Visualising Qualitative Spatial Representations

Raby Whyte

BSc Computing with Artificial Intelligence
(Industrial)
Session 2008/2009

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student)________________________
Summary

This project attempts to aid students and practitioners alike with developing qualitatively described scenes.

Much work has been done in converting an image into a qualitatively described scene. This is generally achieved by splitting the image up into regions, then applying qualitative reasoning to describing the position of each region in respect to one another. However not much work has been done in the opposite, converting qualitative spatial representations into quantitative data, which computers can use to visualise scenes.

To aid students and practitioners it was decided to achieve this in two ways. Checking a qualitatively described scene for consistency, and visualising the qualitatively described scene.
Acknowledgements

Most of all I’d like to thank my project supervisor Professor Tony Cohn for his continued patience, pointers in the right direction and friendly meetings. I would like to thank my assessor Dr Brandon Bennett, for his constructive comments during the progress meeting.

Finally I want to thank many others for there support during the project including Lee Tunnicliffe for his listening ear, Rhiannon Adams for her stern motivation and to Matthew Revell for help with \LaTeX.
## Contents

1 Project Introduction ................................................. 1

1.1 Problem Definition .............................................. 1

1.2 Project Aim and Objectives .................................... 2

1.2.1 Minimum Requirements ......................................... 2

1.2.2 Possible Enhancements .......................................... 3

1.2.3 Deliverables ...................................................... 3

1.3 Relevance to the degree ........................................... 3

1.4 Project Plan ....................................................... 3

1.4.1 Characteristics of the Project ................................ 3

1.4.2 Chosen Methodology ............................................ 4

1.4.3 Project Schedule and Overview ................................ 5

2 Background Research ................................................. 6

2.1 Introduction ....................................................... 6

2.2 Qualitative Spatial Representation ............................. 6

2.2.1 Applications of Qualitative Spatial Representation ....... 7

2.2.2 Region Connection Calculus ................................. 7

2.2.3 Other Notable Calculi .......................................... 9

2.2.4 Reasoning ....................................................... 11

2.3 Visualising Spatial Relationships ............................... 12

3 Preparation and Design of the Solution ........................ 14

3.1 Implementation Plan ............................................ 14

3.1.1 Iteration One .................................................... 14
3.1.2 Iteration Two .................................................. 15
3.2 Testing and Evaluation Plan ........................................ 15
  3.2.1 Iteration One .............................................. 15
  3.2.2 Iteration Two ............................................. 16
3.3 Software Architecture ........................................... 17
3.4 Design Issues ................................................... 22
  3.4.1 Software Architecture with Embedded Algorithms .......... 23
  3.4.2 Presentation Layer Application .............................. 24
  3.4.3 Application and Programming Environment .................. 25
3.5 Conclusions ....................................................... 26

4 Delivery of the Solution ........................................... 27
  4.1 Iteration One: Architecture and Algorithms ...................... 27
    4.1.1 Checking Correctness of a Qualitative Scene Algorithm Design .... 27
    4.1.2 1-Dimension Visualisation Algorithm Design .................. 30
    4.1.3 Implementation of the Algorithms ........................... 32
    4.1.4 Testing .................................................. 33
    4.1.5 Conclusion ............................................... 35
  4.2 Iteration Two: Application ...................................... 36
    4.2.1 Improved 1-Dimension Visualisation Algorithm Design .......... 36
    4.2.2 Improvements to the ‘Checker’ Algorithm Design ................ 37
    4.2.3 Web Application .......................................... 38
    4.2.4 2-Dimension Visualisation Algorithm Design .................. 40
    4.2.5 Providing the User with Alternative Correct Visualisations .... 41
    4.2.6 Testing and User Evaluation ................................ 42
    4.2.7 Conclusion ............................................... 43

5 Evaluation and Conclusions ....................................... 44
  5.1 Evaluation ..................................................... 44
    5.1.1 Evaluation Criteria ...................................... 44
    5.1.2 Evaluation using Evaluation Criteria ......................... 45
      5.1.2.1 Minimum Requirements and Extensions .................... 45
Chapter 1

Project Introduction

1.1 Problem Definition

Qualitative spatial descriptions are used by us on a daily basis. We generally naturally describe objects attributes and position in space using qualitative natural language. For example ‘the kettle is to the left of the toaster’. Computers however describe positional information using numbers. A computer may describe the kettle and toaster as a collection of pixels, and these pixels could be described as $n$ number of pixels apart [37].

Mapping human qualitative spatial descriptions into quantitative information that computers can understand has been attempted using several calculi, this area of research is called Qualitative Spatial Representation and Reasoning [17] and is discussed further in Chapter 2. These calculi generally use first-order logic to describe spatial scenes [17] and need all spatial relationships to be described to function correctly.

Notice in the example above the exclusion of what we deem obvious spatial constraints such as ‘The kettle and the toaster are above the floor’ or ‘The kettle and the toaster are in the building’. Thinking of every spatial relationship is a problem when developing spatial algebra to describe a scene. Even a relatively simple scene contains many spatial constraints we take for granted. Missing spatial relationships may produce a description of an unintentional scene, or be incorrect in accordance to the logical rules of the spatial description language.

When presented with a moderately complicated set of rules describing relationships between arbitrary objects, reading the relationships and then visualising the scene in ones head is difficult. This is
especially problematic if a set of spatial relationships is being extended and the user is imagining an incorrect scene compared to the intentional. Several scenes can be described from one qualitative spatial configuration too [19], increasing the problem. Comparing a mathematical and logical mind with a mind tuned for spatial awareness, the thought processes are different [33] theoretically justifying the problem.

To justify the problem a subset of students and ex-students that have studied the Knowledge Representation and Reasoning module were given some questions. This was completed during the initial phase of the project, to ensure the project’s aim and motivation was meaningful. See Appendix D. The results of which motivated the projects continuation.

1.2 Project Aim and Objectives

The aim of this project is to enhance the role of human cognition in the process of developing qualitative spatial descriptions. This will be achieved via the visualisation of a scene when presented with a set of qualitative spatial configurations, and algorithms to check the validity of a set of relationships inputted. Hence solving of the two problems described above by aiding the user in thinking about all relations present within the scene. The algorithms developed will be demonstrated via the development of a prototype application.

The objectives to be achieved by completing this project are as follows:

- Learn more about Qualitative Spatial Representation and the calculi used for defining relationships.
- Design a prototype application and algorithm which checks a set of qualitative relationships for validity.
- Design a prototype application and algorithm which visualises qualitative spatial relationships the user has described.
- Identify the limitations and possible improvements of the algorithms developed to aid in further development.

1.2.1 Minimum Requirements

The minimum requirements meet the core project objectives:

- Design a tool which checks a qualitative spatial configuration for validity.
- Design a tool which enables the visualisation of qualitative spatial configurations.
- An evaluation of the system to aid in further work.
1.2.2 Possible Enhancements

Below are some initial possible enhancements to the solution:

- Design an algorithm which ‘fills-in’ missing relationships within a qualitative description of a scene.
- Design an algorithm which gives an improved visual appeal to the one-dimensional visualisation.
- Design and build an extensible more usable application that interfaces with the algorithms.
- Design an algorithm which gives a more comprehensive true visualisation using two-dimensions.
- Design the tool to be extensible, so it is easy to ‘plug-in’ extra spatial configuration languages at a later date.
- When the algorithm decides on a relationship to add to the configuration it may be one of several choices. A system could be provided that records this and allows a different relationship to be chosen.

1.2.3 Deliverables

My deliverables will be primarily a written report, and secondarily a prototype application.

1.3 Relevance to the degree

Throughout my degree I have vested interest in both Artificial Intelligence and Computer Graphics. The subject of this paper combines these two fields and allows me to extend my knowledge in these areas. A number of modules studied in the past three years which are key relevance to this project are: Algorithm Design; The fundamentals of AI; Software Engineering and Computer Graphics.

1.4 Project Plan

1.4.1 Characteristics of the Project

This project is different compared to most software development projects. Most traditional software development involves non-technical stakeholders and a monetary value of some description. This project only has one developer and it’s purpose is not to develop a finished application that will be deployed and used, but a prototype that will demonstrate a possible approach to solving a problem. The primary result is a software design which is a solution to a problem rather than a ‘fully-fledged’ application.
These project characteristics will need to be taken into account when choosing a project methodology to follow.

### 1.4.2 Chosen Methodology

The project is small, relatively short and has one developer. Therefore bloated software development methodologies such as the V-model [52], Spiral Model[9] and the Waterfall Model[58, 61]) that focus on long-term planning and correct practices being adhered to therefore will be ignored. The agile software project management approach is more favourable to adopt as it is tailored for smaller projects [1] and also reduces the administration that plagues fore mentioned methodologies.

The agile approach however does have its limitations as it focuses primarily on teamwork and this project has just one developer. Test-driven development is another feature of the Agile approach that would be counter-productive for this project, as everything coded doesn’t need to be impenetrable (referring to security) and perfect.

A better approach for this project would be to use the iterative software development process [43] customising it to the specific characteristics of this project. The iterative development approach focuses on building small-parts of the system one at a time, achieving interim milestones each of which mark the end of an iteration. This would be an ideal method for this project as the minimum requirements can be focused on in earlier iterations, whereas enhancements can be achieved in later iterations. This approach has the advantages of the agile model, small project overhead and low maintenance but none of the limitations. It can be easily tailored to the characteristics of this project too.

The initial plan, background research, and the overall design of the system architecture can be
achieved prior to any iteration. Justified because time does not allow for research to be done for no gain other than realising the wrong path has been taken, and the overall design to be discarded and re-started. The only aspects which would be iterated would be the design, implementation and testing of small components that contribute to the whole.

1.4.3 Project Schedule and Overview

The initial schedule of the project has been provided in Appendix B and reflects the methodology chosen above.

This project will start with some relevant background research, see chapter 2. Chapter 3 will then use the information obtained within the background research to detail the preparation and design issues for consideration before starting the implementation phase. Since part of the project is incremental, chapter 4 will be split into several separate sections, each of which detailing the design, implementation and testing of each iteration respectfully. Chapter 5 brings this report to a close with an evaluation and conclusion.
Chapter 2

Background Research

2.1 Introduction

Firstly an exploration of the field of Qualitative Spatial Representation and Reasoning is necessary to provide an overview of the subject. The applications of this area of research is too worth studying briefly to gauge the importance of the subject. The RCC calculus will then be the calculi of primary focus of this project. Due to the relative high adoption within the QSR community, measured by the amount of research carried out to extend the calculus [75, 57, 56, 6, 8, 34, 78, 76, 69]. The users of the solution will be students or practitioners of the subject, therefore the primary representation language should be the most adopted. Other significant representation approaches will be discussed, as the solution is then extensible to other approaches. Research will then focus on previous attempts to solve the problem before, and approaches that may prove worthwhile for further exploration.

A systematic strategy for identifying relevant literature is adopted. This strategy uses two major papers as a base. The search for relevant primary literature is then expanded. Focusing on these authors and related keywords, a combination of Information Retrieval tools are used for this. Secondary sources will be identified through the parsing of relative citations and references present in primary papers.

2.2 Qualitative Spatial Representation

Qualitative Spatial Representation (QSR) and Reasoning provides a manner in which to represent and reason about spatial information using symbolic descriptions. This is achieved from a limited vocabulary
as there is no purely qualitative spatial reasoning mechanism due to the “poverty conjecture” [59, 18].

For example qualitative descriptions may exist to describe relationships between regions, without using numerical computation [54]. Freska states that qualitative spatial knowledge is obtained “by comparing features within the object domain rather than by measuring them in terms of some artificial external scale.”[31]. Qualitative spatial representations of a scene using a higher level of abstraction is in contrast to traditional numerical approaches that dominate the Computer Science field [37].

The challenge of the QSR community over the last 15 years [45] has been “to provide calculi which allow a machine to represent and reason with spatial entities of higher dimension, without resorting to traditional quantitative techniques”[19]. To achieve this, QSR tries to represent and reason about space using extended spatial entities and the relations between them, at the appropriate level of abstraction desired. A wide range of approaches have been put forward by the community using first order logic [13] involving spatial attributes, such as topology [5, 19], orientation [35], shape and distance [34]. Most recent work on qualitative spatial reasoning though has focussed on mereotopological languages. Here primitives are limited to mereological (part whole) and topological relations [17, 15, 36, 55].

