recall for 0.9 threshold is shown to be 0 in Table 4.16, the reason being that the actual recall value of 0.002551 has been used for calculating the FScore rather than the rounded recall value which is 0.

Note that Table 4.15 shows that for the threshold 1.0, the precision of algorithm A is 0.62 while the precision of Algorithms B, C and D are 1. This is because there are cases where the student label contains all the words present in a label of model solution (and hence algorithm A returns a similarity score of 1.0), but it has been marked incorrect by the human expert. For example, the similarity score according to algorithm A between the student label "wait for card to be read" and the model solution label "read card" is 1 but the human expert has marked the student label as incorrect.

4.6.1 Analysis for effective algorithm

The effectiveness of an algorithm can be measured in terms of precision and recall. Table 4.15 shows that algorithm D has the highest value of precision for all thresholds except for the threshold value of 0.7 for which the algorithm (A ∩ B ∩ C ∩ D) has the highest value. Table 4.16 shows that the algorithm (A ∪ B ∪ C ∪ D) has the highest recall value for all thresholds followed closely by the algorithm A. So algorithm A has high recall but low precision which is to be expected as there will always be trade-off between precision and recall. High precision can be achieved at the cost of high recall and vice verse. So an overall analysis based on a FScore value that combines precision and recall is more useful than analysing precision and recall separately. If only individual algorithms are considered then algorithm A has highest FScore value for all threshold values except 0.6 threshold for which algorithm B has slightly

	Count	%
Synonyms used by the students	17	100%
Found in WordNet	2	11.8%
Present in question text	6	35.3%

Table 4.18: Semantic Analysis of Synonyms

higher FScore value. If the hybrid algorithms are also considered, then the algorithm (A ∪ B ∪ C ∪ D) has highest FScore value for all threshold values except for 0.5 threshold for which the algorithm A has slightly higher FScore. So for the dataset in this Thesis the hybrid approach is best.

4.6.2 Analysis for optimal threshold

Setting the threshold is an important aspect. Table 4.17 shows that the maximum value of FScore for all algorithms is at threshold 0.5. This implies that out of the threshold values 0.5 through 1.0, 0.5 is the optimal threshold for all algorithms. Also Tables 4.15 and 4.16 show that lower values of the threshold result in high recall but low precision, while a higher value of the threshold results in high precision but low recall. There is a possibility that the FScore is higher for thresholds lower than 0.5. But lowering the threshold would further lower precision to the point of unusability.

4.7 Semantic stage results

Semantic analysis was carried out by first manually identifying all the synonyms used by students. Some of the synonyms were used by multiple students and so the duplicates were removed. These synonyms are shown in the Table 6.10 in the Appendix section 6.6. Two experiments were carried out for semantic analysis. First using the semantic approach explained in the Section 3.4.3 of Chapter 3. Second using the semantic similarity algorithms. The following sections describe the results of these experiments.

4.7.1 HCI system approach

The synonyms used by students were checked for in the WordNet (Mill 95b; Fell 98) using the HCI system approach explained in Section 3.4.3 of Chapter 3. Furthermore the text of the question was also checked for presence of these synonyms in order to evaluate the extent to which the choice of synonyms by students is affected by the text of the question. Table 4.18 shows these results. It may be noted that the total number of synonyms used by the students as shown in this section in Table 4.18 is just 17 compared to 64 shown in the previous section in Table 4.7. This means that students have used 64 unique labels which in turn contain 17 unique synonymous words as more than one label can contain the same synonym. For example two different student labels “update balance” and “update new balance” map to the same correct label “update total” and use the word “balance” as synonym for the word “total”. As shown in Table 4.18 only 2 out of 17 synonyms used by the students were listed in WordNet as synonyms and hence this approach seems to be only marginally effective to find the synonyms. One of the reasons for this is that WordNet is generic and not specific to a domain. Table 4.18 also shows that 6 out of 17 synonyms used by the students were found in the text of the question. It may be reiterated that in the coursework analysed in this Thesis, the lecturer had no prior information that it would be used for analysis and hence creation of the coursework and its administration was not influenced in any manner. This suggests that the text of the question would be an effective source to extract the synonyms used by students. This also suggests that a question should be drafted carefully because its text influences the choice of labels by students. Although such work is out of scope for this Thesis, it opens up potential areas for future work.

4.7.2 Semantic similarity algorithms

As explained in Section 3.4.3 of Chapter 3 the proposed framework discards the approach to use the semantic similarity algorithms because of their blackbox nature. But experiments were carried out to find the effectiveness of semantic similarity algorithms. Experiments were run on three semantic similarity algorithms namely Wu & Palmer (Wu 94), Lin (Lin 98) and Path length (Pede 04). These three algorithms were chosen because they have a range

Algorithm	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**
Wu & Palmer	8	6	6	5	4	1
Path length	3	1	1	1	1	1
Lin	3	3	3	2	2	1

Table 4.19: Synonyms found using the semantic similarity algorithms

between 0 and 1 and so it is easy to compare them. Other semantic similarity algorithms for example Jiang & Conrath (Jian 97) and Resnik (Resn 95) have ranges from 0 to a big number. The three chosen semantic similarity algorithms were run using the system available at (Pede 09a) on 15 synonyms out of the 17 used by the students. They could not be run on the remaining two synonyms because in one case the word "offpeak" was not found in WordNet and in another case the word "valid" was neither noun or verb. Recall from Section 3.4.3 of Chapter 3 that the semantic similarity algorithms take into account only hyponymy and hypernymy relationships and are applicable only to nouns and verbs as adjectives and adverbs don't have hyponymy and hypernymy relationships.

The semantic similarity algorithms return a score between 0 and 1. If the score was more than a certain threshold value, the words were judged to be synonyms. Six different threshold values from 0.5 through 1.0 were used. Table 4.19 shows the number of synonyms found using the three semantic similarity algorithms. These results show that Wu & Palmer (Wu 94) is best having found 8 out of 15 synonyms used by the students. These results also show that lower thresholds result in more synonyms being found and different semantic similarity algorithms have different optimal thresholds. The results show that semantic similarity algorithms can be effective to find the synonyms. However the semantic algorithms were run only on the synonyms used by the students and not on the whole corpus of coursework. In order to find the negative effect of these algorithms, they need to be run on all the words in the corpus of coursework. This will be interesting to evaluate and is a topic for future work. Since the framework is extensible, so the semantic similarity algorithms can be quite easily integrated into the system using the configuration XML file.

4.8 Summary

This chapter has discussed the results of the experiments carried out using the framework proposed in Chapter 3.

The first section of this chapter explained the data collection process for the corpus of student coursework that was collected to carry out experiments in this Thesis. Data was collected from coursework for second year Computer Science undergraduates at Brunel University. This corpus consisted of 160 student courseworks and was chosen for experiments as it was large scale, realistic and to some extent representative of the UK HEI home students.

The second and third section of this chapter measured the extent of the diagram label matching problem and showed that it is a substantial problem. The second section explained the results of the experiments carried out to empirically measure the scale of the label matching problem. These results show that the problem of labels is substantial and a set of syntax *and* semantic algorithms would be required to solve this problem. The third section explained the categorisation of the data carried out manually. Here each label in the student coursework was assigned a category depending on whether it would require just the syntax algorithms or semantic algorithms or a combination of both in order to be matched to the correct label in the model solution. These results were upper bound for the performance of a perfect label matching process over the corpus of coursework used in this Thesis. These results show that only 14.7% of the total correct labels can be matched using syntax algorithms alone, while an additional 7% of correct labels would benefit from the syntax algorithm if combined with a semantic algorithm. So overall the syntax algorithms would be useful in matching 21.7% or just over a fifth of the correct labels. This implies the importance of semantic algorithms in matching labels without which e-assessment of diagrams containing labels will be difficult.

The remaining section of this chapter evaluated the proposed framework and showed that that it is effective but only to limited extent and needs to be further refined for the semantic stage. The fourth section presented the analysis of the performance of two preprocessing stages namely auto spelling correction and abbreviation expansion. These results show that the auto correct stage was quite useful by correcting 86% of the total misspelled labels but the abbreviation expansion stage did not perform well as it could only expand one of the four abbreviations. The fifth section discussed the results of four syntax algorithms. These results show that the hybrid syntax algorithm explained in Section 3.4.2 of Chapter 3 performed better than the three existing syntax algorithms. These results also show that out of the threshold values 0.5 through 1.0, 0.5 is the optimal threshold for all the four syntax algorithms analysed in this Thesis. The last section of this chapter analysed the results of using the WordNet for semantic matching and showed that the WordNet is only marginally effective for finding synonyms and needs to be further refined.

The next chapter concludes the Thesis by summarizing, discussing the limitations and the opportunities arising out of this Thesis for further work.

Chapter 5

Conclusion, Limitations And Future Work

5.1 Summary of research

This Thesis has measured the extent of the diagram label matching problem and proposed and evaluated a configurable extensible framework to solve it.

In order to measure the extent of the diagram label matching problem experiments involving basic text manipulation techniques were run on 160 items of coursework for second year Computer Science undergraduates at a UK HEI. This corpus was chosen for experiments as it was large scale, realistic and to some extent representative of the UK HEI home students. First, the effect of basic text manipulation techniques in reducing the number of different labels was assessed. Then the impact of scale was explored by measuring the number of new unique labels added per 10 students. The results of these experiments have been presented in Chapter 4. These results show that the basic text manipulation techniques have the positive effect of reducing the number of unique labels by almost 73%, however, in practice this still leaves us with 537 unique labels which has a considerable impact if these must be examined manually. The results also show that there is little evidence that new unique labels are being added at a rate of less than 30 per set of 10 additional student diagrams. Nor does this rate appear to be declining. So the cumulative growth of synonyms only shows a limited tendency to reduce at the margin despite using a range of text processing techniques. If these results were to be repeated in other corpora of student diagrams, and there is no evidence to suggest that this corpus is atypical, it suggests that the problem of matching labels is significant and cannot be easily avoided for the e-assessment of diagrams. The finding implies that a set of better syntax and semantic similarity algorithms would be required to solve the problem.

In order to find the extent to which syntax and semantic algorithms would solve the label matching problem, all the correct labels in the student diagrams were manually categorised according to whether they would require just the syntax algorithms, just the semantic algorithms, or a combination of both for being matched to the label in the model solution. The results for this have been presented in Section 4.4.2 of Chapter 4. In this one experiment I found that just over a fifth of the correct labels were found by using syntax and simple semantic algorithms. If this result were to be repeated in other corpora of student diagrams, and there is no evidence to suggest that this corpus is atypical, it implies the importance of semantic algorithms in matching labels without which e-assessment of diagrams containing labels would be difficult.

In order to solve the diagram label matching problem explained above, a configurable and extensible framework was proposed. A new hybrid syntax matching algorithm was also proposed. This hybrid approach is a combination of existing syntax algorithms. The proposed framework has five stages and has been explained in Chapter 3. The *first* stage disambiguates the labels to produce a set of cleaned labels that are used for the subsequent stages. The *second* stage runs the syntax algorithms on these cleaned labels to produce a matrix for the syntactic similarity index. The *third* stage uses WordNet to produce a list of synonyms in the form of a synonym XML file. The *fourth* stage uses this synonym XML file and re-executes the *second* syntax stage to produce a matrix for the combined similarity matrix. The final *fifth* stage analyses this combined similarity matrix and marks the labels in the students' diagrams as correct if a match can be found in the model diagram and incorrect if a match can not be found in the model diagram.

The proposed framework is configurable, extensible, generic by design and easy to use.

It is not confined to an algorithm, an implementation of an algorithm or a set of parameters. In order to achieve this it uses configuration XML (W3C 08), dynamic loading of classes (Micr 99b; Micr 09) and two design patterns namely the strategy design pattern (Wiki 10g) and the facade design pattern (Wiki 10b). A software prototype implementation of the framework was developed in order to evaluate it.

The first three stages of the proposed framework and hybrid syntax matching algorithm were then evaluated on the same 160 coursework items on which previous experiments to measure the extent of the label matching problem were run. The results showed that the auto correct stage was quite useful by correcting 86% of the total misspelled labels but the abbreviation expansion stage did not perform well as it could only expand one of the four abbreviations. The results also showed that the hybrid approach was better than the three existing syntax algorithms used alone. The results also showed that the WordNet is only marginally effective in finding the synonyms. Overall the results showed that that the framework has been effective but only to limited extent and needs to be further refined for the semantic stage.

5.2 Contributions

The contributions of this Thesis are as follows:

- **Highlighted diagram label matching problem**
  This Thesis has highlighted the importance of label matching in the e-assessment of diagrams by empirically measuring the extent of the diagram label matching problem on 160 coursework at a UK HEI. The results showed that the diagram label matching is a substantial problem and cannot be easily avoided for the e-assessment of diagrams.

- **Proposed and evaluated a framework to match diagram labels**
  This Thesis has proposed a framework to match the labels in a diagram and evaluated the first three stages of the proposed framework on a coursework at UK HEI. The results show that syntax algorithms can be helpful only to a limited extent and better semantic algorithms are needed to match diagram labels for e-assessment.

- **Introduced a hybrid syntax matching algorithm**

  This Thesis has proposed and evaluated a hybrid syntax matching algorithm. The results showed that the hybrid approach was better than the existing syntax algorithms.

- **Provided corpus of coursework for further experiments**

  To carry out the experiments in this Thesis a corpus of student coursework was collected from a UK HEI the details of which has been mentioned in Chapter 4. Considering the time and effort required to collect the coursework for experiments, this corpus coursework can be used by the research community for further experiments. The corpus of coursework has been released at (Jaya 10) in the following three formats.

  - Borland Architect CASE tool project format(Toge 08)
  - XMI Format
  - JPEG PDF Format

- **Prototype Open source implementation of the proposed framework**

  This Thesis provides an open source implementation of the proposed framework. This open source prototype implementation of the framework can be used by the research community for further implementation and evaluation. The Java source code for this prototype implementation is shown in Appendix 7. The Java code, Java documentation javadoc and netbeans project folder of the prototype has also been published at (Jaya 10) so that it is easy to download and reuse. A screen shot of the published website is provided at Section 6.9 of Appendix 6.

5.3 Limitations and future work

Following are limitations of this Thesis and directions for future work to overcome these limitations.

- More experiments with different corpus of coursework

  This Thesis only conducted a single empirical study on a corpus of coursework for second year Computer Science undergraduates at Brunel University. Although this corpus was large scale, realistic and to some extent representative of the UK HEI home students it would be essential to see this work replicated by other researchers using different corpus of coursework and groups of students.

- Extend framework and Combined Hybrid Syntactic algorithm

  The framework proposed in this Thesis does not handle the scenario where same label occurs more than one time in the same student diagram or in the model solution. This is unlikely to be a frequent occurrence. One solution to handle this is to consider the surrounding labels to distinguish between the same labels and use this to find the matching label. The proposed framework should be extended to handle this scenario.

  Also the Combined Hybrid Syntactic algorithm proposed in this Thesis does not handle scenario where a student has used a negation word for example “not” and an antonym to represent a label in the model solution. For example “not valid” to represent “invalid”. One solution is to have a list of antonyms for each word in the model solution and judge the labels as matched if it consists of a negation word and an antonym of word in the model solution label. Applying this rule will judge “not valid” and “invalid” as matched because “not valid” consists of a negation word (not) and an antonym of “invalid”. The algorithm also does not handle the scenario where there is more than one negation word in a label. Although this is unlikely to be a frequent occur but can be an area for future work. This algorithm should be extended to handle this scenario.

- Graphical user interface (GUI) for configuring the framework

  The framework in this Thesis can be used without explicitly configuring it as all the parameters are set to default values but the lecturer may need to configure it so as to optimise for a coursework. This configuration can be done by modifying the configuration XML file which has been which has been shown in the Appendix Section 6.3. However a graphical user interface (GUI) to configure the framework could be useful as it will allow users unfamiliar with XML to configure and use the framework.

  The GUI could have a sliding cursor for the threshold. On moving the cursor, the

FScore, precision and recall from the test data would be displayed. A lecturer could use this feature to set the optimal threshold using the test data. The GUI could also have checkboxes selecting the existing syntax algorithms to be used for calculating the syntactic similarity, and a browse button to embed any other syntax algorithm. It could also have two text boxes for entering the list of special characters, stopwords. These text boxes should be auto-populated with the default values for special characters and stopwords.

- Image Import plugin

  The framework proposed in this Thesis is based on the XML format and takes diagram as input in XMI format (OMG 07b). XMI is a well known standard to express UML diagrams and many open source and commercial tools support it for example ArgoUML (Argo 10) and Borland Together (Toge 08). The open source tool NetBeans (Corp 09) also promises to support it in the near future. However in many cases the students submit the diagram not in XMI format (OMG 07b) but in an image format for example JPEG or GIF. So further work should be carried out to extend the prototype so that it can input the diagrams in image formats for example JPEG, GIF, SVG etc. (Mian 99). A parser to parse the XMI file and fetch various labels has been included in the framework. By design the framework is modular and based on XML, so it can be extended for other types of diagrams by writing a relevant parser middleware component.