2.2.1 Applications of Qualitative Spatial Representation

For some situations, representing a spatial scene with a higher level of abstraction is more akin to the user’s needs [62], therefore more ideal [38]. Hence why interest in this area comes from a wide variety of areas including (this is by no means an exhaustive list): Robot navigation within Artificial Intelligence [32]; Several comprehensive applications in the area of Geographic Information Systems (GIS) [16, 29], and similarly within weather applications [3]; Describing document structures using qualitative techniques [72]; Early stages of designing physical structures for use in Engineering, an area called qualitative kinematics [24]; and Using spatial abstractions in industrial environments for describing networks, especially useful in ubiquitous computing [39].

2.2.2 Region Connection Calculus

The RCC calculus created by Randell, Cui and Cohn [19] in its most general form, provides a topological approach to representing and reasoning about space. This is achieved using first order predicate calculus (FOPC) [13]. Most traditional approaches for representing space use the point as the base entity in which to build upon. A point is an entity that has a location in space or on a plane, but has no extent. Therefore a 0-dimensional object, impossible to visualise in the real world. RCC however uses Regions as the
Figure 2.1: A subsumption lattice of dyadic relations defined in terms of C, taken from [19]

calculus’s base entity. As any real body in space will not be a single dimensionless point, the region is a more intuitive true base representation of a real body [19]. These regions are non empty and regular which means they must all be of an arbitrary uniform dimension. Regions too can be multi pieced rather than just internally connected [19], which is true when thinking about real world logical concepts. For example a university campus may be split geographically.

A primitive relation of connectedness between two regions exists named C(a,b). This reads as ‘a connects with b’. Two regions are said to “connected if and only if their topological closures share at least one point” [19]. The base relation of connectedness is highly expressive, reflexive and symmetric. Cohn et. al. has defined a calculus of different binary relationships between regions using this base relation [17]. See figure 2.1.

The relations P, PP, TPP, and NTTP are non symmetrical and thus have converses Pi, PPi, TPPi and NTTPi. The relation O(a,b) represents an overlap between the two regions. P(a,b) vaguely represents “a being a part of b”. These ‘middle’ relations help define the building blocks of the RCC-8 relations, see figure 2.2. Comparing the visualisation of each relationship in figure 2.2 with the relationships in figure 2.1 the logical foundation of the relationships can be seen.

RCC-8 is a set of 8 base relations, DC (Disconnected), EC (Externally Connected), PO (Partially Overlapping), TPP (Tangential Proper Part), TPPI (The inverse of TPP), NTTP (Non Tangential Proper Part), NTTPi (Inverse of NTTP), and EQ (Equal), respectfully. These are provably Jointly Exhaustive and Pairwise Disjoint (JEPD), which means they are mutually disjoint and provide an exhaustive set of
possible relations figure 2.1.

A pure mereological calculus exists which is simpler to reason with called RCC-5. This representation calculus does not recognise the boundaries of each region as there are no tangential aspects and nothing to denote external connectedness. Five JEPD base relations are presented rather than eight. These are: DR (DiscRete) which is a merger of DC and EC within RCC-8; PO (Partially Overlapping); EQ (EQual); PP (Proper part) and PPi (Inverse of PP) which replaces NTTP as TPP cannot be represented without boundaries.

The egg-yolk calculus can be used to account for boundaries of regions which are indefinite [56, 16]. This is achieved by including a new binary relation to RCC, ‘x is crisper than region y’. The yolk is the minimum extension and the egg is the maximum extension, making the white the area of indeterminacy [8, 19]. This approach was useful for modelling oil spills [16] amongst many other applications.

A conceptual neighbourhood graph, or a continuity graph, can be built from spatial relations belonging to a calculi. A key concept in qualitative reasoning is that change over time is continuous. A conceptual neighbourhood demonstrates possible bi-directional direct topological transitions between relations in both directions. For example, the RCC-5 conceptual neighbourhood shows that PP(a,b) cannot occur from DC(a,b), without PO happening in-between. This makes sense in the ‘real world’. RCC relations within the conceptual neighbourhood assumes continuous translation of shape deformations and movement [19, 17].

RCC has been extended for many applications and purposes since it’s original introduction into the QSR community by Randell et al. [53]. Including the adding of distance [34], and orientation [14]. RCC-11 has also been proposed [78].

2.2.3 Other Notable Calculi

Randell et al. justified that disregarding the distinction between open and closed regions when proposing the RCC Calculus [53]. This was justified, as regions whether they are closed or open occupy the same
amount of space, and therefore pose “deep conceptual problems’ [19]. Open and closed regions were used in one of the original qualitative spatial theories by Clarke [15]. Separately Egenhofer developed an approach to topological relations using nine possible intersections between the interior, exterior and boundaries of two spatial entities. This makes it possible to use entities of different dimensions or distinguish different degrees of intersection [54].

Introduced in 1983, Allen’s Interval Algebra [2] is a calculus developed for temporal reasoning. The calculus defines possible relations between time intervals. A transitivity table (later coined composition table [19]. See figure 2.4) can be used as a basis for reasoning about temporal descriptions of events. Together with a converse operation, Allen’s Interval Algebra can be used as a relational algebra describing spatial configurations. Relations are interpreted by describing the relative position of spatial objects. This also has been adopted for three-dimensional objects by listing the relation for each coordinate separately [2]. See figure 2.3.

A different approach to the qualitative representation of scenes breaks away from the use of regions. Sumitomo and Takahashi justified this by stating the abstraction provided by using only positional information on the relations of objects is too coarse and insufficient information is being given for practical problems [66, 63]. A combination of four primitive symbolic representations have been proposed. Points are defined as a primitive class. Lines have an inherent orientation and connect two points, they are not allowed to cross. A sequence of two or more lines is known as a circuit. Lines and points within circuits can be shared, and can be viewed as a boundary between areas. An area consists of a sequence of two or more circuits. Therefore an area is a one piece connected region. The primitive parts make up the name of this calculi, PLCA.

Powerful operations on scenes described with PLCA have also been defined [66, 63]. For example the integration or division of areas. Combinations of these operations can be used to change areas, similar to the conceptual neighbourhood defined in RCC. Attributed PLCA introduces a new object

![Figure 2.3: Allen’s thirteen interval-interval relations, taken from [19]](image-url)
Region which can be used to describe an object in space and is made up of. This is similar to a region within the RCC. Therefore it is possible to treat both calculi similarly in visualisation. The combination of operations and attributes can result in the dynamic changing of visualising a spatial scene. The example presented by Sumitomo and Takahashi [66, 63] showed a birds-eye view of a tree over the ground, how the scene would naturally look. This was changed to the floor being higher than the tree.

Several extensions to RCC exist which propose to represent both discrete and continuous objects. Wang et al. proposed Multi-dimensional Spatial Occlusion (MSO), a multidimensional extension of RCC suitable for both real and abstract objects [75]. MSO introduces points and lines which are common in real-world applications. These points and lines are combined with the region primitive proposed in the RCC [19]. Occlusion is also represented by the introduction of a 3-dimensional viewpoint, via a similar method used by the Occlusion Calculus (OCC) [40]. A Composition table has been proposed for reasoning combining MRCC (Multi dimensional Region Connected Calculus [77]), MSO and the occlusion metrics making reasoning complex.

Cohn, experienced in QSR community, stated that “it is unlikely that a single universal spatial representation language will emerge” [18]. It is most likely a library of different languages will be used in different situations.

2.2.4 Reasoning

An important form of qualitative reasoning is based on the composition table [19]. A composition table is based on a JEPD set of relations from a QSR calculi, see figure 2.4. The table can be represented as an n x n matrix, where n equals the number of JEPD relationships in the calculi. The logical properties of the relations R1(a,b) and R2(b,c) are used in compositional deduction to infer the possible relations of R3(a,c). The entries in the table are often disjunctive relations because of the qualitative nature of the calculus. Composition tables for RCC have been generated to provide a basic reasoning mechanism for RCC. Each cell within the matrix corresponds to a theorem [53]. Using a composition table can
be a simple method of reasoning for limited binary relations, but it is hard to determine which theories are "complete’. Therefore automatically constructing a composition table can be complex. Generally propositional logic [13] is used for this task [19].

Relations can be checked for consistency by a method known as ‘triangle checking’ [19]. For example in the composition table the relationship between regions a and b, and the relationship between regions b and c can be used to find a list of all possible relations that could be present between regions a and c. The triangle here is non-cyclic, therefore representing several triangles can be possible using a directed acrylic graph (DAG). A DAG is a directed graph with no directed cycles and has a linear ordering of its nodes. This means each node comes before all nodes to which it has outbound edges [68]. Each node of the DAG could represent a region, and each edge represent a JEPD relation. As every region has a relation with every other region within a scene, the DAG would also be a clique. This is a good way to represent a scene due to the graph's scalability. DAG’s can be represented on computers in many ways, using structured objects, as ASCII or as a matrix [64]. Using a composition table, triangle checking can then be achieved by parsing the DAG. This means a qualitative spatial scene can be checked for validity.

The SparQ toolbox [21, 74, 73] is a tool which attempts to make standard reasoning techniques available to users outside of the QSR field. SparQ also provides the functionality to convert between quantitative spatial information to qualitative spatial information. The reasoning techniques provided by this tool kit could be explored further.

2.3 Visualising Spatial Relationships

Resulting from an extensive literature search, the visualisation of spatial relationships has rarely been attempted to date. The most notable attempts found were Kumokawa and Takahashi’s visualisation of PLCA expressions [42, 66, 63], and Wiebrocks et. al’s converting of qualitative scenes into a quantitative representation for visualisation [64]. Kumokawa and Takahashi suggested an “algorithm drawing RCC has not been attempted before” [42], therefore finding another application for comparison is improbable.

The PLCA visualisation approach transformed a PLCA expression into a graph [42]. Once a graph exists which represents points and lines of a planar PLCA expression, the graph is split into connected subgraphs. Each subgraph is then independently drawn as a series of straight lines. The location and size of each drawn subgraph is then determined by circuits and areas within the PLCA expression. This
approach is not directly possible with topologically based calculi’s such as RCC. This is due to the fundamental differences in the languages, as Kumokawa stated “PLCA is a more refined classification for equivalent figures than the other QSR systems” [42].

Wiebrocks matrix visualisation approach constructs a labelled directed graph with symbolic and numeric constraints [64]. Relations between pairs of objects are represented by transformation matrices. The matrices are integrated as part of the graph, which is parsed during visualisation. This method though uses qualitative descriptions of the scene containing positional information, and using this information calculates the appropriate transformation matrix. For example “The fridge stands at the wall’ and “The fridge stands to the right of Stefanie’[64]. As less constraints exist in RCC, positional information is excluded making this approach unsuitable.

Alan Borning’s ThingLab, a constraint-oriented simulation laboratory [10, 11] attempts to visualise sets of defined constraints. Example constraints include ‘All lines should be the same thickness’ or ‘The kettle should always be in the kitchen’. These would be visualised by introducing objects into the world, and the constraints would then be used to determine these objects location’s in space.

Graph drawing, a branch of graph theory was explored. This area however focuses on the direct two-dimensional representation of graphs. The visualisation of nodes, edges and attributes, respectfully. If the nodes of a graph represented regions, and edges represented relationships technically this could be a visualisation of a qualitative spatial scene. This however was not the approach originally in mind, as it wouldn’t be useful from the user’s perspective. However parsing the graph using a topological sorting algorithm [67] may prove useful. Although like-for-like systems do not exist, the fundamental ideas explored in the above approaches are useful for designing an algorithm and application.
Chapter 3

Preparation and Design of the Solution

Using the project schedule and methodology discussed previously, a low level implementation plan is constructed which meets all minimum and extension requirements. Software architecture will be presented incorporating design issues which need to be addressed. The appropriate environment and tools which will be used for the solution are discussed.

3.1 Implementation Plan

The development of the solution will be split into the following two iterations. The end of each iteration will see some testing followed by an evaluation. The results of which will be consolidated and discussed within the evaluation chapter.