- Explore spell check algorithms and abbreviation expanders

  The results of experiments in this Thesis show that the open office spell checker is quite effective as it successfully auto corrects 86% of the incorrectly spelled labels. However it also over corrects the 14% of incorrectly spelled labels. So there is a need for further research to investigate the effectiveness of other spell checkers and auto correct algorithms. Also the spell checker generally suggests multiple words and so one has to decide which one to pick up for replacing the incorrectly spelled label. For this study we selected the first suggested word for the auto correction but it may be useful to see the effect of selecting other words suggested by the spell checker.

A further interesting direction for future work would be to explore if the spell check stage of the framework can be used to detect plagiarism based on the notion that similar spelling mistakes in different coursework indicates plagiarism. Informally we observed that several student solutions contained identical spelling errors and this was not detected by the manual marking process.

- Concatenated words

  As explained in Chapter 3, sometimes students concatenate multiple words in a label for example in the label "UpdateTotal". In the framework proposed in this Thesis the spell checker has been used to separate the concatenated words. The results in Table 4.11 show that the spell checker has been effective in dealing with concatenated words as it could successfully separate words in 45 out of 51 concatenated labels. However as the spell checker could not deal with the remaining 6 out of 51 concatenated labels, an interesting direction for future work would be to explore better ways to deal with concatenated words for example separating the words using the embedded upper case. If upper case proves to be effective then the concatenated words in a label need to be separated before converting the label to lowercase for further processing.

- Calculating the optimal threshold

  The framework proposed in this Thesis is based upon the concept of selecting the maximum similarity index from various algorithms and considering the pairs which have similarity indices greater than the threshold as matched. The results discussed in Section 4.6 of Chapter 4 show that lower values of the threshold result in high recall but low precision, while a higher value of the threshold result in high precision but low recall. These results also show that out of the threshold values 0.5 through 1.0, 0.5 is the optimal threshold for all the four syntax algorithms analysed in this Thesis. Further work can be done to refine the process to determine the optimal threshold. One possible method to determine the optimal threshold would be to use past training data which, although tedious, might give more reliable results. The advantage of this approach is that it can give more reliable values for threshold and hence improve overall reliability of the e-assessment process. The disadvantage is that it can be difficult to collect training data but if the same assessment is used over a period of time then the difficulty may be overcome by the economics of scale.

- Level of detail in the model solution

  One of the points to consider while automatically marking diagrams is the level of detail in the model solution. Students may produce answers in more detail than required assuming it can only act as basis for more marks. On the other hand lecturers may produce a less detailed model solution and supplement the model solution with human judgement while marking. There is a trade-off between the level of detail in the model solution and the accuracy of the e-assessment system. A more detailed model solution will penalise students who have provided coursework with just enough detail but reward students who have produced more detailed solution. So it is important to consider the level of detail in the model solution for e-assessment. One solution is to have multiple model solutions, one with just enough detail and others with more detail. The student diagram can be compared with all the different model solutions and maximum value out of the different marks awarded by different model solutions can be awarded to the student as final marks. However, this approach would entail more work on the part of the lecturer.

- Handling differing level of decomposition

  The manual categorisation of labels in Table 4.7 shows that the syntax and semantic relationship “Synonym” which this Thesis deals with, accounts only for 29.7% of correct labels. The categories for example “Differing level of decomposition” which account for the remaining 70.3% of the correct labels are not handled by this Thesis and and is an area for future work. It will be useful to explore the extent to which ontologies can deal with the “Differing level of decomposition” category. The domain specific ontologies can be generated automatically from the text of the question and related domain specific documents.

- Reduce human intervention during semantic stage

  The framework proposed in this Thesis requires manual intervention by the lecturer during the semantic stage to select the correct senses from WordNet for each word in the model solution. This manual intervention is not a major limitation because it needs to be done only for the labels in the model solution which are generally not too many; additionally the manual intervention reduces if same question is used multiple times. But reducing the amount of manual intervention can be helpful and is a topic for future work. Two possible directions for this are efficient human computer interfaces and word sense disambiguation techniques. Also WordNet is generic and not specific to any particular domain. So future work should explore the use of domain specific sources of senses and synonyms.

  Efficient human computer interfaces such as navigable tree based expandable check box nodes would help the lecturer to choose the senses of a word quickly and hence reduce the time spent on manual intervention. Such a tree based HCI is used by the Assess By Computer system to allow the lecturer to review and update the matched labels (Jone 05; Tsel 05). So a direction for future work is to develop and evaluate effective human computer interfaces to help lecturers choose the word senses.

  As discussed in Section 3.4.3 of Chapter 3, the framework proposed in this Thesis uses manual intervention by the lecturer to select the senses over using automatic word sense disambiguation techniques because the word sense disambiguation field is an open research problem and difficult to apply on labels as they consist of just a few words and not complete sentences. But the coursework question text may be of help in finding the sense in which the word has been used. This is supported by the results discussed in Section 4.7 of Chapter 4 which show that 6 out of 17 synonyms used by the students were present in the text of the question. So a direction for future work is to explore the use of the coursework question text in finding the sense in which word has been used.

- Explore factors influencing student's choice of labels and semantic resources

This Thesis has explored the problem of diversity of diagram labels used by students. In the semantic analysis, it was explored if the specification of the case study affected the choice of labels. Table 4.18 shows that 6 out of 17 synonyms used by the students were found in the text of the question which suggests that the text of question influences the choice of labels by students. Although such work is out of scope for this Thesis but it opens up potential area for future work. Research should be carried out to study the factors that influence the student's choice of a label. Apart from the text of the question, some of the other factors that can be explored are the personal factors of the student like behavioural traits, background and level of knowledge of the subject. The text of the question is particularly useful to explore because the lecturer has control over it unlike other factors over which the lecturer has no control.

Also as the results discussed in Chapter 4 show that WordNet is only marginally effective, there is a need to explore other semantic resources for example ConceptNet (Liu 04) and CYC (Lena 95).

- Extend e-learning standards

  Extend OKI (Tech 07) standards to incorporate the Application Programming Interface (API) for e-assessment tools for diagram marking. Extend IMS QTI specification to incorporate fields for automatic assessment of diagrams. The study by Jayal (Jaya 07b) shows that there is little evidence of widespread adoption of the e-learning standards.

- Develop new standards for diagram coursework and marking schemes

  In order to evaluate the e-assessment tools, collecting and human marking of the coursework is required. The real coursework can only be collected after exams which don't happen throughout the year. The human marking scheme is also difficult to capture as many times it involves human judgement and is difficult to formally state the marking rule. All this takes considerable effort and time. So storing the coursework and marking schemes in a generic way so that they can be retrieved by others for reuse would be helpful for the research community at large. Pete Thomas of Open University has proposed to create a generic e-learning standard for storing the diagram coursework and capturing the marking schemes[1].

[1] There is no published work for this idea but only an informal email exchange with Pete Thomas of Open University.

Bibliography

[Ala 05] K. Ala-Mutka. "A Survey of Automated Assessment Approaches for Programming Assignments". *Computer Science Education*, Vol. 15, No. 2, pp. 83–102, 2005.

[Argo 10] ArgoUML. "ArgoUML User Manual". Tech. Rep., ArgoUML, Jan 2010.

[Atki 04] K. Atkinson. "GNU Aspell Spell Checker". `http://aspell.net/`, Accessed on 20 March 2010, 2004.

[Bane 02] S. Banerjee and T. Pedersen. "An adapted Lesk algorithm for word sense disambiguation using WordNet". *Computational Linguistics and Intelligent Text Processing*, pp. 117–171, 2002.

[Batm 06] F. Batmaz and C. Hinde. "A Diagram Drawing Tool for Semi-Automatic Assessment of Conceptual Database Diagrams". *Proceedings of the 10th CAA International Computer Assisted Assessment Conference*, Vol. 4, pp. 71–82, 2006.

[Beev 02] C. Beevers. "The SCHOLAR Programme in Scottish education". *Proceedings of International Conference on Computers in Education*, pp. 490–491, 2002.

[Blac 10] Blackboard. "Blackboard Home". `http://www.blackboard.com/`, Accessed on 20 March 2010, 2010.

[Brow 08] T. Browne, R. Hewitt, M. Jenkins, and R. Walker. "Technology Enhanced Learning Survey". *UCISA, available online at http://www. ucisa. ac. uk/publications/tel_survey. aspx*, 2008.

[Brow 97a] G. Brown, J. Bull, and M. Pendlebury. *Assessing student learning in higher education.* Routledge, 1997.

[Brow 97b] G. Brown, J. Bull, and M. Pendlebury. *Assessing Student Learning in Higher Education.* Routledge, 1997.

[Buda 01] A. Budanitsky and G. Hirst. "Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures". In: *Workshop on WordNet and Other Lexical Resources*, 2001.

[CETI 08] CETIS. "CETIS-Documents and resources about the UK LOM core". `http://www.cetis.ac.uk/profiles/uklomcore/`, Accessed on 20 March 2010, 2008.

[Chap 06] S. Chapman. *String Similarity Metrics For Information Integration.* University of Sheffield, 2006.

[Chin 03] P. Chin. "Virtual Learning Environments". *Learning and Teaching Support Network (LTSN) Physical Sciences Centre*, 2003.

[Chri 06] P. Christen. "A comparison of personal name matching: Techniques and practical issues". In: *Sixth IEEE International Conference on Data Mining Workshops, ICDM Workshops 2006*, pp. 290–294, 2006.

[Clar 02] J. Clark. "A product review of WebCT". *The Internet and Higher Education*, Vol. 5, No. 1, pp. 79–82, 2002.

[Cohe 03] W. Cohen, P. Ravikumar, and S. Fienberg. "A comparison of string distance metrics for name-matching tasks". In: *Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web (IIWeb-03)*, pp. 9–10, Citeseer, 2003.

[Cole 05] J. Cole. *Using moodle.* O'Reilly, 2005.

[Cons 10] I. G. L. Consortium. "IMS Inteoperability Project Groups". `http://www.imsglobal.org/interoperabilitygroups.html`, Accessed on 20 March 2010, 2010.

[Corl 05] C. Corley and R. Mihalcea. "Measuring the Semantic Similarity of Texts". *Empirical Modeling of Semantic Equivalence and Entailment*, Vol. 100, p. 13, 2005.

[Corp 01] O. Corporation. "OpenOffice Lingucomponent Thesaurus Development". `http://lingucomponent.openoffice.org/thesaurus.html`, Accessed on 20 March 2010, 2001.

[Corp 09] O. Corporation. "NetBeans UML". `http://netbeans.org/projects/uml/`, Accessed on 20 March 2010, 2009.

[Corp 10a] O. Corporation. "Open Office 3.2". `http://www.openoffice.org/`, Accessed on 20 March 2010, 2010.

[Corp 10b] O. Corporation. "Packaging Programs in JAR Files". `http://java.sun.com/docs/books/tutorial/deployment/jar/`, Accessed on March 2010, 2010.

[Dict 10] Dictionary.com. "Dictionary reference". `http://dictionary.reference.com/`, Accessed on 20 March 2010, 2010.

[Doyl nd] D. Doyle. "English Stopwords". n.d. http://www.ranks.nl/tools/stopwords.html, Accessed on 20 March 2010.

[Fell 98] C. Fellbaum *et al.* *WordNet: An electronic lexical database.* MIT press Cambridge, MA, 1998.

[Feng 06] M. Feng, N. Heffernan, and K. Koedinger. "Addressing the testing challenge with a web-based e-assessment system that tutors as it assesses". *Proceedings of the 15th international Conference on World Wide Web*, pp. 307–316, 2006.

[Fiel 02] R. Fielding and R. Taylor. "Principled design of the modern Web architecture". *ACM Transactions on Internet Technology (TOIT)*, Vol. 2, No. 2, pp. 115–150, 2002.

[Fong 05] S. Fong. "Wordnet wnconnect". 2005. http://dingo.sbs.arizona.edu/ sandiway/wnconnect/, Accessed on 20 March 2010.

[Foun 09] T. A. S. Foundation. "Apache Codec Project". http://commons.apache.org/codec/userguide.html, Accessed on 20 March 2010, 2009.

[Frak 03] W. Frakes and C. Fox. "Strength and similarity of affix removal stemming algorithms". In: *ACM SIGIR Forum*, pp. 26–30, ACM New York, NY, USA, 2003.

[Fran 03] D. Frankel. *Model driven architecture*. Wiley New York, 2003.

[Fran 09] F. Frandsen. "Java API For Hunspell Spell Checker". http://dion.swamp.dk/hunspell.html, Accessed on 20 March 2010, 2009.

[Gamm 95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-wesley Reading, MA, 1995.

[Harm 91] D. Harman. "How effective is suffixing?". *Journal of the American Society for Information Science*, Vol. 42, No. 1, pp. 7–15, 1991.

[Hart 94] R. Hart. "Improved Algorithms for Identifying Spelling and Word Order Errors in Student Responses.". Tech. Rep., Language Learning Laboratory, University of Illinois at Urbana-Champaign, G70 Foreign Languages Building, 707 S. Mathews St., Urbana, IL 61801., 1994.

[Higg 02a] C. Higgins, P. Symeonidis, and A. Tsintsifas. "Diagram based CBA using DATsys and CourseMaster". *Proceedings of International Conference on Computers in Education*, pp. 167–172, 2002.

[Higg 02b] C. Higgins, P. Symeonidis, and A. Tsintsifas. "The marking system for CourseMaster". *Proceedings of the 7th annual Conference on Innovation and Technology in Computer Science Education*, pp. 46–50, 2002.

[Higg 06] C. Higgins and B. Bligh. "Formative computer based assessment in diagram based domains". *Proceedings of the 11th annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pp. 98–102, 2006.

[Hill 06] M. Hillenmeyer. "Dice coefficient". http://www.stanford.edu/~maureenh/quals/html/ml/node69.html, Accessed on 20 March 2010, 2006.

[Hirs 98] G. Hirst and D. St-Onge. "Lexical chains as representations of context for the detection and correction of malapropisms". *WordNet: An electronic lexical database*, Vol. 305, p. 332, 1998.

[Hodg 05] W. Hodgins *et al.* "IEEE Learning Object Metadata". http://ltsc.ieee.org/wg12/files/IEEE_1484_12_03_d8_submitted.pdf, Accessed on 20 March 2010, 2005.

[Hogg 98] G. Hoggarth and M. Lockyer. "An automated student diagram assessment system". *Proceedings of the 6th annual Conference on the Teaching of Computing and the 3rd annual Conference on Integrating Technology into Computer Science Education: Changing the Delivery of Computer Science Education*, pp. 122–124, 1998.

[Jaya 07a] A. Jayal, M. Cartwright, and M. Shepperd. "Premark: A System Designed To Organising Course Work For Assessment". In: *5th International Conference on E-Governance, Hyderabad, India, Dec 28-30, 2007*, December 2007.

[Jaya 07b] A. Jayal and M. Shepperd. "An evaluation of e-learning standards". In: *5th International Conference on E-Governance, Hyderabad, India, Dec 28-30, 2007*, December 2007.

[Jaya 09a] A. Jayal and M. Shepperd. "An improved method for label matching in e-assessment of diagrams". *ITALICS, Innovation in Teaching And Learning in Information and Computer Sciences*, Vol. 8, No. 1, 2009.

[Jaya 09b] A. Jayal and M. Shepperd. "The problem of labels in e-assessment of diagrams". *Journal on Educational Resources in Computing (JERIC)*, Vol. 8, No. 4, p. 12, 2009.

[Jaya 10] A. Jayal. "Ambikesh PhD Thesis Output". urlhttp://sites.google.com/site/ambi1999/research, Accessed on March 20, 2010, 2010.

[Jian 97] J. Jiang and D. Conrath. “Semantic similarity based on corpus statistics and lexical taxonomy”. *Arxiv preprint cmp-lg/9709008*, 1997.

[John 88] R. Johnson and B. Foote. “Designing reusable classes”. *Journal of Object-Oriented Programming*, Vol. 1, No. 2, pp. 22–35, 1988.

[Jone 05] C. Jones. “Teachers need help too: aiding the marking process through a Human-Computer Collaborative approach”. In: *8th Human Centred Technology Postgraduate Workshop*, 2005.

[Kere 05] M. Kerejeta, M. Larranaga, U. Rueda, A. Arruarte, and J. Elorriaga. “TOKA: A Computer Assisted Assessment Tool Integrated in a Real Use Context”. *ICALT 2005. Fifth IEEE International Conference on Advanced Learning Technologies*, pp. 848–852, 2005.

[Khed 05] K. Khedo. “Computer-assisted assessment system at the University of Mauritius”. In: *IEEE 3rd International Conference on Computational Cybernetics, ICCC 2005*, pp. 187–193, 2005.

[Leac 98] C. Leacock and M. Chodorow. “Combining local context and WordNet similarity for word sense identification”. *WordNet: An electronic lexical database*, Vol. 49, No. 2, pp. 265–283, 1998.

[Lear 04] A. D. Learning. “SCORM - Shareable Content Object Reference Model”. `http://www.scormsoft.com/scorm`, Accessed on 20 March 2010, 2004.

[Lena 95] D. Lenat. “CYC: A large-scale investment in knowledge infrastructure”. *Communications of the ACM*, Vol. 38, No. 11, pp. 33–38, 1995.

[Lesk 86] M. Lesk. “Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone”. In: *Proceedings of the 5th annual international conference on Systems documentation*, pp. 24–26, ACM, 1986.

[Lin 98] D. Lin. "An information-theoretic definition of similarity". In: *Proceedings of the International Conference on Machine Learning*, pp. 296–304, 1998.

[Liu 04] H. Liu and P. Singh. "ConceptNeta practical commonsense reasoning tool-kit". *BT Technology Journal*, Vol. 22, No. 4, pp. 211–226, 2004.

[LLC 10a] S. LLC. "Abbreviations API". `http://www.abbreviations.com/abbr_api.asp`, Accessed on 20 March 2010, 2010.

[LLC 10b] S. LLC. "List of Abbreviations". `http://www.abbreviations.com/`, Accessed on 20 March 2010, 2010.

[Lovi 68] J. Lovins. "Development of a Stemming Algorithm.". Tech. Rep., Massachusetts Inst of Tech Cambridge Electronic Systems Lab, 1968.

[McGe 05] M. McGee Wood, J. Sargeant, and C. Jones. "What students really say". *Proceedings of the 9th CAA International Computer Assisted Assessment Conference*, 2005.

[Memi 09] Memidex. "Memidex Dictionary". `http://www.memidex.com/`, Accessed on 20 March 2010, 2009.

[Mian 99] J. Miano. *Compressed Image File Formats: Jpeg, Png, Gif, Xbm, Bmp.* Addison-Wesley, 1999.

[Micr 04] S. Microsystems. "Java class path". `http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/classpath.html`, Accessed on 20 March 2010, 2004.

[Micr 09] S. Microsystems. "Class Java Platform SE 6". `http://java.sun.com/javase/6/docs/api/java/lang/Class.html`, Accessed on 20 March 2010, 2009.

[Micr 99a] S. Microsystems. "Java class file format". `http://java.sun.com/docs/books/jvms/second_edition/html/ClassFile.doc.html`, Accessed on 20 March 2010, 1999.

[Micr 99b] S. Microsystems. "Java Security Architecture". http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/security-spec.doc5.html, Accessed on 20 March 2010, 1999.

[Mill 95a] G. Miller. "WNGLOSS(7WN) manual page". http://wordnet.princeton.edu/man/wngloss.7WN.html, Accessed on 20 March 2010, 1995.

[Mill 95b] G. Miller and C. Fellbaum. "Wordnet Search". http://wordnetweb.princeton.edu/perl/webwn, Accessed on 20 March 2010, 1995.

[Nava 01] G. Navarro. "A guided tour to approximate string matching". *ACM computing surveys (CSUR)*, Vol. 33, No. 1, p. 88, 2001.

[Neil 00] C. Neill. "Algorithm Implementations Paice Husk stemming algorithm". http://www.comp.lancs.ac.uk/computing/research/stemming/Links/implementations.htm, Accessed on 20 March 2010, 2000.

[Neme 10] N. Nemeth. "Hunspell open source spell checking". http://hunspell.sourceforge.net/, Accessed on 20 March 2010, 2010.

[OMG 07a] OMG. "UML Version 2.1.2". Tech. Rep., Object Management Group, November 2007.

[OMG 07b] OMG. "XMI 2.1.1". Tech. Rep., Object Management Group, December 2007.

[Paic 90] C. Paice. "Another stemmer". In: *ACM SIGIR Forum*, pp. 56–61, ACM New York, NY, USA, 1990.

[Patw 03] S. Patwardhan, S. Banerjee, and T. Pedersen. "Using measures of semantic relatedness for word sense disambiguation". *Computational Linguistics and Intelligent Text Processing*, pp. 241–257, 2003.

[Pede 04] T. Pedersen, S. Patwardhan, and J. Michelizzi. "Wordnet:: similarity-measuring the relatedness of concepts". In: *Proceedings of the National Conference on Artificial Intelligence*, pp. 1024–1025, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2004.

[Pede 08] T. Pedersen. “Semantic Similarity Measures”. http://marimba.d.umn.edu/similarity/measures.html, Accessed on 20 March 2010, 2008.

[Pede 09a] T. Pedersen and M. J. “WordNet Similarity”. http://marimba.d.umn.edu/cgi-bin/similarity/similarity.cgi, Accessed on 20 March 2010, 2009.

[Pede 09b] T. Pedersen and V. Kolhatkar. “WordNet:: SenseRelate:: AllWords: a broad coverage word sense tagger that maximizes semantic relatedness”. In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Demonstration Session*, pp. 17–20, Association for Computational Linguistics, 2009.

[Pere 05] D. Perez, A. Gliozzo, C. Strapparava, E. Alfonseca, P. Rodriguez, and B. Magnini. “Automatic Assessment of Student free-text Answers underpinned by the Combination of a Bleu-inspired algorithm and Latent Semantic Analysis”. *In Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference*, pp. 358–362, 2005.

[Poll 02] M. Pollock. “Introduction of CAA into a Mathematics Course for Technology Students to Address a Change in Curriculum Requirements”. *International Journal of Technology and Design Education*, Vol. 12, No. 3, pp. 249–270, 2002.

[Port 06] M. Porter. “An algorithm for suffix stripping”. *Program: Electronic Library and Information Systems*, Vol. 40, No. 3, pp. 211–218, 2006.

[Resn 95] P. Resnik. “Using information content to evaluate semantic similarity in a taxonomy”. In: *International Joint Conference on Artificial Intelligence*, pp. 448–453, 1995.

[Russ 18] C. Russell. “Soundex Algorithm”. http://www.pat2pdf.org/pat2pdf/foo.pl?number=1261167, Accessed on 20 March 2010, 1918.

[Sala 09] M. Salahli and T. Canakkale. “AN APPROACH FOR MEASURING SEMAN-

TIC RELATEDNESS BETWEEN WORDS VIA RELATED TERMS". *Mathematical and Computational Applications*, Vol. 14, No. 1, pp. 55–63, 2009.

[Sanf 98] A. Sanfilippo, N. Calzolari, S. Ananiadou, R. Gaizauskas, P. Saint-Dizier, and P. Vossen. "EAGLES Preliminary Recommendations on Semantic Encoding, Word Sense Disambiguation". *The EAGLES Lexicón Interest Group*, 1998.

[Seco 04a] N. Seco. "Java WordNet Similarity Library JAVASIMLIB". `http://eden.dei.uc.pt/~nseco/javasimlib.tar.gz`, Accessed on 20 March 2010, 2004.

[Seco 04b] N. Seco, T. Veale, and J. Hayes. "An Intrinsic Information Content Metric for Semantic Similarity in WordNet". *Proc. of ECAI*, Vol. 4, pp. 1089–1090, 2004.

[Sedg 03] R. Sedgewick. *Algorithms in Java.* Addison-Wesley Professional, 2003.

[Smir 08] I. Smirnov. "Overview of Stemming Algorithms". Tech. Rep., DePaul University, 2008.

[Smit 04] N. Smith, P. Thomas, and K. Waugh. "Interpreting Imprecise Diagrams". *Proceedings of the Third International Conference in the Theory and Application of Diagrams. March*, pp. 22–24, 2004.

[Sura 02] P. Suraweera and A. Mitrovic. "KERMIT: a Constraint-based Tutor for Database Modeling". *Proc. ITS*, pp. 377–387, 2002.

[Take 90] H. Takeda, P. Veerkamp, and H. Yoshikawa. "Modeling design process". *AI magazine*, Vol. 11, No. 4, p. 37, 1990.

[Tech 07] M. I. of Technology. "Open Knowledge Initiative". `http://www.okiproject.org/`, Accessed on 20 March 2010, 2007.

[Tera 04] A. Terada, T. Tokunaga, and H. Tanaka. "Automatic expansion of abbreviations by using context and character information". *Information Processing and Management*, Vol. 40, No. 1, pp. 31–45, 2004.

[Thom 04] P. Thomas. "Drawing diagrams in an online examination". *Proceedings of the 8th CAA International Computer Assisted Assessment Conference*, 2004.

[Thom 07a] P. Thomas, N. Smith, and K. Waugh. "Computer assisted assessment of diagrams". *Proceedings of the 12th annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pp. 68–72, 2007.

[Thom 07b] P. Thomas, K. Waugh, and N. Smith. "Learning and Automatically Assessing Graph based Diagrams". *Proceedings of the 7th ALT Association for Learning Technology Conference*, 2007.

[Thom 09] P. Thomas, N. Smith, and K. Waugh. "The Role Of Labels In The Automatic Assessment Of Graph-Based Diagrams". *23rd ICDE World Conference on Open Learning and Distance Education*, 2009.

[Toge 08] B. Together. "Borland Together". Tech. Rep., Borland Together, April 2008.

[Tsel 05] C. Tselonis and J. Sargeant. "Diagram matching for human computer collaborative assessment". *Proceedings of the 9th CAA International Computer Assisted Assessment Conference*, 2005.

[Tsel 07] C. Tselonis and J. Sargeant. "Domain-specific formative feedback through domain-independent diagram matching". *Proceedings of the 11th CAA International Computer Assisted Assessment Conference*, 2007.

[Ukko 92] E. Ukkonen. "Approximate string-matching with q-grams and maximal matches* 1". *Theoretical Computer Science*, Vol. 92, No. 1, pp. 191–211, 1992.

[Univ 96] I. University. "Oxford English Dictionary List of Abbreviations". `http://www.indiana.edu/~letrs/help-services/QuickGuides/oed-abbr.html`, Accessed on 20 March 2010, 1996.

[Vais 04] V. Vaishnavi and W. Kuechler. "Design Research in Information Systems". `http://desrist.org/design-research-in-information-systems/`, Accessed on 20 March 2010, 2004.

[Vale 03] S. Valenti, F. Neri, and A. Cucchiarelli. "An overview of current research on automated essay grading". *Journal of Information Technology Education*, Vol. 2, pp. 319–330, 2003.

[Veks 07] D. V. Veksler. "Measures of Semantic Relatedness". 2007. http://cwl-projects.cogsci.rpi.edu/msr/, Accessed on 20 March 2010.

[W3C 08] W3C. "Extensible Markup Language (XML) 1.0". Tech. Rep., World Wide Web Consortium (W3C), November 2008.

[Wagn 74] R. Wagner and M. Fischer. "The String-to-String Correction Problem". *Journal of the ACM (JACM)*, Vol. 21, No. 1, pp. 168–173, 1974.

[Went 00] T. Wentling, C. Waight, J. Gallaher, J. La Fleur, C. Wang, and A. Kanfer. "E-learning-A review of literature". *Knowledge and Learning Systems Group, University of Illinois at Urbana-Champaign learning. ncsa. uiuc. edu/papers/e-learnlit. pdf*, 2000.

[Whit nd] S. White. "How to Strike a Match". n.d. http://www.catalysoft.com/articles/StrikeAMatch.html, Accessed on 20 March 2010.

[Wiki 10a] Wikipedia. "F1 score". http://en.wikipedia.org/wiki/F1_score, Accessed on 20 March 2010, 2010.

[Wiki 10b] Wikipedia. "Facade design pattern". http://en.wikipedia.org/wiki/Facade_pattern, Accessed on 20 March 2010, 2010.

[Wiki 10c] Wikipedia. "Precision and Recall". http://en.wikipedia.org/wiki/Precision_and_recall, Accessed on 20 March 2010, 2010.

[Wiki 10d] Wikipedia. "Soundex Algorithm". http://en.wikipedia.org/wiki/Soundex, Accessed on 20 March 2010, 2010.

[Wiki 10e] Wikipedia. "Stemming". http://en.wikipedia.org/wiki/Stemming, Accessed on 20 March 2010, 2010.

[Wiki 10f] Wikipedia. "Stop words". http://en.wikipedia.org/wiki/Stop_words, 2010.

[Wiki 10g] Wikipedia. "Strategy design pattern". http://en.wikipedia.org/wiki/Strategy_pattern, Accessed on 20 March 2010, 2010.

[Wiki 10h] Wikipedia. “WordNet”. `http://en.wikipedia.org/wiki/WordNet`, Accessed on 20 March 2010, 2010.

[Wu 94] Z. Wu and M. Palmer. “Verbs semantics and lexical selection”. In: *Proceedings of the 32nd Annual Meeting on Association for Computational Linguistics*, pp. 133–138, Association for Computational Linguistics Morristown, NJ, USA, 1994.

Chapter 6

Appendix A

6.1 Technologies used for developing software prototype of the framework

The following Open Source technologies have been used in the partial implementation of the framework.

- Java
- Netbeans
- Linux
- Hunspell spell checker (Neme 10) for the stage S1.3 and S1.4
- Open Office (Corp 10a) for the stage S1.3 and S1.4
- Paice Husk Stemmer (Paic 90; Neil 00) for the stage S1.6
- Implementation of syntax matching algorithms (Chap 06) for stage S2

List of Stopwords	List1	a, an, the, this, that
	List2	to, of, for, from, and, be, or, on, if, in, with, by, as, but, at
	List3	i, he, she, it, we, you, they
	List4	is, am, are, do, did, was, were, has, have, been

Table 6.1: List Of Stopwords

6.2 List of parameters used during pre-processing stage

6.2.1 List of special characters

The special character ’ (single quote) was removed while the following list of special characters were replaced by single space.

```
!,",,$,%,^,&,*,(,),-,_,+,=,{,},[,],~,#,;,:,@,<,>,.,?,/,‘,\\,|
```

6.2.2 List of stopwords

Table 6.1 lists the stopwords used during the pre-processing stage.

6.3 Configuration XML File

The following is an example of the configuration xml file which is used to configure various algorithms and parameters for the software prototype of the framework proposed in this Thesis.

```
<toolconfiguration>
<casesensitive value="true"> </casesensitive>
<trimming value="true"> </trimming>
<specialcharacter>
<!--Path of xml file containing the special characters-->
<path>
</path>
```

```
</specialcharacter>
<stopword>
<!--Path of xml file containing the stopwords-->
<path>
</path>
</stopword>
<abbreviation>
<!--Path of xml file containing the abbreviations-->
<path>
</path>
</abbreviation>
<spellchecker>
<!--Details of the spell check algorithm-->
<algorithm name="">
<concreteclass></concreteclass>
</algorithm>
</spellchecker>
<stemming>
<!--Details of the stemming algorithm-->
<algorithm name="">
<concreteclass></concreteclass>
</algorithm>
</stemming>
<syntaxstage>
<algorithms>
<!--Details of the syntax matching algorithms-->
<!--Note that user can mention more than one syntax algorithm-->
<algorithm name="ExactMatchSyntaxAlgorithm" id="ExactMatchSyntaxAlgorithm"
active="true" weight="0.1">
<concreteclass>ExactMatchSyntaxAlgorithm.java</concreteclass>
```

```
</algorithm>
<algorithm name="EditDistanceSyntaxAlgorithm" active="true">
<concreteclass>EditDistanceSyntaxAlgorithm.java</concreteclass>
</algorithm>
<algorithm name="QGramDistanceSyntaxAlgorithm" active="true">
<concreteclass>QGramDistanceSyntaxAlgorithm.java</concreteclass>
</algorithm>
<algorithm name="SimonWhiteAlgorithm" active="true">
<concreteclass>SimonWhiteAlgorithm.java</concreteclass>
</algorithm>
<algorithm name="SoundexSyntaxAlgorithm" active="true">
<concreteclass>SoundexSyntaxAlgorithm.java</concreteclass>
</algorithm>
<algorithms>
<searchalgorithm>
<!--Details of the search syntax matching algorithms-->
<!--This algorithm will take input by running the various syntax algorithms
and then apply some search algorithm to return a final value for the syntax
similarity index value-->
<!--Read about search algorithms
at http://en.wikipedia.org/wiki/Search_algorithm-->
<!--In computer science, a search algorithm, broadly speaking, is an algorithm
that takes a problem as input and returns a solution to the problem,
usually after evaluating a number of possible solutions.-->
<!--Although there can be many entries for algorithms in the searchalgorithm
but only one can have value active=true. This search algorithm with value of
active=true will be used for searching the search space.-->
<algorithm name="MaxSearchAlgorithm" id="MaxSearchAlgorithm" active="true">
<!--This search algorithm "MaxSearchAlgorithm" searches all the possible
values of the syntax similarity and returns the maximum value. This is based
```

```
on the argument that if any one of the various syntax algorithms outputs that
two labels are similar, then they are indeed similar.-->
<concreteclass>MaxSearchAlgorithm.java</concreteclass>
</searchalgorithm>
</algorithm>
<!--Value for threshold-->
<thresholdvalue> 0.6 </thresholdvalue>
</syntaxstage>
<semantic>
<!--Path of xml file containing the semantic synonyms generated by the
wordnet interface-->
<path>
</path>
</semantic>
<!--Final analysis stage-->
<analysis>
<thresholdvalue> 0.6 </thresholdvalue>
<analysis>
</toolconfiguration>
```

6.4 Document Type Definition (DTD) of synonym XML file

The following is the DTD for the synonym.xml file proposed in this Thesis.

```
<?xml encoding="UTF-8"?>
<!ELEMENT synonymset(description?, word+) >
<!ATTLIST synonymset name CDATA #IMPLIED>
<!ELEMENT description (#PCDATA)>
<!ELEMENT word(synset+)>
<!ATTLIST word value CDATA #REQUIRED
description CDATA #IMPLIED
```

```
>
<!ELEMENT synset(sense,synonymn+,partofspeech,examplesentence+)>
<!ATTLIST synset wordnetdatabseid CDATA #REQUIRED
selectedbylecturer(yes|no) #REQUIRED
description CDATA #IMPLIED
>
<!ELEMENT sense EMPTY>
<!ATTLIST sense value CDATA #REQUIRED >
<!ELEMENT synonymn EMPTY>
<!ATTLIST synonymn value CDATA #REQUIRED >
<!ELEMENT partofspeech EMPTY>
<!ATTLIST partofspeech value (noun|verb|adjective|adverb|other) #REQUIRED >
<!ELEMENT examplesentence EMPTY>
<!ATTLIST examplesentence value CDATA #REQUIRED >
```

6.5 Case Study: Problem Specification and Model Solution

The following case study was used for experiments carried out in this Thesis. **The Cockle Card System:** Chipolata Buses of Marlin on Sea plan to invest in a new bus card system. In addition to a travel card (monthly, weekly, daily), there is a pre-pay card (pay-as-you-go), where customers can purchase credit in advance, and a concessions card, allowing free off peak travel for certain groups of people. Each bus is to be fitted with a card reader which will read the card, update the amount of credit (for pre-pay) or check it is valid (travel cards or concession cards). Different fares are charged for peak and off peak services. If the card is not valid for some reason (e.g. out of date, no credit or cant be used at peak times) the reader should give an audible warning to prompt the driver to read the display and take appropriate action. The reader should also give a valid 'beep' so that the driver and passenger know that the card has been read. The pre-pay card needs to be debited each time it is used. However, there is a daily cap so that it never exceeds the amount that would be charged for a daily travel card. There is a flat fare for each journey, but peak journeys (before 9.30 am) cost

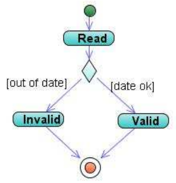


Figure 6.1: Part of Question

more than off peak journeys. The amount charged is displayed. Current costs:

1. Daily travel card £6
2. Weekly travel card £35
3. Monthly travel card £120
4. Single peak journey £2
5. Single off peak journey £1

The following activity diagram [refer to Figure 6.1] only partially models the requirements in the case study. Complete it. The Model Solution, as devised by the lecturer who marked the coursework, is presented in Figure 6.2.