3.1.1 Iteration One

This iteration focuses on meeting the minimum requirements.

- Design an algorithm which checks the validity of a qualitative spatial configuration. \((Minimum requirement1)\)
- Implement and test the algorithm. \((Minimum requirement1)\)
- Design an algorithm which visualises a qualitative spatial configuration in 1-dimension. \((Minimum requirement 2)\)
- Implement and test the algorithm. \((Minimum requirement 2)\)
• Implement the designed software architecture. *(Extension 5)*

• Integrate the algorithms into the software architecture. *(Extension 5)*

**Test and Evaluate**

### 3.1.2 Iteration Two

This iteration addresses many possible extensions to the minimum requirements, improving the overall quality of the solution.

• Expand the design and implementation of the ’checking’ algorithm to ‘fill-in’ missing relationships. *(Extension 1)*

• Improve the implementation of the visualisation algorithm defined in iteration one to provide the user with a more appealing visualisation. *(Extension 2)*

• Design and build a more usable application which uses the software architecture. *(Extension 3)*

• Design an algorithm which visualises a qualitative spatial configuration in 2-dimensions. *(Extension 4)*

• Implement this algorithm, then integrate it into the application. *(Extension 4)*

• Design a system which allows the user to change a visualisation of a qualitative spatial visualisation, to another valid visualisation. *(Extension 6)*

**Test and Evaluate**

### 3.2 Testing and Evaluation Plan

Here follows a description of the testing which will be performed after each iteration of development. The evaluation chapter will use the results of this to gauge how successful the project was in solving the problem.

#### 3.2.1 Iteration One

Black-box testing will comprise of inputting a collection of different qualitative spatial configurations, some of which satisfy boundary conditions, into both algorithms. This method of testing was chosen over white-box testing because here we are focusing on the specification or functionality of an application rather than finding bugs [4]. White-box testing will be replaced by debugging which will be carried out during the development.
An analysis of any differences between the desired and the actual output will be completed. The number of tests processed as expected will be converted into a percentage metric. The evaluation of the success of this iteration will be achieved by comparing this metric against a desired metric.

### 3.2.2 Iteration Two

The problem this project is attempting to solve is related to human users creating qualitative spatial scenes. The result of this iteration will be a usable application. Therefore it is necessary to test the performance of the application resulting from this iteration via the involvement of users.

Three main methods are commonly used to carry out user based evaluations. These are user interviews, questionnaires and observation [23]. User observations and interviews require much more time to complete than questionnaires. Due to the time constraints of this project it was decided that questionnaires would be the best approach for user evaluation.

A sample of users will test the application under observation. Preece et al [51] explains how “observation will be useful in the early stages of a software project to gain more understanding of the users needs”, and in the later stages to “determine if these needs have been met”. Observation can be adapted here for both reasons that Preece has identified. One, using observation for evaluating if the needs of the user have been met. Two, adapting the users needs into iteration three if time allows, or document them to aid future extensions to this work. During observation, any errors or user comments will be recorded as qualitative information.

A questionnaire will be presented which asks the user how they would rank the application for usability, and ask if they think the application meets the requirements. Wai-Ching remarks that “questions should progress from the open and general...to the detailed and closed” [44]. Following this suggestion, the questionnaire will begin with open qualitative based questions to gauge the overall opinion of the application. After then some quantitative questions will be asked to gather more structured feedback, as this is more easily adapted for evaluation purposes. The user will be asked to rate the application on some usability attributes. Lastly the questionnaire will gather demographics to aid in understanding the results (as suggested by [51]), to see how competent the user is with the understanding of knowledge representation and reasoning as a topic, and with web application use in general.

The improved algorithms will be tested by the developer using black-box testing. The same scenes used in previous testing, and some additional incomplete scenes will be used.
3.3 Software Architecture

The definition of Software Architecture is different depending on the scenario. Bernard F. Witts’ definition of Software Architecture stating the “structure of a program or computing system, which comprise software components, the externally visible properties of those components, and the relationships between them.” [7] which will be the definition adopted here.

The application is relatively small compared to most enterprise software. The methodology of the project also trims the software life cycle. For this reason all architecture principles won’t be followed, only ones which add value. Traditionally architecture description languages (ADL) are used, and different views on the software are presented for different stakeholders. The architecture documentation here will consist of three elements: this section of the report; accompanying UML diagrams; and documentation generated by the programming environment used to construct the architecture. This is sufficient for a computing audience.

Architecture is design but not all design is architectural. The overall skeleton architecture of the system is designed here. The designed algorithms (see chapter 4) will be implemented within appropriate methods within this skeleton architecture. The application developed within iteration two will sit on top of the architecture defined here, and use its powerful functionality. This is illustrated in figure 3.1.
The applications act as the presentation layer in a 2-tier like structure, they give an access point for the user to input a qualitatively described scene. The scene is represented as a collection of powerfully structured objects in the software architecture. This representation is used by two algorithms. The first checks the scene for validity. The second takes the corrected scene and visualises it. Depending on how the application uses the architecture the visualisation produced could be of several different formats.

This layered approach is necessary to make the overall system:

- Extensible so any new capabilities can be added to the software without major changes to the underlying architecture.
- Modular so independent software components don’t rely on each other leading to better maintainability. The components then could be implemented and tested in isolation before being integrated to form a desired software system.
- Reusable so another application built at a later date could use this underlying architecture.

This adheres to the advantages implementing extension 5 brings (Design the tool to be extensible, so it is easy to “plug-in’ extra spatial configuration languages at a later date). Whereas extension 3 (Design an extensible more usable application that implements the designed algorithms.) refers to the extensibility of the application using the architecture, so upgrades can be easily built.

The Object-Orientated (OO) [71, 65] style of programming has been chosen on the basis it supports the architecture principles this project would benefit from, discussed above. The features of OO design, such as information hiding, modularity, encapsulation, data abstraction, polymorphism and inheritance, can be used to make the architecture modular, extensible and reusable.

Architecture designs that was looked into and discarded was: Using a RESTful (Representational State Transfer) [25] on SoA (Software as a Service) architecture [49]; A loosely-coupled Event Driven architecture; and a Component-based Software Engineering approach [65]. These were discarded due to the unneeded complexity they bring to the implementation. Components of an architecture accessed via the use of Web Services will add much complexity to the applications using it. Making an architecture networked means all applications using it would need to be executed on machines with a network. Although HTTP is a transport-protocol which is universally adopted and standard across all users machines, the solution should be able to be used locally too.

From the background reading we know a RCC relation consists of two nodes, connected by a relationship. A node can have an identifying label attached. A relationship belongs to a particular qualitative spatial representation language and can be it’s own inverse. The RCC language has a composition ta-
Figure 3.2: UML Class relationship diagram with scaffolding. This means both private methods and public mutators and accessors are shown.

ble, which can be used by applications using the architecture for reasoning. Different languages have different relationships, or in the case of PLCA none at all.

PLCA can use the above concepts by making a node equal to a point. A line equal to a relation between two nodes minus the relationship. A circuit is a set of lines (or relationships), and hence a set of points (or nodes) can be computed, as each line contains a set of two points. An area is a set of circuits. The outermost circuit of a PLCA expression can be identified by attributing a boolean to a circuit. The design can also be extended to be used by the Occlusion Calculus as the calculi is similar to RCC-8. Each relation consists of two nodes and the language uses a composition table for reasoning. See figure 3.3 for the designed architecture, which is explained and justifies below.

The design is orthogonal in nature as representation calculi, relation representation, visualisation approaches, and input systems can be evolved and extended individually without affecting other parts of the software.

The Language

For calculi not included here, the object-orientated architecture has been designed to provide extensi-
bility for the ‘plugging-in’ of other languages. This has been achieved by the Alanguage abstract class. Representations of different qualitative calculi can be extended from this class, and used by the system. A language must have a `checkMatrix(ArrayList<IIInputType>)` method for the system to work. This is a place holder where algorithm one will take a set of objects representing a qualitative scene and generate a correct matrix representation. This architecture decision was decided by analysing the difference between languages (from the background reading). Common attributes were put within the abstract class, and different attributes are implemented within concrete classes.

Matrices have been used as a computationally sound method for representing Directed Acrylic Graphs (DAGs) in the community for some time, as discussed within the background reading. Operations on different edges and vertices are achieved using iterators, as triangle checking is much easier to implement this way. Each languages composition table is also represented by a matrix. Therefore a decision was made to use this same approach in our algorithms. An alternative method that was explored in some depth, was the representation of the DAG using external graph packages Such as JGraphT and iGraph. JGraph was discarded early on due to it’s primary function being to visualise graphs rather than operate on the internal structure, as this complication made it difficult to use. iGraph was written in C and is only on version 0.5. The developer doesn’t have in depth knowledge of C, and the version the external package was at made it a risk to use here. Therefore graph packages were disregarded to favour the matrix representation.

It is worth noting that an implementation of Prolog, reasoning using the SPARQ toolkit was considered rather than using a matrix representation of composition tables. The former was discarded due to the large learning curve for the developer, and the need for a clean integration with visualisation algorithm.

It was decided to place the `checkMatrix(ArrayList<IIInputType>)` method within the language class. Logically the language contains the rules which a scene is being checked against. The scene specifies the abstract ALanguage as a property, making any concrete class extending the abstract class available for use. A scenes language can be switched by `setLanguage(ALanguage)`, and when done so keeps all current nodes and the relations between nodes active (with the option to delete all relations). This method was changed from automatically deleting all relationships as it aids the user in converting relations from one language to another.

**The Nodes**

A collection of nodes is represented within a NodeFactory. This factory gives access to the nodes,
and is composed within a scene. Implementing the simple factory design pattern [30] in this way has a number of powerful advantages: The replacement of nodes within a scene can be achieved efficiently. The node management is extracted from the scene class making the system robust. The number of nodes contained within a scene is truly dynamic. These reasons justified this design decision. The abstract factory design pattern was considered as it provides similar advantages, but added unneeded complexity as several null classes would have to be created.

**The Relations**

RCC styled languages use the concrete Relation class which contains two nodes and a relationship between them. This simple structure extends the IInputType interface. This allows several different representations of relations between nodes possible, as different languages use different relation types. For example PLCA could use a collection of Areas (containing a collection of circuits, lines and points) as it’s input into the algorithm due to this abstract design. This design decision was made to meet extension 5 of the project requirements.

A major design issue that was debated for some time was the representation of the relationship within the relation class. Representing the relationship as a string, where the string equals the same string used to represent the relationship within the concrete language class was decided. For an error free solution the algorithm must then check it’s input matrix and throw an `InvalidRelationship` exception if strings are mis-built. The drawback of the decision is a break in Object-Oriented principles [71, 65], but was needed to make the architecture not too unrealistically complicated.

The OO method of achieving this would be to represent every relationship, belonging to every language defined within the software, as a class which implemented the IRelationship interface. The concrete classes would contain information about its symmetric properties and an identifier. In OO theory this is sound. Although using this bloated structure for representing a relationship within matrices, would be difficult and unnecessary. There are no advantages gained from the structure as the string
representation of the relationship name would have to be checked when operating on cells. The string would just merely be stored in its own class instead of directly within the matrix making the application slower. Another problem that arises from using this structure is the matrix containing different types of objects: Relationships; Nodes; and Strings for the blank cells. Therefore everything would have to be type checked and then cast, (or represented using the base Object class in Java). Ordinary String checking is more computationally sound than checking String’s contained within objects on the heap.

**The Visualiser**

Three visualisation outputs will be attempted by this project. A simple one-dimensional Visualisation, an improved visual appeal to the one-dimensional visualisation and a two-dimensional visualisation. The scene is being visualised. Therefore it was decided the scene should have a visualiser which it can use. A correctly described scene can be visualised by any three approaches, and should be able to be visualised by any other methods decided to be implemented in the future. Therefore a decision was made to make the scene class contain any visualiser that implements the IVisualiser interface. Now the scene can switch visualisers on-the-fly, and the system is extensible for further visualisers to be added.

The Visualise(Scene scene) method of the Visualisation class is a place holder for the visualiser algorithm designed and implemented within the implementation phase of the project.