6.5.1 Syntax algorithm result table

6.6 Semantic data

Table 6.10 shows the words in the student diagram that are judged by human marker to be synonymous with words in the model solution.

6.7 Sample code to add an algorithm

As explained in Section 3.5, the actual code to use the porter stemming algorithm in the framework is as follows. The following code shows the way in which the framework reads

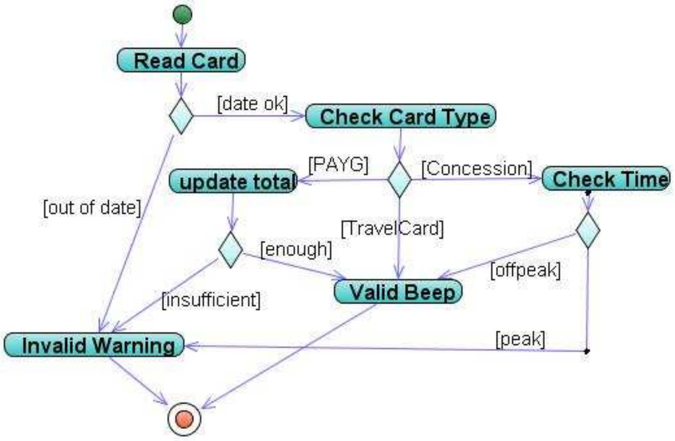


Figure 6.2: Model Answer

Algo	0.5	0.6	0.7	0.8	0.9	1.0
A	141	38	32	32	32	31
B	68	45	26	17	4	4
C	55	30	10	2	1	0
D	41	21	10	2	0	0
B ∩ C ∩ D	34	14	8	1	0	0
B ∪ C ∪ D	83	52	27	18	5	4
A ∩ B ∩ C ∩ D	34	13	8	1	0	0
A ∪ B ∪ C ∪ D	143	57	38	34	32	31
A - (B ∪ C ∪ D)	60	5	11	16	27	27
(B ∪ C ∪ D) - A	2	19	6	2	0	0
A - (B ∩ C ∩ D)	107	25	24	31	32	31
(B ∩ C ∩ D) - A	0	1	0	0	0	0

Table 6.2: True Positives For All Threshold

Algo	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**
A		103	6	0	0	1
B		23	19	9	13	0
C		25	20	8	1	1
D		20	11	8	2	0
B ∩ **C** ∩ **D**		20	6	7	1	0
B ∪ **C** ∪ **D**		31	25	9	13	1
A ∩ **B** ∩ **C** ∩ **D**		21	5	7	1	0
A ∪ **B** ∪ **C** ∪ **D**		86	19	4	2	1
A - (B ∪ **C** ∪ **D)**		55	-6	-5	-11	0
(B ∪ **C** ∪ **D) - A**		-17	13	4	2	0
A - (B ∩ **C** ∩ **D)**		82	1	-7	-1	1
(B ∩ **C** ∩ **D) - A**		-1	1	0	0	0

Table 6.3: Rate of Decrease of True Positives For All Threshold

Algo	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**
A	242	48	32	32	27	19
B	127	57	19	7	0	0
C	120	53	12	3	0	0
D	51	14	5	0	0	0
B ∩ **C** ∩ **D**	47	11	4	0	0	0
B ∪ **C** ∪ **D**	164	85	26	8	0	0
A ∩ **B** ∩ **C** ∩ **D**	47	9	2	0	0	0
A ∪ **B** ∪ **C** ∪ **D**	272	107	46	35	27	19
A - (B ∪ **C** ∪ **D)**	108	22	20	27	27	19
(B ∪ **C** ∪ **D) - A**	30	59	14	3	0	0
A - (B ∩ **C** ∩ **D)**	195	39	30	32	27	19
(B ∩ **C** ∩ **D) - A**	0	2	2	0	0	0

Table 6.4: False Positives For All Threshold

Algo	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**
A		194	16	0	5	8
B		70	38	12	7	0
C		67	41	9	3	0
D		37	9	5	0	0
$\mathbf{B} \cap \mathbf{C} \cap \mathbf{D}$		36	7	4	0	0
$\mathbf{B} \cup \mathbf{C} \cup \mathbf{D}$		79	59	18	8	0
$\mathbf{A} \cap \mathbf{B} \cap \mathbf{C} \cap \mathbf{D}$		38	7	2	0	0
$\mathbf{A} \cup \mathbf{B} \cup \mathbf{C} \cup \mathbf{D}$		165	61	11	8	8
$\mathbf{A}$ **-** $(\mathbf{B} \cup \mathbf{C} \cup \mathbf{D})$		86	2	-7	0	8
$(\mathbf{B} \cup \mathbf{C} \cup \mathbf{D})$ **-** $\mathbf{A}$		-29	45	11	3	0
$\mathbf{A}$ **-** $(\mathbf{B} \cap \mathbf{C} \cap \mathbf{D})$		156	9	-2	5	8
$(\mathbf{B} \cap \mathbf{C} \cap \mathbf{D})$ **-** $\mathbf{A}$		-2	0	2	0	0

Table 6.5: Rate of Decrease of False Positives For All Threshold

Algo	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**
A	251	354	360	360	360	361
B	324	347	366	375	388	388
C	337	362	382	390	391	392
D	351	371	382	390	392	392
$\mathbf{B} \cap \mathbf{C} \cap \mathbf{D}$	358	378	384	391	392	392
$\mathbf{B} \cup \mathbf{C} \cup \mathbf{D}$	309	340	365	374	387	388
$\mathbf{A} \cap \mathbf{B} \cap \mathbf{C} \cap \mathbf{D}$	358	379	384	391	392	392
$\mathbf{A} \cup \mathbf{B} \cup \mathbf{C} \cup \mathbf{D}$	249	335	354	358	360	361
$\mathbf{A}$ **-** $(\mathbf{B} \cup \mathbf{C} \cup \mathbf{D})$	332	387	381	376	365	365
$(\mathbf{B} \cup \mathbf{C} \cup \mathbf{D})$ **-** $\mathbf{A}$	390	373	386	390	392	392
$\mathbf{A}$ **-** $(\mathbf{B} \cap \mathbf{C} \cap \mathbf{D})$	285	367	368	361	360	361
$(\mathbf{B} \cap \mathbf{C} \cap \mathbf{D})$ **-** $\mathbf{A}$	392	391	392	392	392	392

Table 6.6: False Negative For All Threshold

Algo	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**
A		103	6	0	0	1
B		23	19	9	13	0
C		25	20	8	1	1
D		20	11	8	2	0
B ∩ **C** ∩ **D**		20	6	7	1	0
B ∪ **C** ∪ **D**		31	25	9	13	1
A ∩ **B** ∩ **C** ∩ **D**		21	5	7	1	0
A ∪ **B** ∪ **C** ∪ **D**		86	19	4	2	1
A - (B ∪ **C** ∪ **D)**		55	-6	-5	-11	0
(B ∪ **C** ∪ **D) - A**		-17	13	4	2	0
A - (B ∩ **C** ∩ **D)**		82	1	-7	-1	1
(B ∩ **C** ∩ **D) - A**		-1	1	0	0	0

Table 6.7: Rate of Increase of False Negative For All Threshold

Algo	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**
A	353	547	563	563	568	576
B	468	538	576	588	595	595
C	475	542	583	592	595	595
D	544	581	590	595	595	595
$\mathbf{B} \cap \mathbf{C} \cap \mathbf{D}$	548	584	591	595	595	595
$\mathbf{B} \cup \mathbf{C} \cup \mathbf{D}$	431	510	569	587	595	595
$\mathbf{A} \cap \mathbf{B} \cap \mathbf{C} \cap \mathbf{D}$	548	586	593	595	595	595
$\mathbf{A} \cup \mathbf{B} \cup \mathbf{C} \cup \mathbf{D}$	323	488	549	560	568	576
$\mathbf{A}$ **-** $(\mathbf{B} \cup \mathbf{C} \cup \mathbf{D})$	487	573	575	568	568	576
$(\mathbf{B} \cup \mathbf{C} \cup \mathbf{D})$ **-** $\mathbf{A}$	565	536	581	592	595	595
$\mathbf{A}$ **-** $(\mathbf{B} \cap \mathbf{C} \cap \mathbf{D})$	400	556	565	563	568	576
$(\mathbf{B} \cap \mathbf{C} \cap \mathbf{D})$ **-** $\mathbf{A}$	595	593	593	595	595	595

Table 6.8: True Negative For All Threshold

Algo	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**
A		194	16	0	5	8
B		70	38	12	7	0
C		67	41	9	3	0
D		37	9	5	0	0
B ∩ **C** ∩ **D**		36	7	4	0	0
B ∪ **C** ∪ **D**		79	59	18	8	0
A ∩ **B** ∩ **C** ∩ **D**		38	7	2	0	0
A ∪ **B** ∪ **C** ∪ **D**		165	61	11	8	8
A - (B ∪ **C** ∪ **D)**		86	2	-7	0	8
(B ∪ **C** ∪ **D) - A**		-29	45	11	3	0
A - (B ∩ **C** ∩ **D)**		156	9	-2	5	8
(B ∩ **C** ∩ **D) - A**		-2	0	2	0	0

Table 6.9: Rate of Increase of True Negative For All Threshold

Word in model solution	Synonyms in student diagram	Hypernym/Hyponym in student diagram
total	amount, balance	credit, debit
valid	accepted	
check	identify, detect	
time		offpeak, peak, hours
update		reduce
beep		sound, audible warning
invalid		error, problem, reject
warning		notice

Table 6.10: Synonyms in student diagrams

the name of the concrete class from the configuration XML file, creates an instance of that concrete class and calls its methods.

```
Scanner sc = new Scanner(new File(’’configuration.xml’’));
String strClassName=sc.next();
Class myClass1 =Class.forName(strClassName);
StemmingInterface stemmingInterface=null;
stemmingInterface=(StemmingInterface)myClass1.newInstance();
String result=stemmingInterface.getStem(’’mylabel1’’);
```

6.8 Sample code to configure the combined hybrid syntax algorithm

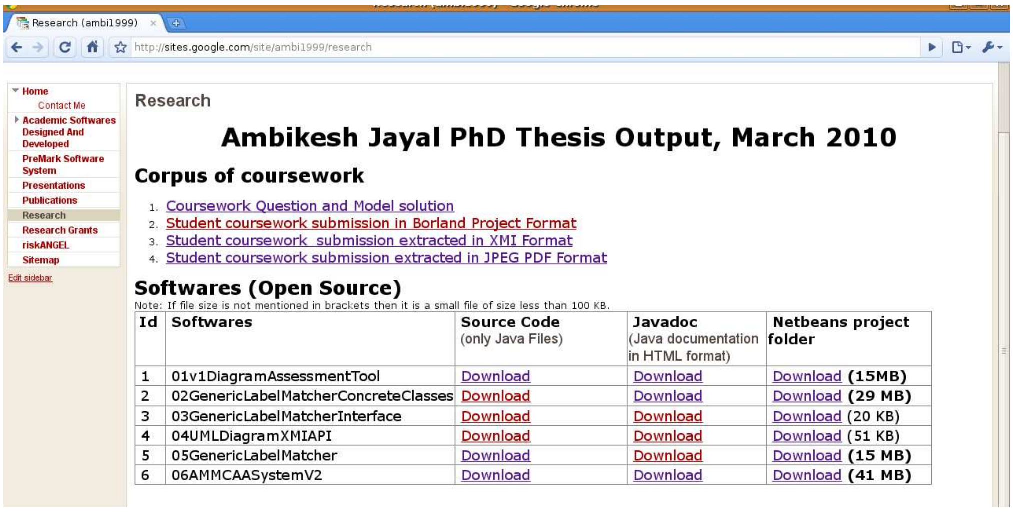
The user can configure the combined hybrid syntax algorithm using the configuration XML file. To configure the hybrid algorithm to include an additional syntax algorithm, add the following entry in the configuration XML file. Please note that the user needs to provide the concrete class for the syntax algorithm.

¡algorithm name=”NewSyntaxAlgorithm” active=”true”¿ ¡concreteclass¿NewSyntaxAlgorithmCo
¡/algorithm¿

To configure the hybrid algorithm to exclude a syntax algorithm, either delete the entry of the syntax algorithm from the configuration XML file or set the value of active attribute to false.

6.9 Software for framework (Java code) and Corpus of coursework

Figure 6.3 shows the screen shot of the software and corpus of coursework published as part of this Thesis at (Jaya 10).



Research

Ambikesh Jayal PhD Thesis Output, March 2010

Corpus of coursework

1. Coursework Question and Model solution
2. Student coursework submission in Borland Project Format
3. Student coursework submission extracted in XMI Format
4. Student coursework submission extracted in JPEG PDF Format

Softwares (Open Source)

Note: If file size is not mentioned in brackets then it is a small file of size less than 100 KB.

Id	Softwares	Source Code (only Java Files)	Javadoc (Java documentation in HTML format)	Netbeans project folder
1	01v1DiagramAssessmentTool	Download	Download	Download **(15MB)**
2	02GenericLabelMatcherConcreteClasses	Download	Download	Download **(29 MB)**
3	03GenericLabelMatcherInterface	Download	Download	Download (20 KB)
4	04UMLDiagramXMIAPI	Download	Download	Download (51 KB)
5	05GenericLabelMatcher	Download	Download	Download **(15 MB)**
6	06AMMCAASystemV2	Download	Download	Download **(41 MB)**

Figure 6.3: Screen shot for the published Software and Corpus of coursework

Chapter 7

Appendix B

7.1 Java Source code for DiagramAssessmentTool.jar

This is the main program that creates a swing GUI with which the end user interacts.

```java
package uk.ac.brunel.gui;

/**
 * @author       Ambikesh Jayal, School of IS, Computing & Maths, Brunel University.
 * @author       ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version      1.0,  25-Aug-2009
 * @since JDK1.6
 */

import java.awt.*;
import javax.swing.*;
import javax.swing.GroupLayout.*;
import java.awt.event.*;

import java.util.Arrays;
import uk.ac.brunel.genericlabelmatcher.LabelMatcher;
import uk.ac.brunel.xmi.XMIDOMDataLoader;
```

```
public class DiagramAssessmentToolGUI {

    public static void main(String[] args) {

        DiagramAssessmentToolGUI diagramAssessmentToolGUI = new
            DiagramAssessmentToolGUI();
        diagramAssessmentToolGUI.generateGUI();

    }
    JTextField tfCorrectAnswerFilePath = new JTextField();
    JTextField tfStudentAnswerFilePath = new JTextField();
    JTextArea textAreaForFeedback= new JTextArea();

    public void generateGUI() {

        JLabel labelTitle = new JLabel(”Brunel University LTDU CIF 2009
            Project”);

        JFrame frame = new JFrame(”Brunel University LTDU CIF 2009
            Project”);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400,400);

        GroupLayout layout = new GroupLayout(frame.getContentPane());
        frame.getContentPane().setLayout(layout);



        JLabel labelCorrectAnswerFilePath = new JLabel(”
            strCorrectAnswerFilePath”);
        JLabel labelStudentAnswerFilePath = new JLabel(”
            strStudentAnswerFilePath”);
        JLabel labelForButton=new JLabel(”Please Press button ”);
        JLabel labelForFeedback = new JLabel(”Feedback”);

        JButton jbutton1 = new JButton(”Automatically Mark”);


        textAreaForFeedback = new JTextArea();
        textAreaForFeedback.setColumns(100);
        textAreaForFeedback.setLineWrap(true);
        textAreaForFeedback.setRows(40);
        textAreaForFeedback.setWrapStyleWord(true);
        textAreaForFeedback.setEditable(false);
        JScrollPane jScrollPane1 = new JScrollPane(textAreaForFeedback);
        frame.getContentPane().add(jScrollPane1);


        jbutton1.addActionListener(new MyTestActionListener());



        layout.setAutoCreateGaps(true);

        layout.setAutoCreateContainerGaps(true);



        GroupLayout.SequentialGroup hGroup = layout.createSequentialGroup
            ();

          hGroup.addGroup(layout.createParallelGroup(Alignment.LEADING).
                addComponent(labelCorrectAnswerFilePath).addComponent(
                    labelStudentAnswerFilePath).addComponent(
                    labelForButton).addComponent(labelForFeedback));
        hGroup.addGroup(layout.createParallelGroup(Alignment.LEADING).
                addComponent(tfCorrectAnswerFilePath).addComponent(
                    tfStudentAnswerFilePath).addComponent(jbutton1).
                    addComponent(textAreaForFeedback));


        layout.setHorizontalGroup(hGroup);



        GroupLayout.SequentialGroup vGroup = layout.createSequentialGroup
           ();

        vGroup.addGroup(layout.createParallelGroup(Alignment.LEADING).
                addComponent(labelCorrectAnswerFilePath).addComponent(
                    tfCorrectAnswerFilePath));
        vGroup.addGroup(layout.createParallelGroup(Alignment.LEADING).
                addComponent(labelStudentAnswerFilePath).addComponent(
                    tfStudentAnswerFilePath));

        vGroup.addGroup(layout.createParallelGroup(Alignment.LEADING).
           addComponent(labelForButton).addComponent(jbutton1));

        vGroup.addGroup(layout.createParallelGroup(Alignment.LEADING).
                addComponent(labelForFeedback).addComponent(
                    textAreaForFeedback));

        layout.setVerticalGroup(vGroup);

        frame.pack();
        frame.setVisible(true);
    }
//start of action listener code
    class MyTestActionListener implements ActionListener {

        public void actionPerformed(ActionEvent ae) {
```

```
        //String strCorrectAnswerFilePath = "/home/ambi/01MYRES/01Am/LTDU
           /LTDU Symposium/v3_Correct Diagrams.xmi";
        //String strStudentAnswerFilePath = "/home/ambi/01MYRES/01Am/LTDU
           /LTDU Symposium/v3_Correct Diagrams.xmi";

        String strCorrectAnswerFilePath ="";
        String strStudentAnswerFilePath ="";
        String strOutputFeedback = "";
        int marks=0;

        String[][] arrayStringOutput=null;

            try{
            strCorrectAnswerFilePath=tfCorrectAnswerFilePath.getText();
            strStudentAnswerFilePath=tfStudentAnswerFilePath.getText();

            //Using the UMLDiagramXMIAPI.jar
            XMIDOMDataLoader xmiDOMDataLoader = new XMIDOMDataLoader();
            String[][] strArrayLabelDetailsOfCorrectAnswer=new
               XMIDOMDataLoader().getDetailsOfAllLabels(
               strCorrectAnswerFilePath);
            String[][] strArrayLabelDetailsOfStudentAnswer=new
               XMIDOMDataLoader().getDetailsOfAllLabels(
               strStudentAnswerFilePath);

            //Using the GenericLabelMatcher.jar
              LabelMatcher labelMatcher=LabelMatcher.getInstance("");

    //String[] arrayStr1={"Select Recipe", "Assemble Ingredients", "Cook
       meal", "Set the table", "Eat"};
    //String[] arrayStr2={"turn up at arranged time", "eat the meal
       produced by friend", "must select a recipe for their meal",
  //"assemble the ingredient", "cook the meal", "Set table", "eat meal"};

String[] arrayStr1=new String[strArrayLabelDetailsOfCorrectAnswer.length];
String[] arrayStr2=new String[strArrayLabelDetailsOfStudentAnswer.length];

for(int i=0;i<arrayStr1.length;i++){
     arrayStr1[i]=strArrayLabelDetailsOfCorrectAnswer[i][0];
}

for(int i=0;i<arrayStr2.length;i++){
     arrayStr2[i]=strArrayLabelDetailsOfStudentAnswer[i][0];
}

 double[][] arrayDetailsOfMatchedStringsFromSecondArray=labelMatcher.getDetailsOfMatchedStringsFromSecondArray(arrayStr1, arrayStr2);

 for(int i=0;i<arrayDetailsOfMatchedStringsFromSecondArray.length;i++){
      if(arrayDetailsOfMatchedStringsFromSecondArray[i][1]>=0){
           //match has been found, so positive feedback
           strOutputFeedback=strOutputFeedback+strArrayLabelDetailsOfCorrectAnswer[i][2] + "\n \n";
           marks++;
      }else{
      //match has not been found, so negative feedback
           strOutputFeedback=strOutputFeedback+strArrayLabelDetailsOfCorrectAnswer[i][3] + "\n \n";
      }
}
```

```
strOutputFeedback=strOutputFeedback+ "\n\n Total Makrs: " + marks;

//add raw feedback
strOutputFeedback=strOutputFeedback+ "\n\n Raw Feedback: \n\n ";
strOutputFeedback=strOutputFeedback+"arrayDetailsOfMatchedStringsFromSecondArray[][]: \n + " + Arrays.deepToString(arrayDetailsOfMatchedStringsFromSecondArray) + "\n\n";
strOutputFeedback=strOutputFeedback+"strArrayLabelDetailsOfCorrectAnswer[][]: \n + " + Arrays.deepToString(strArrayLabelDetailsOfCorrectAnswer) + "\n\n";
strOutputFeedback=strOutputFeedback+"arrayStr1[]: \n + " + Arrays.deepToString(arrayStr1) + "\n\n";
strOutputFeedback=strOutputFeedback+"strArrayLabelDetailsOfStudentAnswer[][]: \n + " + Arrays.deepToString(strArrayLabelDetailsOfStudentAnswer) + "\n\n";
strOutputFeedback=strOutputFeedback+"arrayStr2[]: \n + " + Arrays.deepToString(arrayStr2) + "\n\n";

for(int i=0;i<arrayDetailsOfMatchedStringsFromSecondArray.length;i++){
//      System.out.println(arrayDetailsOfMatchedStringsFromSecondArray[i][0] + "     " + arrayDetailsOfMatchedStringsFromSecondArray[i][1] + "     " +arrayDetailsOfMatchedStringsFromSecondArray[i][2]  );
}



        }catch(Throwable ex){
            ex.printStackTrace();

            strOutputFeedback= strOutputFeedback + "SORRY, SOME EXCEPTION OCCURRED. PLEASE RERUN OR CONTACT
```

```
                                ADMINISTRATOR";
                        strOutputFeedback=strOutputFeedback + "\n\n ************ Exception Message \n\n ex.getMessage() + \n" + ex.getMessage();
                        strOutputFeedback=strOutputFeedback + "ex.getLocalizedMessage() \n" + ex.getLocalizedMessage();
                        strOutputFeedback=strOutputFeedback + "ex.toString() \n" + ex.toString();
                }

                textAreaForFeedback.setText(strOutputFeedback);

        }
    }
}
```

7.2 Java Source code for UMLDiagramXMIAPI.jar

This component extracts the labels from UML diagrams in XMI format.

```
package uk.ac.brunel.xmi;

import javax.xml.parsers.*;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import org.w3c.dom.*;
import org.w3c.dom.Node.*;

/**
 * Java API to access the labels present in a diagram represented as an XMI file.
 * @author       Ambikesh Jayal, School of IS, Computing & Maths, Brunel
    University.
 * @author       ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version      1.0,  25-Aug-2009
 * @since JDK1.6
 */


public class XMIDOMDataLoader {

    private static final String UMLActionState = "UML:ActionState";
    private static final String UMLModelElementtaggedValue = "UML:
        ModelElement.taggedValue";
    private static final String UMLTaggedValuedataValue = "UML:
        TaggedValue.dataValue";
    private static String strFilePath = "/home/ambi/01MYRES/01Am/LTDU/
        LTDU Symposium/v3_Correct Diagrams.xmi";
    //for testing
    public static void main(String[] args) {
        XMIDOMDataLoader xmiDOMDataLoader = new XMIDOMDataLoader();
        String strFilePath = "/home/ambi/01MYRES/01Am/LTDU/LTDU Symposium
            /v3_Correct Diagrams.xmi";
        new XMIDOMDataLoader().getDetailsOfAllLabels(strFilePath);

    }

    /*
     * This function returns list of labels. string[i][0]="lablel",
        string[i][1]="xmiid", string[i][2]="positive feedback",
     * string[i][3]="negative feedback"
     *
     */
    public String[][] getDetailsOfAllLabels(String strXMIFilePath) {
```

```
    String[][] strArrayLabelDetails = null;
    if (!strXMIFilePath.endsWith(”.xmi”)) {
        //file does not end with xmi, so do nothing
        strArrayLabelDetails = null;
    } else {
        //now it ends with .xmi and hence probaby it is indeed an xmi
            file, process it

        try {
            DocumentBuilderFactory factory = DocumentBuilderFactory.
                newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();

            Document document = builder.parse(strXMIFilePath);
            strArrayLabelDetails = processDocument(document);
        } catch (Exception ex) {
            ex.printStackTrace();
        }


    }

    System.out.println(”Arrays.deepToString(strArrayLabelDetails) ” +
        Arrays.deepToString(strArrayLabelDetails));
    return strArrayLabelDetails;
}

/*
 * +--------------------------------------------+
 * | METHOD: printElements                      |
 * +--------------------------------------------+
 */
private String[][] processDocument(Document doc) {
```

```java
        String[][] strArrayLabelDetails = null;
        List<String[]> listStates = new ArrayList<String[]>();

        NodeList nodelist = doc.getElementsByTagName("*");
        Node node;

        for (int i = 0; i < nodelist.getLength(); i++) {
            node = nodelist.item(i);
            System.out.println("****" + node.getNodeName() + " ");
            String strNodeName = node.getNodeName();

            if (strNodeName.equals(UMLActionState)) {
                //this starts the processiong of a state machine
                String[] arrayLabelDetail = processUMLActionState(node);
                if (arrayLabelDetail != null) {
                    listStates.add(arrayLabelDetail);
                }
            }

        }

        strArrayLabelDetails = new String[listStates.size()][];
        for (int i = 0; i < listStates.size(); i++) {
            strArrayLabelDetails[i] = listStates.get(i);
        }

        return strArrayLabelDetails;
    }

    private String[] processUMLActionState(Node nodeActionState) {
        String[] arrayLabelDetail = new String[4];

        String label = "";
        String xmiid = ””;
        String feedback = ””;
        String positiveFeedback = ””;
        String negativeFeedback = ””;

        NamedNodeMap namedNodeMap = nodeActionState.getAttributes();
        try {
            label = namedNodeMap.getNamedItem(”name”).getNodeValue();
            xmiid = namedNodeMap.getNamedItem(”xmi.id”).getNodeValue();
        } catch (Exception ex) {
        }

        if (label.equals(””)) {
            return null;
        }

        //the followinng two lines with lots of getFirstChild().
            getNextSibling() are a way to get to the feedback value which
             is present in the
        //tag <UML:TaggedValue.dataValue>.
        Node child1 = nodeActionState.getFirstChild().getNextSibling().
            getFirstChild().getNextSibling();
        Node childNodeUMLTaggedValuedataValue = child1.getFirstChild().
            getNextSibling();

        String nameChildNodeUMLTaggedValuedataValue =
            childNodeUMLTaggedValuedataValue.getNodeName();
        // System.out.println(”^^^^^ nameChildNodeUMLTaggedValuedataValue
             ” + nameChildNodeUMLTaggedValuedataValue);
        if (nameChildNodeUMLTaggedValuedataValue.equals(
            UMLTaggedValuedataValue)) {
            feedback = childNodeUMLTaggedValuedataValue.getTextContent();
        }
        arrayLabelDetail[0] = label;
        arrayLabelDetail[1] = xmiid;
        arrayLabelDetail[2] = feedback.split("]")[0].replace("[", "").replace("]", "");
        arrayLabelDetail[3] = feedback.split("]")[1].replace("[", "").replace("]", "");
        //arrayLabelDetail[2]=feedback;
        //arrayLabelDetail[3]=feedback;

        return arrayLabelDetail;
    }


}
```