**The Scene**

The scene class is the main class and encapsulates the functionality provided by all other parts. Using this class makes it easier for programmers of external applications to use the software architecture.

Good error handling is important in any application, but is arguably much more important when building a software architecture. When building an application using an architecture the programmer will want to know if the error was produced by his or the library’s code. If it is the latter it may be produced by an unexpected input, therefore the programmer can alter this - but only if they know what the precise problem is. A proposed exception structure is provided as a UML diagram within appendix H.

### 3.4 Design Issues

It is important to consider the characteristics of the project before discussing potential tools and approaches. This eliminates ones which are obviously inappropriate. The solution is split into two major parts: The Object-Orientated software architecture designed in section 3.3 with embedded algorithms;
and an application with appropriate Graphical User Interface (GUI) which uses this framework. From a tools perspective each parts needs are different. The needs are dependent upon design decisions made within each part.

### 3.4.1 Software Architecture with Embedded Algorithms

An important design decision regarding the output from the visualisation algorithm is important to make at this stage. Different approaches to the visualisation method would change the characteristics of the software.

The basic 1-dimensional visualisation will use ASCII to visualise the scene as this should be the easiest to produce using basic file I/O. This decision was made as the algorithm would be easy to test via the command-prompt. A text file representing the scene would also be more useful to the user.

The improved 1-dimensional and the 2-dimensional visualisation could be done in a variety of ways. The chosen method was to use Scalable Vector graphics (SVG) [22] which uses an XML syntax to represent the scene. This tool was chosen as both XML creation and SVG viewing has standard libraries (JAXP is one example) in most languages. SVG can also be viewed within most modern web browsers, allowing the opportunity for a web application sitting on the layer above the architecture. The XML can also be saved outside of the software.

SVG was chosen over alternative approaches such as representing 1D lines using the Java2D library, and using a custom controls using Swing. Creating and managing a set of custom controls for visualising a scene is too time-intensive and constrains the implementation to specific language, which is counter-productive to the purpose of the architecture design. The advantages of SVG outweigh that of Java2D, and Java2D is too complicated to use for drawing a simple 1D image.

The needs from the design software architecture are as follows:

- The software architecture designed above needs to be developed using a tool which supports Object Orientation.
- The architecture defined above should be implemented in a commonly used language to make it easier to be extended.
- Support for the implementation language being called from a variety of sources is important for making the architecture usable by the widest possible application types possible.
- A tool which automatically generates documentation for the classes would be an advantage.
- Computation time is not too much of a major issue here as the architecture is small.
• The tool must be able to create SVG, and be compatible with file I/O as chosen above.

Based on the above observations a compiled language rather than a scripting language would suit the needs of this application better as the user doesn’t need a specific interpreter to run it. Compiled languages also perform faster at run time, checks all dependencies at compile time and uses less memory whist running. Well suited programming languages matching most of the criteria include Java [47] and JavaDoc [48], C# and SandCastle [20] and C++ and Doxygen[70].

C# is a high-level Microsoft Windows based language that uses the vast .NET class library. Supporting the OO paradigm, having a large user base and being a language known well by the developer makes this well suited for the implementation of the architecture. Microsoft’s new SandCastle documentation generation tool compatible with C# is also good. The one downside of this language however, is that pure C# is only available on the Windows operating system. Java has all the advantages of C# as it has a large class library, a large user base and has an admirable mature documentation generation tool, but is fully-cross platform. As the developer of the project knows both technologies well and is not compliant with C++, Java will be the programming language chosen for implementation of the main software.

3.4.2 Presentation Layer Application

This purpose of this application is to provide a Graphical User Interface (GUI) for the user, where qualitative spatial relationships will be inputted. The GUI must also support the display of the visualisation as output. Firstly a design decision must be made between a windowed application that is executed locally, and a web based application. The primary advantage of a web based application is the ability to access it from any machine, increasing the availability of the application. Other advantages include quicker development time, no installation time and platform independent. Disadvantages though include slower loading time, and no direct secondary storage. Advantages of windowed based applications have over web based is they are more flexible when configuring for specific needs. Here a web based approach is chosen for it’s clear advantages.

Researching appropriate tools for web application development the following tools are appropriate for the task:

Java Servlets [50] are a specific implementation of a Java class, containing overridden methods written in Java that run on a web server. Web requests are processed by these overridden methods for retrieving and submitting data to and from a web browser. This is achieved via a layered approach on-top of HTTP, adhering to the W3C Web Architecture principles. This makes the technology usable
on all machines with an appropriate network connection (or locally if required) and Web Browser. Java Servlets can use any other Java code such as the implemented software architecture. It also facilitate most features of the Java API, including Applets, Java2D, JGraphT, I/O and JAX for SVG creation. As Java Servlets are executed in a web browser, a dedicated control for translating SVG into an image is not necessary as the Web Browser already has this functionality.

Java Server Pages (JSP) [50] are a Java technology that allows for web pages to be generated dynamically in response to a user request. A combination of JavaScript and Java code can be embedded into the standard HTML web page. This allows the content of the page to come from results of Java code itself.

Other notable approaches include ASP.NET and PHP. ASP.NET is a web application framework that allows developers to create web applications using any .NET programming language [46]. PHP is a scripting language that when used in conjunction with a database can also offer the ability to create web applications. However as the developer has little knowledge of using such technologies; the underlying architecture is coded in Java; and Java Servlets are highly compatible with all features required by the application - using Java Servlets for development is logical.

3.4.3 Application and Programming Environment

A software development project such as this requires two environments to be considered, the programming environment and the environment that will run the application. As the chosen language was Java the integrated development environment (IDE) used for the implementation of this project must be capable of both debugging and running Java code, as well as Java syntax correction. A Web based application requires a web server or container in order for the application to be accessible. Software capable of hosting Java Servlet applications also must be chosen.

When choosing a programming IDE for Java a developer has several choices, as many different applications offer similar functionality. JCreator [60], Eclipse [27] and NetBeans [12] are three examples of integrated development environments, all of which meet the requirements of this project. JBuilder requires the purchase of a license whereas Eclipse and NetBeans are both freely available and open source. The developer already has much experience with using Eclipse and has good support for developing Web Application. The Web Tools Platform is an extended version of Eclipse and contains most libraries used for web application development [28]. Plug-ins for testing web applications are provided without having to deploy them on a web server.
Compiled Java class files | \&lt;Tomcat Home\&gt;\webapps\QSRVisual\WEB-INF\classes\ 
Images | \&lt;Tomcat Home\&gt;\webapps\QSRVisual\images\ 
Java library files | \&lt;Tomcat Home\&gt;\webapps\QSRVisual\WEB-INF\lib\ 

Table 3.1: Correct directory structure for web applications running on a Tomcat web container

In order to run the web application and allow access via a web browser, web server software that was capable of running Java Servlets is needed. Apache Tomcat [26] is an open source project that supports Java Servlet technology ‘out of the box’ without any extra configuration or plug ins. There are several open source alternatives to Tomcat, such as the Jigsaw web container for example. However, for Jigsaw to run Java Servlets several plugins need to be installed.

Tomcat requires a certain structure to be used when running Java web based applications. In order to run the system correctly, the application files need to placed in specific tomcat subdirectories. Tomcat requires that compiled Java sources files such as the implemented software architecture defined above, any images and any application specific Java libraries files such as JAX, to be placed in the following directories in order to run correctly.

### 3.5 Conclusions

It has been necessary to alter the project plan here. To design the software, decisions had to be made regarding the design of the algorithms. Therefore some of the algorithm design had to be achieved prior to the implementation phase. Regarding time, the project is behind schedule as this task took longer than expected. The original and revised project plans can be compared, see Appendix B and Appendix C.
Chapter 4

Delivery of the Solution

4.1 Iteration One: Architecture and Algorithms

The primary goal of this iteration is to design and implement two separate algorithms which meet the minimum requirements. As an extension the algorithms will be integrated with an implementation of the software architecture. These makes the algorithms more accessible from a variety of applications. This developments will then be tested. The resulting metrics of which will be used to evaluate the solution within the evaluation chapter.

4.1.1 Checking Correctness of a Qualitative Scene Algorithm Design

The design of an algorithm is proposed which meets minimum requirement two, checking a description of a qualitative spatial scene for validity. This algorithm takes a set of relations (a relation refers to an object composed of two nodes and one relationship). An assumption will be made at this stage that we have a fully defined relationship set between all nodes. A valid directed acrylic graph (DAG) represented by a \([n+1 \times n+1]\) matrix, where \(n\) equals the number of nodes within the scene is then generated. Figure 4.1 shows an example of the matrix structure.

Row and column 0 both contain a string representation of the nodes. The relationships between the nodes are also represented as strings. The proposed algorithm has been designed to work with both RCC-5 and RCC-8 calculi. When the algorithm

\[
\begin{bmatrix}
^a & ^a & ^a & ^c \\
^b & ^c & ^b & ^b \\
^c & ^c & ^c & ^b \\
\end{bmatrix}
\]

Figure 4.1: Example relationship matrix using 3 nodes
is integrated into the architecture, the current language is checked using the scene class as a wrapper. Where a relationship is between two identical nodes, an X is used. This design decision was made during implementation to help with debugging, as EQ was the original choice for representing identical node relationships. X is better than EQ in this situation to easily tear ‘real’ EQ relationships from same node relationships.

An alternative representation of the matrix was found during background reading which represented the relationship between two nodes as transformations used in computer graphics. The transformations could be chosen dependant on the relationships the two nodes had with other nodes. This was established using a hierarchy of matrix transformations via the use of a classic scene graph. This made it easier to visualise the scene, as the location of the regions within space is predetermined. Although we would not be able to check the matrices validity against a composition table. As one of the objectives of the project is to check the validity of a qualitative scene, representing relationships using the calculi was the chosen method. Appendix I contains the pseudo code for the building of the relationship matrix.

Once the relationship matrix has been generated from a description of a scene it needs to be checked for validity. A suitable method of performing reasoning on a DAG was discussed in the background reading, see chapter 2. This involved using a composition matrix, and iterating through node’s relationships represented in the DAG and performing ‘triangle checking’. The composition matrix is a square matrix whose first row and column are relationships defined by the composition table’s calculi, see figure 2.4. Triangle checking refers to using two relations, say R1 and R2 to find the correct relationship for R3 within a triangle of nodes, see figure 4.2. Using this reasoning approach for scenes with less than three nodes is unnecessary as all permutations of relationships are correct. Program 1 shows some pseudo code for triangle checking.

Several problems were encountered during the design of this algorithm, a few major points of which are discussed:

- What if empty nodes existed, and a full representation of the scene is not assumed? This algorithm would be ineffective as many triangles would exist without two known edges. Triangle checking must fill in the blank relationships with valid ones. This is discussed as an extension within iteration two.
Program 1 Proposed pseudo code for checking a matrix for validity. The log will contain all incorrectly defined relationships (including empty ones). See appendix I for chooseRelation function pseudo code.

```
//compute and insert computer defined relations

SET n to the matrix length
IF matrix length is greater than 2
    FOR i to n - 3
        FOR j to n - 2
            FOR k to n - 1
                IF matrix[k][j] is null
                    IF matrix[k][j] is not same as result of CALL chooseRelation with Matrix[j][j], Matrix[i][k]
                        WRITE to log
                    END IF
                END IF
            END FOR
        END FOR
    END FOR
END IF
```

- What if one iteration of triangle checking produces one result, and another produces a separate result? As was discovered in the background research (see chapter 2, several qualitative descriptions of a scene could be equally valid therefore this is possible.

- Within the composition table (see figure 2.4) it is possible for a number of relationships to be equally valid for R3 when R1 and R2 are presented. How should one be decided? A basic algorithm would choose the first one that was presented. A more sophisticated program would record all possible valid relations for the graph edge. This could be done by changing the relationship matrix into a three dimensional matrix of strings. An array of possible correct relationships would then exist for each edge of the DAG. Different iterations of the triangle checking would then add extra possible relationships to the array. This may result in an array of relationships, some of which are repeated. A voting scheme could be utilised which gives the edge a single relationship. If no repeated relationships exist the first one could be chosen. An even more sophisticated solution would be to check each possible valid relation against a set of rules. The rules could choose a relation dependant on neighbouring node’s valid relations. The rules could then be tuned to make the visualisation of the matrix as “clean” as possible.