7.3 Java Source code for GenericLabelMatcher.jar

This component matches the labels.

```
package uk.ac.brunel.iface;
/**
 * @author      Ambikesh Jayal, School of IS, Computing & Maths, Brunel University.
 * @author      ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version     1.0,  25-Aug-2009
 * @since JDK1.6
 */

public interface SearchAlgorithmIF{
```

```java
/*
 * syntaxalgorithms[i][0] contains the id of the syntax algorithm class
 * syntaxalgorithms[i][1] contains the double value of the similarity index for the corresponding syntax algorithm in the syntaxalgorithms[i][0]
 **/

public double getCombinedSimilarity(String[][] arraySyntaxSimilarityIndex);
}
```

```java
package uk.ac.brunel.iface;
/**
 * @author       Ambikesh Jayal, School of IS, Computing & Maths, Brunel University.
 * @author       ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version      1.0,  25-Aug-2009
 * @since JDK1.6
 */

public interface SpellCheckerIF{

/*
 * This fucntion returns true if word has been misspelled otherwise false.
 **/
public boolean isMisspelled(String strWord);

/*
 * This function takes a single word or a sentence consisting of words separated by single or multiple space.
 * It then returns the autocottected version of each word,
 * This fucntion returns the first suggested word by the spell checker.
 * Incase the word in not spelled incorrectly then this function returns the same word
 **/
 public String getSpellCheckedText(String strWord);

 /*
  * * This function takes an array of single words.
 * It then returns an array the autocottected version of each word,
 * This fucntion returns the first suggested word by the spell checker. Incase the word in not spelled incorrectly then this function returns the same word
 **/
 public String[] getSpellCheckedText(String[] strWord);

}
```

```
package uk.ac.brunel.iface;
/**
 * @author      Ambikesh Jayal, School of IS, Computing & Maths, Brunel University.
 * @author      ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version     1.0,  25-Aug-2009
 * @since JDK1.6
 */
public interface StemmingAlgorithmIF{



 /*
This function takes a single word or a sentence consisting of words separated by single or multiple space. It then returns the stem of each word,
//inputText can wither be a single word or be a sentence consisting of words separated by single or multiple space
```

```java
*/

public String getStemText(String inputText);


}
```

```java
package uk.ac.brunel.iface;
/**
 * @author      Ambikesh Jayal, School of IS, Computing & Maths, Brunel University.
 * @author      ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version     1.0,  25-Aug-2009
 * @since JDK1.6
 */

public interface SyntaxAlgorithmIF{
public double similarity(String str1, String str2);
}
```

```java
package uk.ac.brunel.genericlabelmatcher;

/**
 * @author      Ambikesh Jayal, School of IS, Computing & Maths, Brunel University.
 * @author      ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version     1.0,  25-Aug-2009
 * @since JDK1.6
 */

public class CommonPaths {

    public static String DATASTORE_BASE_PATH = "";
    public static String DATA_SET_NAME = "";
    public static String DATA_PATH = "";
    public static String REPORTS_PATH = "";
    public static String DATADICTIONARY_FILE_PATH = "";
    public static String FEATURES_PROPERTIES_PATH = "";
    public static String LABEL_MATCHING_ALGORITHM_PROPERTIES_PATH = "";
    public static String ALGORITHM_PROPERTIES_PATH = "";
    public static String STUDENT_ANSWER_PATH = "";
    public static String CORRECT_ANSWER_PATH = "";
    public static String PaiceHusk_Stemmer_RULES_FILE_PATH = "";
    public static String MASTER_DECISION_FILE_PATH = "";
    public static String WORDNET_HOME_PATH="";
    public static String WORDNET_BASE_PATH="";
    public static String WORDNET_RESOURCES_PATH = "";

    public static String TEMP_FOLDER_PATH = "";


    //set paths
    static {
        DATASTORE_BASE_PATH = "/home/ambi/01MYRES/02SwDev/0001_BEAS/01
            _SOFTWARE/01DATASTORE/";
        DATA_SET_NAME = "set10";

        DATA_PATH = DATASTORE_BASE_PATH + DATA_SET_NAME + "/data/";
        REPORTS_PATH = DATASTORE_BASE_PATH + DATA_SET_NAME + "/reports/";
        DATADICTIONARY_FILE_PATH = DATASTORE_BASE_PATH + DATA_SET_NAME +
            "/datadictionary/datadictionary.properties";
        FEATURES_PROPERTIES_PATH = DATASTORE_BASE_PATH + DATA_SET_NAME +
            "/features/features.properties";

        STUDENT_ANSWER_PATH = DATA_PATH + "studentanswer/";
        CORRECT_ANSWER_PATH = DATA_PATH + "correctanswer/";

        PaiceHusk_Stemmer_RULES_FILE_PATH = DATASTORE_BASE_PATH + "
            stemming/stemrules.txt";

        LABEL_MATCHING_ALGORITHM_PROPERTIES_PATH = DATASTORE_BASE_PATH +
            "properties/label_matching_algorithm.properties";

        MASTER_DECISION_FILE_PATH = DATASTORE_BASE_PATH + "Report for
            Framework/Main Reports/Syntax Algo Conparision Report/
            level3_And_spell_check_And_Stemming/raw data/
            Main_09JUL2008_ALL_LEVEL_ALL_DETAILS.xls";

        //WORDNET_RESOURCES_PATH=DATASTORE_BASE_PATH + "resources/wordnet
            /";

        WORDNET_BASE_PATH=DATASTORE_BASE_PATH + "resources/wordnet/";

        WORDNET_RESOURCES_PATH=WORDNET_BASE_PATH + "wordnet_configuration
            /";

        WORDNET_HOME_PATH=WORDNET_BASE_PATH + "WordNet-3.0/";


        TEMP_FOLDER_PATH=DATASTORE_BASE_PATH + "tempfolder/";



    }

    public static String LTDU_Project_Data_File_Path="/home/ambi/01MYRES
        /01Am/LTDU/LTDU Project Stuff/";
}
```

```
package uk.ac.brunel.genericlabelmatcher;

/**
 * @author      Ambikesh Jayal, School of IS, Computing & Maths, Brunel University.
 * @author      ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version     1.0,  25-Aug-2009
 * @since JDK1.6
 */

import java.util.*;
import uk.ac.brunel.iface.SpellCheckerIF;
import uk.ac.brunel.iface.StemmingAlgorithmIF;

public class CommonUtilityService {
private static ToolConfiguration toolConfiguration=new ToolConfiguration();

    public static String[] processStringArray(String[] arrayStr) {
        String[] processedStringArray = new String[arrayStr.length];
        for (int i = 0; i < arrayStr.length; i++) {
            processedStringArray[i] = processString(arrayStr[i]);
        }
        return processedStringArray;
    }

    public static String processString(String str1) {
        if (str1 == null) {
            str1 = "";
        }
        //1. Remove ending and traliing spaces and Convert to Lowercase.
        str1 = str1.trim().toLowerCase();
        //2. Remove all special character. Special character DOES NOT include the inbetween spaces.
        //ListSpecialCharacters={!,”,   ,$,%,^,&,*,(,),−,_,+,=,{,},[,],~,#,;,:,@,’,<,>,.,?,/,‘,   ,\,|,  ,}
        str1 = processForSpecialCharacter(str1);

        //Note: First remove the punctuation words like the, to etc and then remove the inbetween spaecs.
        //Do not do it the other way round.
        //This is because to remove the punctuation words at the very start of the string or at the very end of the string,
        //we need to split the string using the SPACE. And so we need the inbetween spaces for processing the string for
        //punctuation words.



        //3. Remove the inbetween extra spaces except one.
        str1 = processForInbetweenSpace(str1);

        //4. replace abbreviations with their expanded form
        str1 = processForAbbreviations(str1);

        //5. Spell check
        str1 = processForSpellCheck(str1);

        //6. Remove all the punctuation symbols.
        //ListPunctuationWords={the,or,and,if,but,is,are,do,did,has,have,been,for,an,at}
        str1 = processForStopwords(str1);

        //7. Stemming
        str1 = processForStemming(str1);
        return str1;
    }

public static String processForSpellCheck(String s1){
String str1=s1;
    //stemming algorithm
    String[][] spellcheckeralgorithms=toolConfiguration.
       spellcheckeralgorithms;
    for(int i=0;i<spellcheckeralgorithms.length;i++){
    String active=spellcheckeralgorithms[i][2];
  if(active.equalsIgnoreCase("true")){
        //this is the main active search algorithm
          //searchalgorithm[i][3] contains the name of the concrete class

          //String strNameOfConcreteClass=searchalgorithm[i][3];
          String strNameOfConcreteClass=toolConfiguration.PACKAGE_NAME+
              spellcheckeralgorithms[i][3];

           try{
    SpellCheckerIF spellCheckerIF=(SpellCheckerIF)Class.forName(
       strNameOfConcreteClass).newInstance();
    str1=spellCheckerIF.getSpellCheckedText(str1);

    }catch(ClassNotFoundException ex){
        System.out.println("Sorry the class [" + strNameOfConcreteClass +
            "] has not been found. Please either create it or if already
            created, place " +
                "it in the classpath. ");
        ex.printStackTrace();
    }catch(Exception ex){
        System.out.println("Sorry, other exception");
        ex.printStackTrace();
    }
    break;
          }else{
                  continue;
                  }
    }
return str1;
}

public static String processForStemming(String s1){
    String str1=s1;
    //stemming algorithm
    String[][] stemmingalgorithms=toolConfiguration.stemmingalgorithms;
    for(int i=0;i<stemmingalgorithms.length;i++){
    String active=stemmingalgorithms[i][2];
  if(active.equalsIgnoreCase("true")){
        //this is the main active search algorithm
          //searchalgorithm[i][3] contains the name of the concrete class

          //String strNameOfConcreteClass=searchalgorithm[i][3];
          String strNameOfConcreteClass=toolConfiguration.PACKAGE_NAME+
             stemmingalgorithms[i][3];

           try{
    StemmingAlgorithmIF stemmingAlgorithmIF=(StemmingAlgorithmIF)Class.
       forName(strNameOfConcreteClass).newInstance();
    str1=stemmingAlgorithmIF.getStemText(str1);

    }catch(ClassNotFoundException ex){
        System.out.println("Sorry the class [" + strNameOfConcreteClass +
            "] has not been found. Please either create it or if already
            created, place " +
                "it in the classpath.");
        ex.printStackTrace();
    }catch(Exception ex){
        System.out.println("Sorry, other exception");
        ex.printStackTrace();
    }

    break;
          }else{
                  continue;
                  }
    }
return str1;
}
    public static List getSpecialCharacterList() {
        List listSpecialCharacter = new ArrayList();
        String strWordList = ToolConstants.STR_SPECIAL_CHARACTER;
        String strSeparator = ToolConstants.SEPARATER1;

        String[] arrStr1 = strWordList.split(strSeparator);

        for (int i = 0; i < arrStr1.length; i++) {
            listSpecialCharacter.add(arrStr1[i]);
        }
        return listSpecialCharacter;
    }

    public static Map<String, String> getAbbreviationAndTheirExpandedForm(){
    Map<String, String> mapAbbreviationAndTheirExpandedForm=new HashMap<String, String>();
    mapAbbreviationAndTheirExpandedForm.put("msg", "message");
    mapAbbreviationAndTheirExpandedForm.put("exp", "expiry");
```

```java
    return mapAbbreviationAndTheirExpandedForm;
    }

        public static String processForAbbreviations(String str1) {
        String str2 = "";
        //for punctuation words
        Map<String, String> mapAbbreviationAndTheirExpandedForm = CommonUtilityService.getAbbreviationAndTheirExpandedForm();
        //List listStr1 = Arrays.asList(str1.split(ToolConstants.SPACE));
        //System.out.println("listStr1.getClass().getName(): ["+listStr1.getClass().getName()+"]");
        //System.out.println("listStr1: ["+listStr1+"]");

        String[] arrString1 = str1.split(ToolConstants.SPACE);
        String[] arrString2= new String[arrString1.length];

        for (int i = 0; i < arrString1.length; i++) {
            String key1=arrString1[i];
            System.out.println("*********** [" + key1 + "]      ");
           if(mapAbbreviationAndTheirExpandedForm.containsKey(key1)){
            String expandedForm=mapAbbreviationAndTheirExpandedForm.get(key1);
            //replace will not work because it will replace all
            //str2=str2.replace(key1, expandedForm);
            arrString2[i]=expandedForm;
           }else{
            arrString2[i]=key1;
           }
        }

        for (int i = 0; i < arrString2.length; i++) {
            if(str2.equals("")){
                str2=arrString2[i];
            }else{
            str2=str2+" " + arrString2[i];
            }
        }

        return str2;
    }


    public static List getPunctuationWordList() {
        List listPunctuationWord = new ArrayList();
        String strWordList = ToolConstants.STR_PUNCTUATION_WORD;
        String strSeparator = ToolConstants.SEPARATER1;

        String[] arrStr1 = strWordList.split(strSeparator);

        for (int i = 0; i < arrStr1.length; i++) {
            listPunctuationWord.add(arrStr1[i]);
        }
        return listPunctuationWord;
    }

    //Note: This function in a way removes double or more in between
       spaces also because the split function
    //by defaut trims each word in the string. So no need to call the
       function processForInbetweenSpace() after
    //calling this function processForPunctuationWord().
    //public static String processForPunctuationWord(String str1) {
    public static String processForStopwords(String str1) {
        String str2 = "";
        //for punctuation words
        List listPunctuationWord = CommonUtilityService.
           getPunctuationWordList();
```

```
        List listStr1 = Arrays.asList(str1.split(ToolConstants.SPACE));
        //System.out.println(”listStr1.getClass().getName(): [”+listStr1.getClass().getName()+”]”);
        //System.out.println(”listStr1: [”+listStr1+”]”);

        List listString1 = new ArrayList();
        String[] arrString1 = str1.split(ToolConstants.SPACE);
        for (int i = 0; i < arrString1.length; i++) {
            listString1.add(arrString1[i]);
        }
        listString1.removeAll(listPunctuationWord);
        for (int i = 0; i < listString1.size(); i++) {
            if (str2.equals(””)) {
                str2 = (String) listString1.get(i);
            } else {
                str2 = str2 + ” ” + (String) listString1.get(i);
            }
        }
        return str2;
    }

    public static String processForSpecialCharacter(String str1) {
        //for special characters
        List listSpecialCharacter = CommonUtilityService.getSpecialCharacterList();
        String strReplacement = ToolConstants.REPLACEMENT_CHARACTER1;
        for (int i = 0; i < listSpecialCharacter.size(); i++) {
            String strSpecialCharacter = (String) listSpecialCharacter.get(i);
            if (str1.contains(strSpecialCharacter)) {
                str1 = str1.replace(strSpecialCharacter, strReplacement);
            }
        }
        return str1;
    }

    //this function removes all the extra inbetween spaces except one.
    public static String processForInbetweenSpace(String inputText) {
        //for Inbetween Spaces
        StringTokenizer line = new StringTokenizer(””);
        String outputText = ””;

        line = new StringTokenizer(inputText);
        try {
            while (line.hasMoreTokens()) {
                String word = new String();
                word = line.nextToken();
                word = word.trim();
                if (outputText.equals(””)) {
                    outputText = word;
                } else {
                    outputText = outputText + ” ” + word;
                }

            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return outputText;

    }

    public static void main(String[] args) {

        //System.out.println(”UMLCommonUtilityService.
            getSpecialCharacterList(): ”+CommonUtilityService.
            getSpecialCharacterList());
        //System.out.println("UMLCommonUtilityService.
            getPunctuationWordList(): "+CommonUtilityService.
            getPunctuationWordList());
        //System.out.println("UMLCommonUtilityService.processStringLevel2
            (): "+CommonUtilityService.processStringLevel2(" SThe * ((
            tohel the loto na theme to  '\*))\"  "));
        //System.out.println("UMLCommonUtilityService.processStringLevel2
            (): "+CommonUtilityService.processStringLevel3("invalid
            audible warning"));

        //System.out.println("UMLCommonUtilityService.processStringLevel2
            (): ["+CommonUtilityService.processForRemovingEmbeddedSpaces
            ("invalid     audible      warning   ") + "]");

        //System.out.println("UMLCommonUtilityService.processStringLevel2
            (): " + CommonUtilityService.processString("invalid
            audible      warning"));

        //System.out.println("UMLCommonUtilityService.
            processForInbetweenSpace(): " + CommonUtilityService.
            processForInbetweenSpace("         invalid          audible
            warning      "));
        //System.out.println("UMLCommonUtilityService.
            processForInbetweenSpace(): " + new CommonUtilityService().
            processString("         invalid          audible      warning
            "));
        System.out.println("UMLCommonUtilityService.
            processForInbetweenSpace(): " + new CommonUtilityService().
            processForAbbreviations("       msg invalid        msg  audible
                expwarning   exp  "));
        }
}
```

```
package uk.ac.brunel.genericlabelmatcher;

/**
 * @author      Ambikesh Jayal, School of IS, Computing & Maths, Brunel University.
 * @author      ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version     1.0,  25-Aug-2009