- What if one iteration of triangle checking results in one relationship, and another result in a different relationship? As was discovered in the background research (see chapter 2, several qualitative descriptions of a scene could be equally valid therefore this is possible. A simple version of the algorithm would let future iterations of the triangle checking algorithm to overwrite the previous relationship. The voting scheme discussed above would also solve this problem.
4.1.2 1-Dimension Visualisation Algorithm Design

The design of an algorithm is proposed which meets minimum requirement three, producing a basic visualisation of a qualitative spatial scene. An assumption will be made at this stage that we have a valid and full relationship set between all nodes which is passed to this algorithm. To meet the minimum requirements quickly it was decided to first visualise the qualitative spatial scene in 1-dimension. This algorithm has been designed with RCC-5 in mind, but is easily extensible. The visualisation here will be produced within a text file as ASCII. More sophisticated visualisations are proposed in later iterations of the implementation phase.

A major design decision was to decide how the visualisation would look. It was decided that the visualisation should look similar to Allen’s thirteen interval-interval relations discussed in chapter 2. See figure 2.3. This was so the algorithm could be adapted easily to RCC-8 because a connected edge is visualised. This method is also easily legible by the user, as well as easy to produce.

Each node in the DAG, which refers to a relation will be given one line. This line will be represented by a start point ‘—’ and an end point ‘—’. In between these two points a number of middle sections may exist, depending on the complexity of the visualisation. Middle points will be represented by ‘——’. Each time interval will be represented by four characters within the file. At each time interval a decision would be made to either start a node’s line, continue a node’s line, or end a node’s line. Obviously a line cannot be continued or ended if it has not been started. A line too cannot be started if it already is being extended.

Starting a node’s line after the end of the same node’s line that has already passed means the region represented by the node is multi-region. Going back to the problem being solved, the application is generally going to be used to visualise a scene that is normally hard to visualise. Therefore the design decision to make relations single-region was made. This is because

![Figure 4.3: An example of a 1-dimension visualised spatial configuration in ASCII.](image)

![Figure 4.4: An example configuration which cannot be represented as a 1-D visualisation.](image)
the amount of different visualisation configurations presented using multi-regions would look very noisy. A similar problem to this has been mentioned when attempting to visualise a PLCA described scene, called circle packing. All nodes being single region means rules are needed to be in place to enforce single-regions. One method to achieve this is to use a topological sorting algorithm on the DAG, as it strips out repeated elements. However, for this to be effective the data needs to be in the correct format. Using a topological sort on the matrix itself would be ineffective. Therefore the format which can be used by the topological sorting algorithm must be discussed.

For a single line we only need to represent the start and the end. As at each time interval the algorithm would need to know if it is to start or end a line. This can be represented by $s(a)$ and $e(a)$ respectfully. The decision not to represent the continuation of a line within the encoding to input into a topological sorting algorithm was made because several continuation steps could not be represented, as repeated information is stripped. Continuation then depends on whether the start has been seen before, and the end has not been seen before.

Below are the five relationships within the RCC-5 calculus, and there respective represented visualisation using the proposed format above:

- **EQ(a,b)**. Represented by the start points of both nodes starting, and the end points of both nodes ending, within the same time interval.

- **PP(a,b)**. Represented by the start point of $b$ occurring before a’s start point. As well as the end point of $a$ occurring before $b$’s end point.

- **PPi(a,b)**. Represented by the inverse of the above. The start point of $a$ occurring before the start point of $b$. And the end point of $b$ occurring before a’s end point.

- **DC(a,b)**. Represented by the end point of $a$ occurring before the start point of $b$. Or the end point of $b$ occurring before the start point of $a$.

- **PO(a,b)**. Represented by the start point of $a$ before the end and start point of $b$, as well as the end point of $a$ before the end point of $b$. Or represented by the start point of $b$ before the end and start point of $a$, as well as the end point of $b$ before the end point of $a$.

The algorithm takes the square matrix and converts the relations of the scene into the format appropriate for topological sorting using the above list of rules. The topological sorting algorithm then processes the unsorted formatted input. The result of which is then used to determine the start and end points of each line. These start and end points are then parsed and drawn.
Drawing the topologically sorted representations of the start and end points is achieved by iterating over the parsed elements, treating each element as a single time step. Each time step will be treated as 4 characters. If the node within the element is unseen the string representation of the node is printed as character one on a new line, then the ‘|—’ is printed in the appropriate position relative to the current time step. This makes the name of the node the last character on any line, this was important due to the variable nature of the nodes name. If the element has been seen before the ‘—|’ is printed in the appropriate position relative to the current time step. Then for all nodes seen once before bar the one just read, ‘——’ is printed in the appropriate position relative to the current time step.

Several problems was encountered during the design of this algorithm, a few major points of which are discussed:

- Disjunction’s exist with some of the above mentioned configurations. Therefore several different visualisations of the same scene may be possible. An Extension, designed and implemented within iteration three deals with this issue.

- It was realised that some visualisations cannot be implemented within one dimension. This is even true if constrains were loosened regarding multi-regions. For example if three objects are partially overlapping another object, see figure 4.4. The only way to overcome this problem is to upgrade to the next dimension, which will also be attempted in iteration two.

- It was realised that for the topological sorting algorithm to work correctly, the algorithm had to see a set of example objects being sorted. Therefore the first input to the algorithm will be the start and end representation of all nodes from 1 to n, where n equals the number of nodes in the DAG.

- Using the above configuration, s(a) s(b) e(a) e(b) in the context of EQ(a,b) would be drawn as PO(a,b). Each element is considered a single time step. Therefore an additional element is needed to represent the start of both a and b in a single time step, as well as the end of a and b at a single time step. This has been achieved by introducing the elements s(ab) and e(ab) to the configuration. This changes the complexity of proposed algorithm as two nodes may be possible for one element.

### 4.1.3 Implementation of the Algorithms

After the designed architecture was implemented, as proposed, the first algorithm has been integrated within several methods belonging to the *RCC5Language* concrete class. The conversion from the scene description using objects, into the matrix format is performed in `matrix CreateMatrix(ArrayList<IInputType>)`. `boolean CheckMatrix(matrix)` takes a full relationship matrix and checks it for validity. This is
achieved using both the private method `ChooseRelation(String r1, String r2)` and the composition matrix property.

The second algorithm, as proposed by the design, was integrated into the 1DVisualiser concrete class. The `void visualise(matrix)` method computes a 1-dimensional visualisation and writes it as ASCII into a text file. It does this using the `tsort` component, and two temporary flat files named unsorted.txt and sorted.txt. The flow of events is pictured in figure 4.6. If any errors occur during the execution of the algorithms an appropriate error is thrown. See Appendix H for a description of custom error classes.

### 4.1.4 Testing

As laid out in the project methodology, after each iteration testing will be carried out by the developer to see if the implemented features are effective solutions to the problems they were meant to address. Developer testing will involve inputting a qualitatively described scenes, each with a different set of nodes and relationships. The result of this testing should provide adequate information to draw conclusions as to the success of each iteration. The results of each test can be found in appendix G. The findings from this testing are summarised below.

Although the debugging during development was the most effective method of teasing out obvious bugs, many bugs still existed when performing black-box testing. Unfortunately some were unfixable in the time allotted. This effected the results of the testing. Six scenes in total were tested on both algorithms. The scenes spatial configurations were chosen specifically to test different aspects of the algorithms. The reasons why the spatial configurations were chosen is discussed below, along with the results of testing each scene:

- A small scene using PP between three objects. This scene was used as a baseline to ensure the fundamentals of the system work as expected, as bugs were expected. As expected after debugging, the test was a success.

- An impossible scene to visualise in one dimension. This scene contains three regions which partially overlap a single region. The expected result was the checker algorithm will pass, but the visualiser will display an erroneous result. The checker algorithm indeed passed here, but the topological sort failed. Therefore this test is partly a pass and a fail.
Figure 4.6: *UML Sequence Diagram representing the implementation algorithms within the software architecture. Mark one: The algorithm1 interface calls the check scene method. Mark two: The algorithm1 interface calls the visualise method.*
- A scene containing an EQ relationship. The visualiser algorithm had to be adapted to deal with two nodes starting and ending at the same time step. Therefore, it was decided to test this specific functionality. On first testing the system had to be changed as the s(ab) and e(ab) elements were written to the unsorted.text file incorrectly. After this error as fixed the test was a success.

- A scene containing a DC relationship. This test exists to check how the visualiser reacts to the DC relationship. As expected, the test passed.

- A scene which is invalid. A scene which was invalid according to the rules within the composition table was tested. This was chosen to test if the system reacts as expected. Unfortunately, the logging of an incorrect relationship was broken. It was expected that the log StringBuilder object contained a string representation of incorrect relationships, but this was not the case.

- A scene containing a PO relationship. This test exists to check how the visualiser reacts to the PO relationship. As expected, the test passed.

67.6% of the tests on the visualiser algorithm fully succeeded. 83.3% of tests on the checker algorithm were a success. The overall success rate of the system working at this stage is 75%. Therefore, the system works although some issues exist that need to be resolved. A discussion on these results are present in the evaluation chapter.

### 4.1.5 Conclusion

Differences were present comparing the original plan’s timing to the actual timing it took to implement this section. Iteration one took longer than originally expected, and was completed much later in the schedule than anticipated. This has been illustrated within appendix C. Table 4.1.5 explains how the actual implementation of iteration one is different to the plan, and if so, why.

<table>
<thead>
<tr>
<th>Planned Task</th>
<th>Actual</th>
<th>Reason for difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design an algorithm which checks the validity of a qualitative spatial configuration</td>
<td>Achieved along with several enhancements</td>
<td>N/A</td>
</tr>
<tr>
<td>Implement and test the algorithm</td>
<td>Achieved along with several enhancements within the implementation. Although a couple of bugs exist</td>
<td>N/A</td>
</tr>
<tr>
<td>Design an algorithm which visualises a qualitative spatial configuration in 1-dimension</td>
<td>Achieved along with several enhancements</td>
<td>N/A</td>
</tr>
<tr>
<td>Implement and test the algorithm</td>
<td>Achieved, Although a couple of bugs exist</td>
<td>N/A</td>
</tr>
<tr>
<td>Implement the designed software architecture</td>
<td>Achieved</td>
<td>N/A</td>
</tr>
<tr>
<td>Then integrate the algorithms into the implemented software architecture</td>
<td>Achieved already within previous task as the two tasks were combined</td>
<td>During this iteration time was running out therefore the implementation of the algorithms were integrated straight away.</td>
</tr>
</tbody>
</table>

Table 4.1: First iterations goals.
4.2 Iteration Two: Application

The goals of this iteration are to improve the visualisation algorithm, giving the user a more visually appealing description of the scene. The qualitative spatial scene checker algorithm will be improved to handle incomplete scenes. Finally a prototype web application which sits on top of the software architecture is designed and developer. This makes the algorithms more accessible for users. The algorithms in this iteration will then be tested, and a user evaluation will commence on the web application. The metrics which will be used to evaluate the solution within the evaluation chapter.

4.2.1 Improved 1-Dimension Visualisation Algorithm Design

An improvement on the previous visualisation algorithm is presented here. The tool chosen for this task was SVG, as discussed within chapter three. As with the previous algorithm, the matrix is parsed and relationships between nodes are converted into start and end points depending on the rules defined in section 4.1.2. A collection of these start and end points are contained within unsorted.txt and then topologically sorted. This algorithm differs in the interpretation from the topologically sorted data into a visualisation.

The manner in which the nodes are represented will be the same as the previous algorithm, see figure 4.7. Writing an algorithm for visualising different nodes positions differed considerably between different file formats. This is due to the inherent differences between writing directly to a text file and generating SVG elements. Each start or end element represents a time step within the diagrams X direction, in intervals of 10 pixels in SVG. The Y direction represents different node’s at intervals of 10 pixels. The major difference between writing SVG and text on-the-fly, is with ASCII when text is inserted into the middle of a line, the characters present after the inserted text on the line are shifted automatically. With SVG, elements have to be manually given X and Y co-ordinates. Therefore for each element added to the SVG file, the X and Y co-ordinates need to be computed. To achieve this information was needed regarding which nodes have been started and ended. Program 2 demonstrates the pseudo code of the proposed algorithm.
Program 2 Proposed pseudo code for visualising a topological sorted representation of the start and end points of each node.

INITIATE SVG file

INITIATE tuples to (string nodename, bool haveSeenBefore, bool has ended, int yValue)

INITIATE array of tuples

Iterate over nodes in the matrix and get all node names and populate tuples

SET nodeCount to matrix length - 1

SET seenNodeCount to 0

SET time to 0

FOR each element within sorted.txt until EoF

SET startX = time + 10

SET endX = time + 20

IF element is an s

FOR each nodes in s

SET y to seenNodeCount * 10

WRITE line at position (startX, y : endX, y)

WRITE line at position (startX, y-2 : startX, y+2)

SET tuple for current node to (elements nodename, true, false, y)

SET seenNodeCount++

END FOR

END IF

IF element is an e

FOR each nodes in e

SET y to nodes tuple(yValue)

WRITE line at position (startX, y : endX, y)

WRITE line at position (endX, y-2 : endX, y+2)

WRITE node name as text at position (endX + 2, y : endX + 2, y)

SET tuple for current node to (elements nodename, true, true, y)

END FOR

END IF

FOR all nodes in tuple that have been seen and not been ended

SET y to nodes tuples yValue

WRITE line at position (startX, y : endX, y)

END FOR

SET time++

END FOR

4.2.2 Improvements to the ‘Checker’ Algorithm Design

An improvement on the previous algorithm which checks the validity of a qualitatively described scene is proposed here. This algorithm differs from the previous by filling in the relationship when one is found to be incorrect, rather than merely logging the error. Within the ChooseRelation() function (see appendix I), when querying the composition table, the returned value is an array of valid disjunctive relationships. Various methods for choosing one relationship was discussed in iteration one. Here each relationship within the matrix is populated with an array of several arrays containing valid relationships,
one array returned per triangle check. Afterwards a four-dimensional matrix of all possible valid relationships between all nodes exists. Each cell is then queried and all relationships are counted. The relationship occurring the most often is chosen. Program 3 shows the pseudo code for this algorithm.

**Program 3** Proposed pseudo code for filling in blank relationships within the matrix. See appendix I for `chooseRelation()` function pseudo code.

```java
//compute and insert computer defined relations
SET n to the matrix length
IF matrix length greater than 0
    FOR i to n - 3
        FOR j to n - 2
            FOR k to n - 1
                IF matrix[k][j] is null
                    SET matrix[k][j] to result of CALL chooseRelation() with Matrix[j][j], Matrix[i][k]
                END IF
            END FOR
        END FOR
    END FOR
    IF matrix[k][j] has more than one relation
        SET matrix[k][j] to result of COMPUTE most occurring relationship
    END IF
END IF
END IF
```

Problems exists with this algorithm. What if one relationship is valid during the result of one ‘triangle check’ and not valid in another? In this situation an invalid relationship would need to be removed from all other internal arrays. The result then would be an array of an array of valid relationships, taking into account the entire scene. This representation could be used to enable the user to choose between different visualisation configurations, as several valid descriptions of the scene may be valid.

### 4.2.3 Web Application

The previous interface to the software architecture was via the console, for testing purposes. This is improved upon by creating a web interface, see figure 4.9. A Java Servlet is created and deployed locally using the Tomcat Web Container. A Java Servlet differs from classic HTML by programmatically generating content. This is achieved by programming a class which extends the `HttpServletRequest` abstract class provided within the Java Servlet API. A HTTP request is captured by the Servlet, the contents of the overridden `doGet()` and `doPost()` methods are triggered by HTTP GET or POST, respectfully. The programmer can use this to programmatically generate content for the Web Browser to interpret and display.
Figure 4.8: UML Sequence Diagram showing how the web application interacts with the software architecture.
These two events allow access to the software architecture. At marker one within figure 4.8 the web application is launched triggering a HTTP GET event. Here the scene, visualiser and language are all initialised. As the application is being executed locally both visualisation methods will work (ASCII and SVG), however the SVG has been chosen to be integrated here as most modern browsers display this format. The chosen calculi which will be used here is RCC-5, therefore a RCC5Language language object will be created and initialised with the calculi’s composition table.

The web form is then sent to the browser. This provides a basic interface allowing the user to input triples, containing two objects and a connecting relationship. A number of these triples make up the qualitative description of a spatial scene. The user then clicks on the forms submit button triggering a HTTP POST event, see marker two within figure 4.8. It was decided to add a method to the Scene and NodeFactory class which added a batch of nodes to a scene. Otherwise the SetNode(name) would have to be called numerous times which is computationally inefficient. The relationship matrix is created, checked and then filled using the improved algorithm. The scene is then visualised, creating a String containing an SVG document. The string is returned to the Servlet which is then parsed and given to the Web Browser which loads the image. This is demonstrated as a UML sequence diagram, see figure 4.8.

4.2.4 2-Dimension Visualisation Algorithm Design

A 2-dimensional visualisation of a qualitative spatial scene is a better approach than using one dimension. Not only because it provides easier to read diagrams, especially for more complex scenes. Some scenes of specific configurations are impossible to visualise within only one dimension, see section 4.1.2. Therefore it is justified this extension should be attempted. An algorithm which uses a relationship matrix to visualise a scene, would need to differ from the 1-dimensional approach discussed above. The topologically sorted start and end notation used within the previous algorithms could still be used. However, the decisions made where to draw regions in respect to one another would change.
Within the background research when showing relationships between two regions in pictorial form generally circles are used, for example see figure 2.2. As the user of this application will be a student or practitioner of QSR this approach for 2-D visualisation could be adopted, as circles are easy to read and are generally accepted within the QSR community as a form of pictorial representations of relationships. SVG was the chosen tool to visualise in 2-dimensions. SVG needs three attributes to draw a circle, a radius and an XY coordinate for the circle’s centre.

Choosing where to place one object in respect to another is straightforward when adopting a set of rules. For example: To visualise PP or PPI one circle’s radius needs to be smaller than another’s and the XY values of both will be identical; To visualise PO the X value of circle B could be the X value of A plus A’s radius; To visualise DC the X of A needs to be the X of A plus the radius of A and the radius of B plus an arbitrary number to ensure disconnection; EQ would be represented by A and B having equal X, Y and radius values.

However, visualising a set of regions will be much more complicated. This is due to the position of each object needing to change with respect to every other object as the scene is being drawn. For example if we are drawing the scene in image 4.4 using the rules written above, problems would arise. The drawing of object B as partially overlapping object A needs to be drawn at the “other side” of A’s boundary. Therefore the set of rules choosing the positions of circles needs to take all situations into account. The number of different situations when considering infinite regions is large, therefore this task would be challenging. Especially when the visualisation should be kept legible and sensible. A different approach could adapt a Genetic Algorithm (GA) for finding the best fit of a set of regions, as discussed by Kumokawa and Takahashi [42].

4.2.5 Providing the User with Alternative Correct Visualisations

Within section 4.2.2 a method of storing a set of all valid relationships between two nodes was proposed. An array containing a set of array’s each containing a relationship. Each inner array contains all valid relationships for one triangle check. The winning relationship between two nodes was chosen using a voting system. To enable users to choose between different relationship configurations, the amount of configurations possible \( (n) \) needs to be computed. This would be done by taking one valid relationship of one cell in the matrix, and pairing this with one relationship within every other cell in the matrix. Doing this for all relationships would result in \( n \) configurations.

Once this has been achieved all possible configurations could be represented in a clique. Weights
representing the differences in configurations could be placed on the edges. Therefore when a user tells
the application they want an alternative representation of a scene the most different configuration is cho-
sen. The application would then have to be able to process the recording of which scene configurations
have been loaded previously to avoid repetition.

4.2.6 Testing and User Evaluation

As laid out in the project methodology, after each iteration testing will be carried out by the developer to
see if the implemented features are effective solutions to the problems they were meant to address. Iter-
ation two’s testing will comprise of two major parts. The first part comprises of testing the implemented
updated algorithms using various different qualitatively described scenes. The second part involves
observing a few users interact with the application and asking them to complete a short questionnaire.

Developer Testing

The two algorithms tested here were, the updated 1-D visualisation algorithm which produces SVG,
and the updated scene checker algorithm which now fills-in missing relationships. Eight scenes were
tested, six identical from iteration one’s testing, and two additional scenes. The full results of each test
can be found in appendix G. The results of testing are discussed below:

- As with iteration one’s testing, the PP scene, the impossible scene, the DC scene and the PO scene gave
  identical results.

- The invalid scene failed. This was due to the algorithm trying to fill in the invalid relation with a correct
  one from the results of other triangle checking iterations. There exists bugs in this area of the code.

- The EQ scene passed the checking algorithm but failed with the visualisation. This must be related to the
  algorithm drawing two start positions at the same time point. This issue is still unresolved.

- Testing the scene which was incomplete with one relation was successful. Although the scene with several
  incomplete relations failed within the checker algorithm.

60% of the tests on the visualiser algorithm fully succeeded. 75% of tests on the checker algorithm
were a success. The overall success rate of the system working at this stage is therefore 68%. Therefore
the system works although some issues exist that need to be resolved. A discussion of these results are
present in the evaluation chapter.

User Testing
Users were involved with the testing of the Web Application developed within this iteration. The results of this activity are presented and discussed within the evaluation chapter, see section 5.1.2.3.

### 4.2.7 Conclusion

Table 4.1.5 explains how the actual implementation of iteration one is different to the plan, and if so, why.

<table>
<thead>
<tr>
<th>Planned</th>
<th>Actual</th>
<th>Reason for difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expand the design and implementation of the 'checking' algorithm to ‘fill-in’ missing relationships</td>
<td>Achieved</td>
<td>N/A</td>
</tr>
<tr>
<td>Improve the implementation of the visualisation algorithm to provide the user with a more appealing visualisation.</td>
<td>Achieved and more</td>
<td>A topological sorting algorithm was implemented replacing the external component</td>
</tr>
<tr>
<td>Design an algorithm which visualises a qualitative spatial configuration in 2-dimensions</td>
<td>Not achieved but discussed</td>
<td>N/A</td>
</tr>
<tr>
<td>Implement this algorithm, then integrate it into the application.</td>
<td>Not achieved.</td>
<td>Complexities and time issues</td>
</tr>
<tr>
<td>Design a system which allows the user to change between a scenes valid visualisations.</td>
<td>Not achieved but discussed</td>
<td>Time issues</td>
</tr>
</tbody>
</table>

Table 4.2: Second iterations goals.
Chapter 5

Evaluation and Conclusions

5.1 Evaluation

The aim of this project was enhance the role of human cognition in the process of developing qualitative spatial descriptions. Objectives were set out for the project to achieve this aim. Therefore solving the problem. In order to measure the success of the project, appropriate evaluation criteria are presented, see section 5.1.1. How well these criteria are met measures the success of the project. See section 5.1.2 for the results of the evaluation and section 5.1.3 for a discussion.

5.1.1 Evaluation Criteria

Within a software project it is generally good practice to know how the system will be tested before development. Therefore a high level testing and evaluation plan was proposed prior to the design and implementation of the solution. The implementation phase of the project was split into two iterations. Iteration one met the minimum requirements of the project and therefore met the project objectives. Iteration two improved upon the minimum requirements to achieve a better quality solution. The testing and evaluation plan suggested testing the functionality of the implemented algorithms via testing. This functional testing was completed at the end of iterations one and two. The original problem statement involved users, and an application interfacing with the functionality of the algorithms was an extension of the project. Therefore a user evaluation was proposed to test the success of this extension, as well as test the overall functionality of the system.

As the project objectives involved the design of a solution, measuring the success of the project
will take this into account too. Therefore if implementation bugs existed affecting testing metrics, but the design is well orchestrated, this will be discussed. The evaluation criteria are made up from the meeting of the minimum requirements to gauge if the core project objectives were met; the meeting of the extensions to gauge the quality of the solution; and the results from testing and the user evaluation to gauge success in implementing the designed solution. These criteria are evaluated against in the following section.