 * @since JDK1.6
 */

import uk.ac.brunel.iface.SearchAlgorithmIF;
import uk.ac.brunel.iface.SyntaxAlgorithmIF;
import java.util.Arrays;

public class LabelMatcher{

private static ToolConfiguration toolConfiguration=null;
private static LabelMatcher labelMatcher=null;

private LabelMatcher(){
    toolConfiguration=new ToolConfiguration();
}

public static LabelMatcher getInstance(String STR_PATH_CONFIG_FOLDER){

if(labelMatcher==null){
labelMatcher=new LabelMatcher();
}
loadConfiguration(STR_PATH_CONFIG_FOLDER);
```

```
return labelMatcher;

}

//loads the configuration
/*
* STR_PATH_CONFIG_FOLDER : path of the folder contiining the config files
*/
private static void loadConfiguration(String STR_PATH_CONFIG_FOLDER){
toolConfiguration=new ToolConfiguration();
}

//public String[][] getMatchedStringPairs(String[] arrayStr1, String[] arrayStr2){
//      return null;
//      }

/*
 * Note: each element of the array arrayStr1 and arrayStr2 is a single trimmed (no leading and tralining spaces) word with no inbetween spaces.
 *
 * @return: a two dimensional array, with returnArray[i][0] containing value of the index in arrayStr1, returnArray[i][1] containing value of the index in arrayStr2 that matches the arrayStr1[i], and
 * returnArray[i][2] containing the corresponding similarity index value.  If no matching index is present then returnArray[i][1]=-1 , returnArray[i][2]=-1
 * and returnArray[i][1]=-1
 **/
public double[][] getDetailsOfMatchedStringsFromSecondArray(String[] arrayStr1, String[] arrayStr2){
        double[][] arrayDetailsOfMatchedStringsFromSecondArray=new double
            [arrayStr1.length][];

        String[] processedArrayStr1=CommonUtilityService.
            processStringArray(arrayStr1);
        String[] processedArrayStr2=CommonUtilityService.
            processStringArray(arrayStr2);

        double[][] arraySimilarityIndex=new double[processedArrayStr1.
            length][processedArrayStr2.length];
        for(int i=0;i<arraySimilarityIndex.length;i++){
             for(int j=0;j<arraySimilarityIndex[i].length;j++){
             arraySimilarityIndex[i][j]=getSyntaxSimilarityIndex(
                 processedArrayStr1[i],processedArrayStr2[j]);
             }
        }

        System.out.println("*********** " + Arrays.deepToString(
            arraySimilarityIndex));
        double threshold=toolConfiguration.threshold;
        //double threshold=0.2;
        arrayDetailsOfMatchedStringsFromSecondArray=getMatchedPairs(
            arraySimilarityIndex,threshold);
        return arrayDetailsOfMatchedStringsFromSecondArray;

}
/*
     //this is a fundamental function, it takes a 2D double array and
         returns matching pairs
     //Conceptual notion in the 2D array, First column= labels in model
         solution, first row labels in student diagram
     * @return: a two dimensional array, with returnArray[i][0] containing
          value of the index in arrayStr1, returnArray[i][1] containing
        value of the index in arrayStr2 that matches the arrayStr1[i],
        and
 * returnArray[i][2] containing the corresponding similarity index value.
      If no matching index is present then returnArray[i][1]=-1 ,
    returnArray[i][2]=-1
 * and returnArray[i][1]=-1
   */
   public static double[][] getMatchedPairs(double[][]
       arraySimilarityIndex, double threshold){
       /*
        double[][] arraySimilarityIndex1={{0.40,0.10,0.90,0.82},
                                          {0.20,0.30,0.95,0.80},
                                          {0.25,0.30,0.70,0.81}};
        *
        * Rerutns as follows
        * double[][] arrayValueOfIndexesOfMatchingPairs={{0.0,3.0,0.82},
                                                         {1.0,2.0,0.95},
                                                         {2.0,1.0,0.30}};

        */

       double[][] arrayValueOfIndexesOfMatchingPairs=new double[
           arraySimilarityIndex.length][];

       for(int i=0;i<arraySimilarityIndex.length;i++){
           int n=arraySimilarityIndex[i].length-1;//max value, Arrays.
               sort() sorts in ascending order and not in descending
               order
           int valueOfIndexOfNthMaxValue=-10;
           double value1=-1.0;
           //while(n<arraySimilarityIndex[i].length){
           //while(true){
           while(n>=0){
```

```
 //System.out.println("n  " + n);
 valueOfIndexOfNthMaxValue=getIndexOfNthMaxValue(n,
     arraySimilarityIndex[i]);
 //double value1=arraySimilarityIndex[i][
     valueOfIndexOfNthMaxValue];
 value1=arraySimilarityIndex[i][valueOfIndexOfNthMaxValue];
 double[] array1=new double[arraySimilarityIndex.length];
 for(int k=0;k<array1.length;k++){
      array1[k]=arraySimilarityIndex[k][
          valueOfIndexOfNthMaxValue];
 }
 int valueOfIndexOfNthMaxValue2=getIndexOfNthMaxValue(array1.
     length-1,array1);
 double value2=array1[valueOfIndexOfNthMaxValue2];
 if(value1>=value2){
 break;
 }
 n=n-1;
}

arrayValueOfIndexesOfMatchingPairs[i]=new double[3];
arrayValueOfIndexesOfMatchingPairs[i][0]=i;
System.out.println("%%% i=" + i+ " valueOfIndexOfNthMaxValue
    " + valueOfIndexOfNthMaxValue);
//if(arraySimilarityIndex[i][valueOfIndexOfNthMaxValue]>=
    threshold){
if(valueOfIndexOfNthMaxValue>=0 && arraySimilarityIndex[i][
    valueOfIndexOfNthMaxValue]>=threshold){
arrayValueOfIndexesOfMatchingPairs[i][1]=
    valueOfIndexOfNthMaxValue;
}else{
     arrayValueOfIndexesOfMatchingPairs[i][1]=-9;
}
```

```
            //value of the similarity index
            arrayValueOfIndexesOfMatchingPairs[i][2]=value1;
            //arrayValueOfIndexesOfMatchingPairs[i]={i,
               valueOfIndexOfNthMaxValue};
        }
        return arrayValueOfIndexesOfMatchingPairs;
    }
//this function finds the value of index of nth maximum
public static int getIndexOfNthMaxValue(int nthPosition, double[] numbers
   ){
  int indexOfMaxValue=0;
  double maxValue = numbers[0];
  double[] cloneNumbers = numbers.clone();
  Arrays.sort(cloneNumbers);
  for(int i=0;i<numbers.length;i++){
    if(numbers[i] ==cloneNumbers[nthPosition]){
        indexOfMaxValue=i;
          break;
        }
  }
  return indexOfMaxValue;
}


public double getSyntaxSimilarityIndex(String str1, String str2){

    String[][] syntaxalgorithms=ToolConfiguration.syntaxalgorithms;
    String[][] searchalgorithm=ToolConfiguration.searchalgorithm;

        double combinedSyntaxSimilarityIndex=0.0;
        //double[][] arraySyntaxSimilarityIndex=new double[
           syntaxalgorithms.length][2];
        String[][] arraySyntaxSimilarityIndex=new String[syntaxalgorithms.length][2];

        for(int i=0;i<syntaxalgorithms.length;i++){
        //String strNameOfConcreteClass="uk.ac.brunel.genericlabelmatcher.TestConcreteClass1";
        //syntaxalgorithms[i][3] contains the name of the concrete class
        //String strNameOfConcreteClass=syntaxalgorithms[i][3];
        String strNameOfConcreteClass=ToolConfiguration.PACKAGE_NAME+syntaxalgorithms[i][3];

   //syntaxalgorithms[i][0] contains the id of the syntax algorithm class
  arraySyntaxSimilarityIndex[i][0]=syntaxalgorithms[i][1];

  try{
  SyntaxAlgorithmIF syntaxAlgorithmIF=(SyntaxAlgorithmIF)Class.forName(strNameOfConcreteClass).newInstance();
  arraySyntaxSimilarityIndex[i][1]=syntaxAlgorithmIF.similarity(str1,str2)+"";

  }catch(ClassNotFoundException ex){
        System.out.println("Sorry the class [" + strNameOfConcreteClass + "] has not been found. Please either create it or if already created, place " +
                "it in the classpath.");
        ex.printStackTrace();
  arraySyntaxSimilarityIndex[i][1]=-1+"";

  }catch(Exception ex){
        System.out.println("Sorry, other exception");
        ex.printStackTrace();
  arraySyntaxSimilarityIndex[i][1]=-1+"";
  }
  }

  //search algorithm
  for(int i=0;i<searchalgorithm.length;i++){
  String active=searchalgorithm[i][2];
if(active.equalsIgnoreCase("true")){
      //this is the main active search algorithm
        //searchalgorithm[i][3] contains the name of the concrete class

        //String strNameOfConcreteClass=searchalgorithm[i][3];
        String strNameOfConcreteClass=ToolConfiguration.PACKAGE_NAME+
           searchalgorithm[i][3];

         try{
  SearchAlgorithmIF searchAlgorithmIF=(SearchAlgorithmIF)Class.forName(
     strNameOfConcreteClass).newInstance();
  combinedSyntaxSimilarityIndex=searchAlgorithmIF.getCombinedSimilarity
     (arraySyntaxSimilarityIndex);

  }catch(ClassNotFoundException ex){
      System.out.println("Sorry the class [" + strNameOfConcreteClass +
          "] has not been found. Please either create it or if already
          created, place " +
              "it in the classpath. ");
      ex.printStackTrace();
  arraySyntaxSimilarityIndex[i][1]=-1+"";

  }catch(Exception ex){
      System.out.println("Sorry, other exception");
      ex.printStackTrace();
  arraySyntaxSimilarityIndex[i][1]=-1+"";
  }
```

```
    break;
          }else{
                  continue;
                  }
    }

        return combinedSyntaxSimilarityIndex;
}

}
```

```
package uk.ac.brunel.genericlabelmatcher;
/**
 * @author       Ambikesh Jayal, School of IS, Computing & Maths, Brunel University.
 * @author       ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version      1.0,  25-Aug-2009
 * @since JDK1.6
 */

public interface LabelMatcherIF {

//loads the configuration
/*
* STR_PATH_CONFIG_FOLDER : path of the folder contiining the config files
*/
public LabelMatcher getInstance(String STR_PATH_CONFIG_FOLDER);

/*
 * Note: each element of the array arrayStr1 and arrayStr2 is a single trimmed (no leading and tralining spaces) word with no inbetween spaces.
 *
 * @return: a two dimensional array, with returnArray[i][0] containing
     value of the index in arrayStr2 that matches the arrayStr1[i], and
 * returnArray[i][1] containing the corresponding similarity index value.
       If no matching index is present then returnArray[i][0]=-1
 * and returnArray[i][1]=-1
 **/
public String[][] getMatchedStringPairs(String[] arrayStr1, String[]
    arrayStr2);
}
```

```
package uk.ac.brunel.genericlabelmatcher;

/**
 * @author     Ambikesh Jayal, School of IS, Computing & Maths, Brunel
     University.
 * @author     ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version    1.0,  25-Aug-2009
 * @since JDK1.6
 */

import java.io.File;

public class ToolConfiguration{

public static boolean casesensitive=true;
public static boolean trimming=true;

//global settings
public static boolean useSpecialcharacterxml=true;
public static boolean useStopwordxml=true;
public static boolean useAbbreviationxml=true;
public static boolean useSpellcheckeralgorithm=true;
```

```java
public static boolean useStemmer=true;
public static boolean useSynonymxml=true;


public static String specialcharacterxml="";
public static String stopwordxml="";
public static String abbreviationxml="";
public static String spellcheckeralgorithm="";
public static String stemmer="";
public static String synonymxml="";

public static double threshold=0.0;

/*<!--The value attribute of the packagename tag defines the package for all the concrete classes-->
<packagename value="uk.ac.brunel.concreteclasses"></packagename>
 */
public static String PACKAGE_NAME="";
//Note: The path strPathEnglishDictionary should contain two files named en_GB.dic and en_GB.aff
//String strEnglishDictionaryBaseFileName="/home/ambi/01MYRES/01Am/01_ResEAss/02-Work/04-Tools/Spell Checker/Dictionaries/en_GB/en_GB";
public static String strEnglishDictionaryBaseFileName="resources"+File.separator+"Dictionaries"+File.separator+"en_GB"+File.separator+"en_GB";

//="ExactMatchSyntaxAlgorithm" id="ExactMatchSyntaxAlgorithm" active="true" weight="0.1" concreteclass
//two dimensional array syntaxalgorithms[i][0]=name, syntaxalgorithms[i][1]=id, syntaxalgorithms[i][2]=active, syntaxalgorithms[i][3]=concreteclass,
//syntaxalgorithms[i][4]=weight
public static String[][] syntaxalgorithms=null;
```

```
//searchalgorithm[0][0]=name, searchalgorithm[0][1]=id, searchalgorithm[0][2]=active,searchalgorithm[0][3]=concreteclass
public static String[][] searchalgorithm=null;

public static String[][] spellcheckeralgorithms=null;

public static String[][] stemmingalgorithms=null;

public ToolConfiguration(){
test();
}

public void test(){

PACKAGENAME=”uk.ac.brunel.concreteclasses.”;

casesensitive=true;
trimming=true;

specialcharacterxml=””;
stopwordxml=””;
abbreviationxml=””;
spellcheckeralgorithm=””;
String stemmer=””;
String synonymxml=””;

//=”ExactMatchSyntaxAlgorithm” id=”ExactMatchSyntaxAlgorithm” active=”true” weight=”0.1” concreteclass
//two dimensional array syntaxalgorithms[i][0]=name, syntaxalgorithms[i][1]=id, syntaxalgorithms[i][2]=active, syntaxalgorithms[i][3]=concreteclass,
//syntaxalgorithms[i][4]=weight
```

```
//syntaxalgorithms=null;


//String[][] syntaxalgorithms={{"ExactMatchSyntaxAlgorithm","ExactMatchSyntaxAlgorithm","true","ExactMatchSyntaxAlgorithm","1"},
//{"SoundexSyntaxAlgorithm","SoundexSyntaxAlgorithm","true","SoundexSyntaxAlgorithm","1"}};
String[][] syntaxalgorithms={{"ExactMatchSyntaxAlgorithm","ExactMatchSyntaxAlgorithm","true","ExactMatchSyntaxAlgorithm","1"},
{"EditDistanceSyntaxAlgorithm","EditDistanceSyntaxAlgorithm","true","EditDistanceSyntaxAlgorithm","1"},
{"QGramDistanceSyntaxAlgorithm","QGramDistanceSyntaxAlgorithm","true","QGramDistanceSyntaxAlgorithm","1"},
{"SimonWhiteSyntaxAlgorithm","SimonWhiteSyntaxAlgorithm","true","SimonWhiteSyntaxAlgorithm","1"},
{"SoundexSyntaxAlgorithm","SoundexSyntaxAlgorithm","true","SoundexSyntaxAlgorithm","1"}};
this.syntaxalgorithms=syntaxalgorithms;

//searchalgorithm[0][0]=name, searchalgorithm[0][1]=id, searchalgorithm[0][2]=active,searchalgorithm[0][3]=concreteclass
//searchalgorithm=null;
String[][] searchalgorithm={{"MaxSearchAlgorithm","MaxSearchAlgorithm","true","MaxSearchAlgorithm"}};
this.searchalgorithm=searchalgorithm;

//two dimensional array stemmingalgorithms[i][0]=name, stemmingalgorithms[i][1]=id, stemmingalgorithms[i][2]=active, stemmingalgorithms[i][3]=concreteclass
String[][] spellcheckeralgorithms={{"HunspellSpellChecker","HunspellSpellChecker","true","HunspellSpellChecker"}};
this.spellcheckeralgorithms=spellcheckeralgorithms;
```

```
//two dimensional array stemmingalgorithms[i][0]=name, stemmingalgorithms[i][1]=id, stemmingalgorithms[i][2]=active, stemmingalgorithms[i][3]=concreteclass
String[][] stemmingalgorithms={{"PaiceStemmingAlgorithm","PaiceStemmingAlgorithm","true","PaiceStemmingAlgorithm"}};
this.stemmingalgorithms=stemmingalgorithms;

threshold=0.6;

}

}
```

```
package uk.ac.brunel.genericlabelmatcher;
/**
 * @author      Ambikesh Jayal, School of IS, Computing & Maths, Brunel University.
 * @author      ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version     1.0,  25-Aug-2009
 * @since JDK1.6
 */

public class ToolConstants
{
        public static String STR_SPECIAL_CHARACTER="!,\",  ,$,%,^,&,*,(,),-,_,+,=,{,},[,],~,#,;,:,@,',<,>,.,?,/,`,\\,|";

        //removed means to be replaced by no spaces. Like "today's" should be changed to "todays" and not to "today s"
        public static String STR_SPECIAL_CHARACTER_TO_BE_REMOVED="'";

        public static String SEPARATER1=",";
        public static String REPLACEMENT_CHARACTER1=" ";
        //public static String SPACE=" ";
        public static String SPACE="\\s+";
        public static String NOSPACE="";

        public static String STRING_PROCESSING_LEVEL_ONE="Level1";
        public static String STRING_PROCESSING_LEVEL_TWO="Level2";
        public static String STRING_PROCESSING_LEVEL_THREE="Level3";

        //added on 10 march 2008
        public static String STR_ENGLISH_LANGUAGE_ARTICLES="a,an,the,this
            ,that";
        //public static String STR_ENGLISH_LANGUAGE_GRAMMER_WORDS_LIST1="
            to,of,for ,from,and,be,or,on,if ,in,with,by,as,but,at";
        //remove the word 'if' from the list
            STR_ENGLISH_LANGUAGE_GRAMMER_WORDS_LIST1
        public static String STR_ENGLISH_LANGUAGE_GRAMMER_WORDS_LIST1="to
            ,of,for ,from,and,be,or,on,in,with,by,as,but,at";
        public static String STR_ENGLISH_LANGUAGE_GRAMMER_WORDS_LIST2="i,
            he,she,it ,we,you,they";
        public static String
            STR_ENGLISH_LANGUAGE_GRAMMER_WORDS_HELPING_VERBS="is ,am,are,
            do,did ,was,were,has,have,been";

        public static String STR_PUNCTUATION_WORD=
            STR_ENGLISH_LANGUAGE_ARTICLES+ "," +
            STR_ENGLISH_LANGUAGE_GRAMMER_WORDS_LIST1+ "," +
            STR_ENGLISH_LANGUAGE_GRAMMER_WORDS_LIST2+ "," +
            STR_ENGLISH_LANGUAGE_GRAMMER_WORDS_HELPING_VERBS;

}
```

```
package uk.ac.brunel.test;
/**
```

```java
 * @author       Ambikesh Jayal, School of IS, Computing & Maths, Brunel University.
 * @author       ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version      1.0,  25-Aug-2009
 * @since JDK1.6
 */

import uk.ac.brunel.genericlabelmatcher.*;

public class TestClass1 {

public static void main(String[] args){
    System.out.println("Hellooooo ");
    String strNameOfConcreteClass="uk.ac.brunel.genericlabelmatcher.LabelMatcher";
    try{
    //LabelMatcherIF labelMatcherIF=(LabelMatcherIF)Class.forName(strNameOfConcreteClass).newInstance();
    //labelMatcherIF.loadConfiguration("");

    LabelMatcher labelMatcher=LabelMatcher.getInstance("");

    //String[] arrayStr1={"update","total"};
    //String[] arrayStr2={"up","amount","to", "updat"};

    String[] arrayStr1={"Select Recipe", "Assemble Ingredients", "Cook meal", "Set the table", "Eat"};

    //String[] arrayStr2={"turn up at arranged time", "eat the meal produced by friend", "must select a recipe for their meal",
    //"assemble the ingredient", "cook the meal", "Set table", "eat meal"};