5.1.2 Evaluation using Evaluation Criteria

5.1.2.1 Minimum Requirements and Extensions

<table>
<thead>
<tr>
<th>Minimum Requirement</th>
<th>Was this met by the project?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design a tool which checks a qualitative spatial configuration for validity</td>
<td>This criterion has been met. An algorithm has been designed which checks an inputted representation of a scene for validity against a composition table. A tool has been designed and implemented which demonstrates this algorithm which has been evaluated in section 5.1.2.2.</td>
</tr>
<tr>
<td>Design a tool which enables the visualisation of qualitative spatial configurations</td>
<td>This criterion has been met. An algorithm has been designed which visualises a scene using one dimension. A tool has then been designed and implemented to demonstrate this functionality which has been evaluated in section 5.1.2.2.</td>
</tr>
<tr>
<td>An evaluation of the system to aid in further work.</td>
<td>This criterion has been met. Throughout the project there has been a focus on aiding future work. For example, designing an extensible software system, designing algorithms with other languages in mind, and designing the web application as an example of using the software. Further work is discussed in detail in section 5.2.1.</td>
</tr>
</tbody>
</table>

Table 5.1: Project minimum requirements.
<table>
<thead>
<tr>
<th>Extensions</th>
<th>Was this met by the project?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design an algorithm which &quot;fills-in\’ missing relationships\ within a qualitative description of a scene.</td>
<td>This criterion has been met. An algorithm has been designed which fills in missing relationships. This has too been implemented, of which is evaluated in section 5.1.2.2</td>
</tr>
<tr>
<td>Design an algorithm which gives an improved visual appeal to the one-dimensional visualisation.</td>
<td>This criterion has been met. An algorithm has been designed for creating a 1-D visualisation in SVG. This has been implemented, of which has been evaluated in section 5.1.2.2</td>
</tr>
<tr>
<td>Design and build an extensible more usable application that interfaces with the algorithm</td>
<td>This criterion has been met. A web application has been designed and built, which interfaces with the software and algorithms underneath. The application has been evaluated via a user evaluation in section 5.1.2.3</td>
</tr>
<tr>
<td>Design an algorithm which gives a more comprehensive true visualisation using two-dimensions.</td>
<td>This criterion has not been met, but possible approaches have been discussed.</td>
</tr>
<tr>
<td>Design the tool to be extensible, so it is easy to &quot;plug-in\’ extra spatial configuration languages at a later date.</td>
<td>This criterion has been met. The software architecture has been designed and implemented to be extensible.</td>
</tr>
<tr>
<td>A system could be provided that allows different valid relationships to be chosen for different visualisations</td>
<td>This criterion has not been met, but possible approaches have been discussed.</td>
</tr>
</tbody>
</table>

Table 5.2: Project extensions.

5.1.2.2 Quantitative Testing

Testing was performed at the end of both iterations. The purpose of this was to evaluate the implementation of the algorithms, and their functionalities performance. The results were presented in sections 4.1.4 and 4.2.6. Here these results will be analysed further to gauge what they mean.

Analysing where tests failed within iteration one showed that bugs in the code was what negatively
affected the results rather than the design of the algorithms. Iteration two’s results however were more inconclusive as to whether or not they failed due to poor algorithm design or implementation bugs.

5.1.2.3 User Evaluation

When designing the questionnaire clear goals were in mind. The primary purpose of the questionnaire and observation was to measure how useful the functionality of the application was. Secondary goals were to measure how good the application was in terms of several metrics. The purpose of the user evaluation was to evaluate the effectiveness of extension 3 of the project (Design and build an extensible more usable application that interfaces with the algorithm), and gauge the overall impression of the tools usefulness. The design of the questions were based on discussions within section 3.2.

Eight students participated in testing the application, all of which are between 20 and 30 years of age, very computer literate and know of KRR and QSR. Although the sample rate is not representative of the population, it is representative of the users which will benefit from the application.

The users were given two scenes in which to input into the Web Application. One scene was also inputted into the application at the participants own discretion. Questions were asked on the functionality of the application and if the visualisation presented was useful (see appendix F for the full questionnaire). The results of this were not as successful as hoped. The visualisation of scene one worked for all participants, but scene two and the users custom scene were unsuccessful. The impressions of the applications functionality, demonstrated by scene one was positive overall. Participants joked how
useful the application would be in the KRR exam.

Five quantitative based questions were then asked regarding the usability, intuitiveness, attractiveness and overall impressions of the application. The results of which are represented as a graph in figure 5.1. Overall the application was intuitive and usable when it worked. However, the attractiveness was not as good. Overall the user evaluation was not as successful as hoped. The results showed the applications functionality is positively received but could be improved upon by improving the GUI and making it fully working.

5.1.3 Results of the Evaluation

The project has met all criteria from the minimum requirements. Most of the enhancement criteria has also been met, ones which haven’t been met have however been discussed. The implementation though does contain bugs, as demonstrated by the testing and the user observations. Although the results of testing were inconclusive as to whether the implementation works, it is believed the overall design of the algorithms and software, is on the majority, good. A comparison with Existing Solutions would have been useful, but as discussed within the background research chapter this was not achievable.

It is felt that the system won’t be able to cope with larger scenes as further errors would emerge within the implementation. The software does however demonstrate the algorithms and provide a base for further work, which was the aim of the overall project. The objectives of the project has been met and exceeded. the objectives were as follows:

Learn more about Qualitative Spatial Representation and the calculi used for defining relationships, which has been achieved in the background research; Design a prototype application and algorithm which checks a set of qualitative relationships for validity, and Design a prototype application and algorithm which visualises qualitative spatial relationships the user has described. Which has been achieved in the design and implementation chapters. Finally, Identify the limitations and possible improvements of the algorithms developed to aid in further development which was discussed at the end of each algorithms design and in the further work section.

5.2 Conclusions

The problem statement proposed that mistakes may be made when developing qualitative spatial descriptions of scenes, as people may miss out obvious relationships. Separately it was proposed that if a
user is given a set of spatial constraints it will be hard to visualise the scene. The aim of this project was enhance the role of human cognition in the process of developing qualitative spatial descriptions.

This was achieved by providing a system which checks a qualitatively described spatial scene for validity, and provides a visualisation of the scene. The design of algorithms which would achieve this goal were the minimum requirements of the project. A software architecture was also proposed where the algorithms could be implemented, making the functionality available to a variety of applications. These algorithms were then extended to provide better quality solutions. A Web Application was then presented which provided an interface with the algorithms contained within the architecture.

Analysis of the results found in the evaluation chapter has shown that overall, the projects aim has been met. The developer is happy with the design of the system, and the algorithms. This being said, the implementation was poorer than the developer had hoped for.

An appropriate methodology was chosen for the development of the solution. This methodology was successfully followed throughout the design and development process. Splitting the implementation into two iterations helped the developer focus on the important tasks at hand earlier in the project. It is worth noting the original schedule planned for three iterations within the implementation phase. The third containing extensions that took more time, such as the implementation of the 2-D visualisation algorithm for example. However due to unforeseen personal events the developer lost time, therefore a decision was made to cut the implementation phase into two iterations. Other than this the structure of the original project schedule was followed. The time tasks were completed however were delayed. The original and revised project plans can be compared, see appendix B and C.

Overall the problem has been solved but many areas can be improved upon and extended.

5.2.1 Suggestions for Further Work

Further work carried out on this project could include:

- The designed algorithms for checking a qualitatively described spatial scene could be adapted and tested with RCC-8. Within the checking algorithm this would be a simple change in using a different composition table. However the visualisation algorithm would have to be edited to include more rules for defining the start and end positions of visualised regions. The addition of these rules may produce more scenes which are impossible to visualise.

- The web application could be improved in a variety of ways. Instead of the user manually inputting object names, the user could save a set of objects and choose one from a list. Also an add button could add a
relation one at a time to the scene, for on-the-fly visualisations. Instead of processing the checker and the visualiser on the same postback, the checker could execute initially for filling in missing relationships, and give feedback to the user. The user can then choose one of \( n \) valid scenes to visualise on the second postback.

- The extensions discussed within section 4.1.1 related to the scene checker algorithm could be implemented. This includes using a set of rules for ‘good’ relationship configurations.
- The PLCA 2-dimensional visualisation algorithm proposed in [42] proposes using a genetic algorithm (GA) for learning the best visualisation for a scene. This approach could be experimented with and incorporated into this software.
- A set of objects along with their respective relationships could be saved to a database, or a portable XML file. This allows the user to load up previous scenes, and add relationships to scenes at a later date.
- The architecture allows for PLCA expressions to be expressed, but no reasoning algorithm has been proposed. An algorithm could be designed for reasoning and visualising PLCA expressions. This would be massively different from the algorithms developed for RCC. Relationships between nodes on the DAG would not be represented by the JEPD relationships defined in the calculi. Instead what points, lines and circuits contained within what areas would need to be computed.
- Support for representing more complicated regions could be integrated into the system. Including convex hulls and fuzzy boundaries which were mentioned within the background research chapter.
- A more empirical testing suite could be developed for the software, including unit tests. This would make the teasing out of bugs much more efficient.
Bibliography


Appendix A

Appendix A - Personal Reflection

This project has been the most challenging piece of work of my degree, not in terms of technical content, but all the equally important softer skills that us technical people always take for granted. Gathering sources, reading the challenging material, choosing what to write about, and critically writing relevant points about the literature I found extremely challenging - and is where most of the marks lie for this major piece of work. Therefore I would recommend devoting more time to these tasks than anticipated.

The project was also challenging due to emotional responses at the sheer size and importance of the work. I would recommend not to look at the project as a classification decider, or as a piece of work the result of which will ‘effect the rest of your life”’. Treat it like a large piece of coursework. This way you will get things done rather than panicking and planning too much.

I am a little of a perfectionist and a piece of advise given to me that sticks in my mind is to not treat every word you write down as “black or white’. Perfect or rubbish. Treat everything more maturely with shades of grey, some bits will be better than others, this is natural and reflects how people in any industry want you to work. get everything done the best you can rather than perfect or non existent.

Having come to the end of this project I wish I’d had more time to focus my whole efforts on it. I am the kind of person that cannot multi-task, and gets things done one thing at a time. When time is being split with other commitments such as coursework’s, exams and family I focus on one thing at once. Therefore I did large parts of my project in a few weeks here and a week or two there - rather than a couple of hours a day. This means though when having large chunks of time making sure you use that time to it’s fullest. This is different to most peoples advise of ”doing a little bit every day” - but people work differently.
I would recommend realistic planning, and don’t plan too low-level. Unrealistic planning leads to demotivation when deadlines are missed, which in turn contributes to negative emotions discussed above.

Having a good relationship with your project supervisor is important, the regular meetings are a real help to guide you in the right direction. What I found was my way of thinking always had too much breadth rather than depth regarding the problems I was facing within the project. The project supervisor is there to help and helped me look at the problems from the correct angle. I would recommend using your supervisor for help as much as you can.

Overall I really did enjoy the project experience, I feel it has matured and developed vital skills that I now know I was weak on. I wish I could do it all again!
# Appendix B

## Appendix B - Original Project Schedule

![Original Project Schedule Diagram](image)

Figure B.1: Original Project Schedule.
Appendix C - Revised Project Schedule

<table>
<thead>
<tr>
<th>Task</th>
<th>Week number</th>
<th>April</th>
<th>May</th>
<th>Time taken (hrs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appendix C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meetings with sponsors</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Student preferences form</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Literature search meeting/learning</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete project form</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Questionnaire &amp; data gathering meeting</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mid-Project report</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Writing meeting</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Submit contents &amp; draft chapter</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Progress meeting</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subject final report</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Background research</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Research stops called/representations/applications</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write up QIR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Research aspects of RAC you are going to implement</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write up the above</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Research technologies/approaches I could use for solution</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write up a discussion of the above</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write up a draft in IIR methodologies</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design &amp; Development</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overall design</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design iteration 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design iteration 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design iteration 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write up the report</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finalise report</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Appendix C - Revised Project Schedule</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Original plan and completed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Original plan and not completed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Planned work</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partially complete</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overall details</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Completed and not planned</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure C.1: Revised Project Schedule.
Appendix D

Appendix D - Problem Definition

Questionnaire

In the projects initial phase a short questionnaire was written and presented to a subset of students within the School of Computing whom have studied Knowledge Representation and Reasoning. The purpose of this exercise was to validate the meaningfulness of the project. The following questions were presented:

Questions

1. Having studied Qualitative Spatial Reasoning as part-of the Knowledge Representation and Reasoning undergraduate module, can you remember studying spatial algebra? (If no don’t answer the questions below)

2. In your opinion would it have been useful to have an electronic educational tool which visualised Qualitative Spatial Descriptions you inputted?