```

```
        String[] arrayStr2={"turn up at arranged time", "eat the meal
            produced by friend", "must select a recipe for their meal",
        "assemble the ingredient", "cook the meal", "Set table", "eat meal",
            "recip selct"};

        double[][] arrayDetailsOfMatchedStringsFromSecondArray=labelMatcher.
            getDetailsOfMatchedStringsFromSecondArray(arrayStr1, arrayStr2);
        for(int i=0;i<arrayDetailsOfMatchedStringsFromSecondArray.length;i++)
            {
             System.out.println(arrayDetailsOfMatchedStringsFromSecondArray[i
                 ][0] + "    " + arrayDetailsOfMatchedStringsFromSecondArray[i
                 ][1] + "    " +arrayDetailsOfMatchedStringsFromSecondArray[i
                 ][2] );
        }

        }catch(Exception ex){
            System.out.println("Sorry, other exception");
            ex.printStackTrace();
        }
    }


    }
```

7.4 Java Source code for GenericLabelMatcherConcreteClasses.jar

This component has the concrete implementations of the algorithms used by the GenericLabelMatcher component.

```
package uk.ac.brunel.concreteclasses;
/**
 * @author      Ambikesh Jayal, School of IS, Computing & Maths, Brunel
     University.
 * @author      ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version      1.0,  25-Aug-2009
 * @since JDK1.6
 */

import uk.ac.shef.wit.simmetrics.similaritymetrics.Levenshtein;
import uk.ac.brunel.iface.SyntaxAlgorithmIF;

public class EditDistanceSyntaxAlgorithm implements SyntaxAlgorithmIF{
public double similarity(String str1, String str2){
double similarityIndxex=0.0;
//use simmetrics to return the edit distance
//Note: The concrete class levenshtein.java is present in the package uk.ac.shef.wit.simmetrics.similaritymetrics.Levenshtein
Levenshtein levenshtein=new Levenshtein();
similarityIndxex=levenshtein.getSimilarity(str1,str2);
return similarityIndxex;
}
}
```

```
package uk.ac.brunel.concreteclasses;
/**
 * @author      Ambikesh Jayal, School of IS, Computing & Maths, Brunel University.
 * @author      ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version      1.0,  25-Aug-2009
 * @since JDK1.6
 */

import uk.ac.brunel.iface.SyntaxAlgorithmIF;

public class ExactMatchSyntaxAlgorithm implements SyntaxAlgorithmIF{
public double similarity(String str1, String str2){
double similarityIndxex=0.0;
```

```java
 if(str1.equals(str2)){
         similarityIndxex= 1.0;
    }else{
         similarityIndxex  =0.0;
    }
return similarityIndxex;
}
}
```

```java
package uk.ac.brunel.concreteclasses;

/**
 * This java file uses the Open source JNA based Java API for Hunspell which is available from http://hunspell.sourceforge.net/
 * Hunspell is used by Open Office
 * For details about JNA see https://jna.dev.java.net/

 * @author      Ambikesh Jayal, School of IS, Computing & Maths, Brunel University.
 * @author      ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version     1.0,  25-Aug-2009
 * @since JDK1.6
 */


import com.stibocatalog.hunspell.Hunspell;
import java.util.List;
import java.util.StringTokenizer;
import uk.ac.brunel.genericlabelmatcher.ToolConfiguration;
import uk.ac.brunel.iface.SpellCheckerIF;

public class HunspellSpellChecker implements SpellCheckerIF{
Hunspell.Dictionary d=null;
```

```
    public HunspellSpellChecker(){
        try{
    String strEnglishDictionaryBaseFileName=ToolConfiguration.strEnglishDictionaryBaseFileName;
    d = Hunspell.getInstance().getDictionary(strEnglishDictionaryBaseFileName);
        }catch(Exception ex){
            ex.printStackTrace();
        }
    }
@Override
/*
 * This fucntion returns true if word has been misspelled otherwise false.
 **/
public boolean isMisspelled(String strWord){
    return d.misspelled(strWord);
}

@Override
//inputText can be a sentence consisting of words separated by single or multiple space
public String getSpellCheckedText(String inputText){
String outputText="";
//String[] arrayStrWord=inputText.split(" ");
//Note: "\\s+" means one or more empty spaces.
String[] arrayStrWord=inputText.split("\\s+");
String[] arraySpellCorrectedWords=getSpellCheckedText(arrayStrWord);
//now joining
for(int i=0;i<arraySpellCorrectedWords.length;i++){
if(outputText.equals("")){
    outputText=arraySpellCorrectedWords[i];
}else{
outputText=outputText+ " " + arraySpellCorrectedWords[i];
}
}

return outputText;
}

@Override
public String[] getSpellCheckedText(String[] arrayStrWord){
    String[] correctedArray=new String[arrayStrWord.length];
    for(int i=0;i<arrayStrWord.length;i++){
        correctedArray[i]=getFirstSuggestedWord(arrayStrWord[i]);
    }
    return correctedArray;

}
/*
 * This fucntion returns the first suggested word by the spell checker.
     Incase the word in not spelled incorrectly then this function
     returns the same word
 **/
public String getFirstSuggestedWord(String strWord){
    //Note: The path strPathEnglishDictionary should contain two files
        named en_GB.dic and en_GB.aff
    //String strEnglishDictionaryBaseFileName="C:/01Am/01_Res EAss/04-
        Tools/Spell Checker/Dictionaries/en_GB/en_GB";
    //String strEnglishDictionaryBaseFileName=ToolConfiguration.
        strEnglishDictionaryBaseFileName;
    //Hunspell.Dictionary d = Hunspell.getInstance().getDictionary(
        strEnglishDictionaryBaseFileName);
      return getFirstSuggestedLabelByHunspell(d,strWord);
}
```

```

//Label consists to more than one word
       public static String getFirstSuggestedLabelByHunspell(Hunspell.
           Dictionary d, String strLabel){
               String strFirstSuggestLabel=”NULL”;
               String[] arrWord=strLabel.split(” ”);
               for(int i=0;i<arrWord.length;i++){
                   if(i==0){
                       strFirstSuggestLabel=
                           getFirstSuggestedWordByHunspell(d,arrWord[i]);
                   }else{
                   strFirstSuggestLabel=strFirstSuggestLabel+” ” +
                       getFirstSuggestedWordByHunspell(d,arrWord[i]);
                   }
               }
               //System.out.println(strFirstSuggestLabel);
               return strFirstSuggestLabel;

       }

      public static String getFirstSuggestedWordByHunspell(Hunspell.
          Dictionary d, String strWord){
               String strFirstSuggestWord=”NULL”;

               if (d.misspelled(strWord)) {
               List<String> listSuggestedWords=d.suggest(strWord);
                if(listSuggestedWords!=null && listSuggestedWords.size()
                   >0){
                    //alwasy get the first suggested word
                    strFirstSuggestWord=listSuggestedWords.get(0);
                }else{
                    //strFirstSuggestWord=”NO_SUGGESTION”;
                    strFirstSuggestWord=strWord;
                }
               }else{
                   //strFirstSuggestWord="WORD_IS_CORRECT";
                   strFirstSuggestWord=strWord;
               }
               return strFirstSuggestWord;

       }

public static void main(String[] args){
   String inputText = "cappped    update    Updat         updatetotal
       updatotal tota Udate";
   //String inputText = "cappped update Updat updatetotal updatotal tota
        Udate";
   //String words[] = {"cappped","update", "Updat", "updatetotal", "
       updatotal", "tota", "Udate"};
   HunspellSpellChecker hunspellSpellChecker=new HunspellSpellChecker();
   //String correctedWords[]=hunspellSpellChecker.getSpellCheckedText(
       words);
   //for(int i=0;i<correctedWords.length;i++){
       //System.out.println(correctedWords[i]);
   //}

   System.out.println(hunspellSpellChecker.getSpellCheckedText(inputText
       ));
}

}
```

```
package uk.ac.brunel.concreteclasses;



/**
```

```
 * @author       Ambikesh Jayal, School of IS, Computing & Maths, Brunel University.
 * @author       ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version      1.0,  25-Aug-2009
 * @since JDK1.6
 * This java file uses the Open source JNA based Java API for Hunspell which is available from http://hunspell.sourceforge.net/
 * Hunspell is used by Open Office
 * For details about JNA see https://jna.dev.java.net/
 */


import uk.ac.brunel.iface.SearchAlgorithmIF;

public class MaxSearchAlgorithm implements SearchAlgorithmIF{
/*
 * syntaxalgorithms[i][0] contains the id of the syntax algorithm class
 * syntaxalgorithms[i][1] contains the double value of the similarity index for the corresponding syntax algorithm in the syntaxalgorithms[i][0]
 **/

public double getCombinedSimilarity(String[][] arraySyntaxSimilarityIndex){
        //returns the maximum value
        double combinedSimilarity=-1.0;
        for(int i=0;i<arraySyntaxSimilarityIndex.length;i++){
                double simindex=-1.0;
                try{
                simindex=Double.parseDouble(arraySyntaxSimilarityIndex[i][1]);
                }catch(Exception ex){
                        ex.printStackTrace();
                        }
                if(simindex>combinedSimilarity){
                        combinedSimilarity=simindex;
                        }
        }
        return combinedSimilarity;
        }
}
```

```
package uk.ac.brunel.concreteclasses;
/**
 * @author       Ambikesh Jayal, School of IS, Computing & Maths, Brunel University.
 * @author       ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version      1.0,  25-Aug-2009
 * @since JDK1.6
 * This class uses the PaiceJava2.java class written by Christopher O'Neill
 */

import uk.ac.brunel.genericlabelmatcher.*;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.StringTokenizer;
import uk.ac.brunel.iface.StemmingAlgorithmIF;

public class PaiceStemmingAlgorithm implements StemmingAlgorithmIF{



    //the idea is that keep all the words and their  stems in a map so that it can be reused.
    //key=word, vlaue=stem of the word
    public static Map mapWordAndStem=new LinkedHashMap();
    //strWord should be a single word
    private static String getStemWord(String strWord){
        String strStemWord="";
        if(mapWordAndStem.containsKey(strWord)){
            strStemWord=(String)mapWordAndStem.get(strWord);
        }else{

            //To do: write code for integrating different stemmers like potter. The lecturer should be able to mention the
            //stemmer to use in a properties file.
            //for now just use Paice algorithm

            PaiceJava2 paiceJava2=PaiceJava2.getSingletonInstance();
            strStemWord=paiceJava2.stripAffixes(strWord);


            //add to the map for reuse later
            mapWordAndStem.put(strWord,strStemWord);
        }
        return strStemWord;
    }


//inputText can be a sentence consisting of words separated by single or multiple space
public String getStemText(String inputText){

        //PaiceJava2 p = new PaiceJava2(args[2],args[3]);
        //PaiceJava2 p = new PaiceJava2();
        StringTokenizer line = new StringTokenizer("");
        String outputText ="";
    line= new StringTokenizer(inputText);
    try{
        while (line.hasMoreTokens())
            {
            // read word from line and stem word
            String word = new String();
            word = line.nextToken();
            if(outputText.equals(””)){
            outputText=getStemWord(word);
            }else{
            outputText=outputText+ ” ” + getStemWord(word);
            }


            }
        }
    catch(Exception e)
        {
        e.printStackTrace();
        }
return outputText;


    }



}
```

```
package uk.ac.brunel.concreteclasses;
/**
 * @author      Ambikesh Jayal, School of IS, Computing & Maths, Brunel University.
 * @author      ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version     1.0,  25-Aug-2009
 * @since JDK1.6
 */

import uk.ac.brunel.iface.SyntaxAlgorithmIF;

import uk.ac.shef.wit.simmetrics.similaritymetrics.QGramsDistance;

public class QGramDistanceSyntaxAlgorithm implements SyntaxAlgorithmIF{
public double similarity(String str1, String str2){
double similarityIndxex =0.0;
//use simmetrics to return the edit distance
//Note: The concrete class QGramsDistance.java is present in the package uk.ac.shef.wit.simmetrics.similaritymetrics.Levenshtein
QGramsDistance qGramsDistance=new QGramsDistance();
similarityIndxex=qGramsDistance.getSimilarity(str1,str2);
return similarityIndxex;
}
}
```

```
package uk.ac.brunel.concreteclasses;
/**
 * @author      Ambikesh Jayal, School of IS, Computing & Maths, Brunel University.
 * @author      ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version     1.0,  25-Aug-2009
 * @since JDK1.6
 */

import uk.ac.brunel.iface.SyntaxAlgorithmIF;

public class SimonWhiteSyntaxAlgorithm implements SyntaxAlgorithmIF{
public double similarity(String str1, String str2){
double similarityIndxex =0.0;
//use the algorithm developed by Simon White to return the similarity, http://www.catalysoft.com/articles/StrikeAMatch.html
SimonWhiteStringMatchingAlgorithm simonWhiteStringMatchingAlgorithm=new SimonWhiteStringMatchingAlgorithm();
similarityIndxex=simonWhiteStringMatchingAlgorithm.getSimilarity(str1, str2);
return similarityIndxex;
}
}
```

```
package uk.ac.brunel.concreteclasses;
/**
 * This is a simple algorithm for exactly matching two strings. It returns 1 if two strings exactly match otherwise returs 0.
 * @author      Ambikesh Jayal, School of IS, Computing & Maths, Brunel University.
 * @author      ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version     1.0,  25-Aug-2009
 * @since JDK1.6
 */

import java.util.*;
import uk.ac.shef.wit.simmetrics.similaritymetrics.InterfaceStringMetric;

public class SoundexMatchAlgorithm implements InterfaceStringMetric {



    public String getLongDescriptionString(){
    //returns a long string of the string metric description.
        return "This is a Soundex algorithm for matching two strings. It returns 1 if two strings have exactly same Soundex code, otherwise returs 0. " +
               "It uses org.apache.commons.codec to retrieve the Soundex
                   code for a string. ";
    }

    public String getShortDescriptionString(){
    //returns a string of the string metric name.
        return "Soundex Matching Algorithm";
    }
    public float getSimilarity(java.lang.String string1, java.lang.String
         string2){
    org.apache.commons.codec.language.Soundex apacheSoundex=new org.
        apache.commons.codec.language.Soundex();
    String soundexCodeForString1=apacheSoundex.soundex(string1);
    String soundexCodeForString2=apacheSoundex.soundex(string2);
    if(soundexCodeForString1.equals(soundexCodeForString2)){
        return 1;
    }else{
        return 0;
    }
    }

   public String getSimilarityExplained(java.lang.String string1, java.
       lang.String string2){
    //returns a similarity measure of the string comparison.
       return "This is a Soundex algorithm for matching two strings. It
           returns 1 if two strings have exactly same Soundex code,
           otherwise returs 0. " +
               "It uses org.apache.commons.codec to retrieve the Soundex
                   code for a string. ";
   }

   public long  getSimilarityTimingActual(java.lang.String string1, java.
       lang.String string2){
```

```
    //gets the actual time in milliseconds it takes to perform a similarity timing.
     return    string1.length()+string2.length();
   }

    public float getSimilarityTimingEstimated(java.lang.String string1, java.lang.String string2){
    //gets the estimated time in milliseconds it takes to perform a similarity timing.
        return    string1.length()+string2.length();
    }



    public static void main(String[] args) {

        InterfaceStringMetric interfaceStringMetric=new SoundexMatchAlgorithm();
        System.out.println(”*****”+interfaceStringMetric.getSimilarity(”update”, ”updat”));
     }
}
```

```
package uk.ac.brunel.concreteclasses;
/**
 * This is a simple algorithm for exactly matching two strings. It returns 1 if two strings exactly match otherwise returs 0.
 * @author      Ambikesh Jayal, School of IS, Computing & Maths, Brunel University.
 * @author      ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version     1.0,  25−Aug−2009
 * @since JDK1.6
 */

```

```
import uk.ac.brunel.iface.SyntaxAlgorithmIF;

public class SoundexSyntaxAlgorithm implements SyntaxAlgorithmIF{
public double similarity(String str1, String str2){
double similarityIndxex=0.0;
//use simmetrics to return the edit distance
//Note: The concrete class SoundexMatchAlgorithm.java is present in the
    package uk.ac.brunel.mamcaasystem.common.utility.syntactic.*
SoundexMatchAlgorithm soundexMatchAlgorithm=new SoundexMatchAlgorithm();
similarityIndxex=soundexMatchAlgorithm.getSimilarity(str1,str2);
return similarityIndxex;
}
}
```

7.5 Java Source code for GenericLabelMatcherInterface.jar

This component has the interface for the algorithms used by the GenericLabelMatcher component. The user needs this component containing interfaces in order to compile the concrete implementation of the algorithms.

```
package uk.ac.brunel.iface;
/**
 * @author       Ambikesh Jayal, School of IS, Computing & Maths, Brunel
     University.
 * @author       ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version      1.0,  25-Aug-2009
 * @since JDK1.6
 */

public interface SearchAlgorithmIF{

/*
 * syntaxalgorithms[i][0] contains the id of the syntax algorithm class
 * syntaxalgorithms[i][1] contains the double value of the similarity index for the corresponding syntax algorithm in the syntaxalgorithms[i][0]
 **/

public double getCombinedSimilarity(String[][] arraySyntaxSimilarityIndex);
}
```

```
package uk.ac.brunel.iface;
/**
 * @author       Ambikesh Jayal, School of IS, Computing & Maths, Brunel University.
 * @author       ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version      1.0,  25-Aug-2009
 * @since JDK1.6
 */

public interface SpellCheckerIF{

/*
 * This fucntion returns true if word has been misspelled otherwise false.
 **/
public boolean isMisspelled(String strWord);

/*
 * This function takes a single word or a sentence consisting of words separated by single or multiple space.
 * It then returns the autocottected version of each word,
 * This fucntion returns the first suggested word by the spell checker.
 * Incase the word in not spelled incorrectly then this function returns the same word
 **/
 public String getSpellCheckedText(String strWord);

 /*
  * * This function takes an array of single words.
 * It then returns an array the autocottected version of each word,
 * This fucntion returns the first suggested word by the spell checker. Incase the word in not spelled incorrectly then this function returns the same word
 **/
 public String[] getSpellCheckedText(String[] strWord);

}
```

```
package uk.ac.brunel.iface;

/**
 * @author      Ambikesh Jayal, School of IS, Computing & Maths, Brunel University.
 * @author      ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version     1.0,  25-Aug-2009
 * @since JDK1.6
 */
public interface StemmingAlgorithmIF{



 /*
This function takes a single word or a sentence consisting of words separated by single or multiple space. It then returns the stem of each word,
//inputText can wither be a single word or be a sentence consisting of words separated by single or multiple space
*/
public String getStemText(String inputText);


}
```

```
package uk.ac.brunel.iface;
/**
 * @author      Ambikesh Jayal, School of IS, Computing & Maths, Brunel
      University.
 * @author      ambikesh.jayal@brunel.ac.uk, ambi1999@gmail.com
 * @version     1.0,  25-Aug-2009
 * @since JDK1.6
 */

public interface SyntaxAlgorithmIF{
public double similarity(String str1, String str2);
}
```