3. If yes, can you think of any useful features that such a tool should have?

4. After studying the basics of the subject, imagine a user defining a set of spatial constraints that qualitatively describe a scene. Would a tool which fills-in relationships that were missed out
accidentally by the user be useful?

Results

Out of 20 students and 5 ex-students questioned:

<table>
<thead>
<tr>
<th>Question number</th>
<th>Student (S) or Ex-Student (E)</th>
<th>Result (#Yes:#No)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S</td>
<td>20:0</td>
</tr>
<tr>
<td>1</td>
<td>E</td>
<td>2:5</td>
</tr>
<tr>
<td>2</td>
<td>S</td>
<td>20:0</td>
</tr>
<tr>
<td>2</td>
<td>E</td>
<td>1:1</td>
</tr>
<tr>
<td>4</td>
<td>S</td>
<td>18:2</td>
</tr>
<tr>
<td>4</td>
<td>E</td>
<td>1:1</td>
</tr>
</tbody>
</table>

No-one contributed to answering question three, probably due to the qualitative nature of the question and the fact the students were busy when presented with the questionnaire. 95% of those that answered believes a tool which visualises a set of spatial descriptions would be useful. 86% of those that answered believes a tool which fills-in missed relationships would be useful. A basic analysis of the results above suggests the problem definition is justified and the meaningfulness of this project is just.
Figure E.1: SVG code generated using improved 1-d visualisation algorithm.
Appendix F

Appendix F - Questionnaire

F.1 Questionnaire

Introduction I have produced an application which visualises scenes described using the RCC-5 calculus. The RCC-5 calculus describes scenes qualitatively using the relationships between objects. I am now testing the software to get an idea of what the user thinks of functionality. I would appreciate it if you would follow the steps below, and provide some helpful feedback.

Testing

scene 1 = b PP a : c PP a : c PP b

scene 2 = a EQ b : c PO d : a DC c: a DC d : b DC c: b DC d : d PO c

Input scene 1:

What do you think of the visualisation in terms of functionality? Is it useful?

Any possible improvements?

Enter scene 2:
What do you think of the visualisation in terms of functionality? Is it useful?

Any possible improvements?

Enter a scene at your discretion:

What do you think of the visualisation in terms of functionality? Is it useful?

Any possible improvements?

**Ranking**

Please rank my software on a scale of 1-5, 1 being very poor, 5 being excellent for the topics below:

- Usability  is the software easy to use?
- Intuitiveness  is it obvious how to carry out any tasks?
- Attractiveness  does the software look good?
- Overall  what do you think of my software as it is overall?

**Your details**

Please answer the following questions about yourself:

Which age bracket do you fit into?
<15 | 15 to 20 | 20 to 30 | 30 to 40 | 40 to 50 | >50

How computer literate would you say you are?
Very Poor | Poor | OK | Good | Very Good

Have you studied knowledge representation and reasoning before?
No | Yes
Are you aware of qualitative spatial representation and reasoning as an area of research in AI?

Yes | No

Thank you.
Appendix G

Appendix G - Results from Testing

G.1 Iteration One

The nodes and relationships of each scene are presented. The configuration is then tested on different parts of the implemented algorithms within iteration one.

‘Create matrix’ refers to the execution of the CreateMatrix method within the RCC5Language class which creates the matrix from the scene. ‘Check matrix 1’ checks this matrix for validity using triangle checking, a log of incorrect relations is presented if any are incorrect. ‘Create matrix 2’ refers to the execution of the CreateMatrix method a second time passing through the same matrix, which should be corrected by the previous check matrix.

The visualise method within the Visualiser class is where the 1-dimensional visualisation algorithm sits. For testing purposes the functionality of this algorithm has been split into two separate parts. ”Visualiser:sorted” refers to using the correct matrix and creating a topologically sorted file containing start and end points. ‘Visualiser:visual’ refers to using this sorted file to visualise the scene.

If a scenes first tests expected and actual were different, the reason was explored and fixed. A second test using the fixed system was then performed. If a failure still existed due to time constraints the bug will have to stay in the system.
G.1.1 Scene one - Ordinary small scene using PP

Nodes: a b c

Relationships:

- ab = P Pi
- ac = P Pi
- ba = P P
- bc = P Pi
- ca = P P
- cb = P P

<table>
<thead>
<tr>
<th></th>
<th>Create Matrix</th>
<th>Check Matrix 1</th>
<th>Check Matrix 2</th>
<th>sorted</th>
<th>visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>s(a)s(b)s(c)e(c)e(b)e(a)</td>
<td>pass</td>
</tr>
<tr>
<td>Actual Result</td>
<td>pass</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
</tr>
</tbody>
</table>

Problems: Array indexes incorrect. This happened due to the matrix being null at first when it needed to contain an ''X’ in matrix[0][0]

<table>
<thead>
<tr>
<th></th>
<th>Create Matrix</th>
<th>Check Matrix 1</th>
<th>Check Matrix 2</th>
<th>sorted</th>
<th>visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>s(a)s(b)s(c)e(c)e(b)e(a)</td>
<td>pass</td>
</tr>
<tr>
<td>Actual Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
</tbody>
</table>

G.1.2 Scene two - Impossible scene

Nodes: a b c d

Relationships:

- ab = P O
- ac = PO
- ad = PO
- ba = PO
- bc = PO
- bd = PO
- ca = PO
- cb = PO
- cd = PO
- da = PO
- db = PO
- dc = PO

<table>
<thead>
<tr>
<th>-</th>
<th>Create Matrix</th>
<th>Check Matrix 1</th>
<th>Check Matrix 2</th>
<th>sorted</th>
<th>visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>none</td>
<td>pass</td>
</tr>
<tr>
<td>Actual Result</td>
<td>pass</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>-</th>
<th>Create Matrix</th>
<th>Check Matrix 1</th>
<th>Check Matrix 2</th>
<th>sorted</th>
<th>visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>none</td>
<td>pass</td>
</tr>
<tr>
<td>Actual Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>fail</td>
<td>fail</td>
</tr>
</tbody>
</table>

**G.1.3 Scene three - EQ test**

- **Nodes:** a b c d
- **Relationships:**
  - ab = EQ
  - ac = DC
- \( ad = DC \)
- \( ba = EQ \)
- \( bc = DC \)
- \( bd = DC \)
- \( ca = DC \)
- \( cb = DC \)
- \( cd = EQ \)
- \( da = DC \)
- \( db = DC \)
- \( dc = EQ \)

<table>
<thead>
<tr>
<th></th>
<th>Create Matrix</th>
<th>Check Matrix 1</th>
<th>Check Matrix 2</th>
<th>sorted</th>
<th>visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>s(ab)s(ab)s(cd)e(cd)</td>
<td>pass</td>
</tr>
<tr>
<td>Actual Result</td>
<td>pass</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
</tr>
</tbody>
</table>

Problems: \( s(ab) \) and \( e(ab) \) elements were written to the unsorted.text file incorrectly

<table>
<thead>
<tr>
<th></th>
<th>Create Matrix</th>
<th>Check Matrix 1</th>
<th>Check Matrix 2</th>
<th>sorted</th>
<th>visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>s(ab)s(ab)s(cd)e(cd)</td>
<td>pass</td>
</tr>
<tr>
<td>Actual Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
</tbody>
</table>

**G.1.4 Scene four - DC scene**

Nodes: \( a \ b \ c \ d \)

Relationships:

- \( ab = DC \)
- \( ac = DC \)
• ad = DC
• ba = DC
• bc = DC
• bd = DC
• ca = DC
• cb = DC
• cd = DC
• da = DC
• db = DC
• dc = DC

<table>
<thead>
<tr>
<th>-</th>
<th>Create Matrix</th>
<th>Check Matrix 1</th>
<th>Check Matrix 2</th>
<th>sorted</th>
<th>visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>s(a)e(a)s(b)e(b)s(c)e(c)s(d)e(d)</td>
<td>pass</td>
</tr>
<tr>
<td>Actual Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
</tbody>
</table>

**G.1.5 Scene five - Inputting invalid scene**

Nodes: a b c

Relationships:

• ab = EQ
• ac = EQ
• ba = EQ
• bc = DC
• ca = DC
- cb = DC

<table>
<thead>
<tr>
<th>-</th>
<th>Create Matrix</th>
<th>Check Matrix 1</th>
<th>Check Matrix 2</th>
<th>sorted</th>
<th>visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>none</td>
<td>pass</td>
</tr>
<tr>
<td>Actual Result</td>
<td>pass</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
</tr>
</tbody>
</table>

Problems: building the string of incorrect relationships failed

<table>
<thead>
<tr>
<th>-</th>
<th>Create Matrix</th>
<th>Check Matrix 1</th>
<th>Check Matrix 2</th>
<th>sorted</th>
<th>visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>none</td>
<td>pass</td>
</tr>
<tr>
<td>Actual Result</td>
<td>pass</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
</tr>
</tbody>
</table>

G.1.6 Scene six - PO scene

Nodes: a b c d

Relationships:

- ab = PO
- ac = PO
- ba = PO
- bc = PO
- ca = PO
- cb = PO
- cd = PO

<table>
<thead>
<tr>
<th>-</th>
<th>Create Matrix</th>
<th>Check Matrix 1</th>
<th>Check Matrix 2</th>
<th>sorted</th>
<th>visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>s(a)s(b)e(a)s(c)e(b)e(c)</td>
<td>pass</td>
</tr>
<tr>
<td>Actual Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
</tbody>
</table>
G.2  Iteration Two

G.2.1  Scene one - Ordinary small scene using PP

Nodes: a b c
Relationships:

- $ab = PP_i$
- $ac = PP_i$
- $ba = PP$
- $bc = PP_i$
- $ca = PP$
- $cb = PP$

<table>
<thead>
<tr>
<th></th>
<th>Create Matrix</th>
<th>Check Matrix 1</th>
<th>Check Matrix 2</th>
<th>sorted</th>
<th>visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>$s(a)s(b)s(c)e(c)e(b)e(a)$</td>
<td>pass</td>
</tr>
<tr>
<td>Actual Result</td>
<td>pass</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
</tr>
</tbody>
</table>

Problems: Array indexes incorrect. This happened due to the matrix being null at first when it needed to contain an ’X’ in matrix[0][0]

<table>
<thead>
<tr>
<th></th>
<th>Create Matrix</th>
<th>Check Matrix 1</th>
<th>Check Matrix 2</th>
<th>sorted</th>
<th>visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>$s(a)s(b)s(c)e(c)e(b)e(a)$</td>
<td>pass</td>
</tr>
<tr>
<td>Actual Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
</tbody>
</table>

G.2.2  Scene two - Impossible scene

Nodes: a b c d
Relationships:
- ab = PO
- ac = PO
- ad = PO
- ba = PO
- bc = PO
- bd = PO
- ca = PO
- cb = PO
- cd = PO
- da = PO
- db = PO
- dc = PO

<table>
<thead>
<tr>
<th>-</th>
<th>Create Matrix</th>
<th>Check Matrix 1</th>
<th>Check Matrix 2</th>
<th>sorted</th>
<th>visual</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Expected Result</strong></td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>none</td>
<td>pass</td>
</tr>
<tr>
<td><strong>Actual Result</strong></td>
<td>pass</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>-</th>
<th>Create Matrix</th>
<th>Check Matrix 1</th>
<th>Check Matrix 2</th>
<th>sorted</th>
<th>visual</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Expected Result</strong></td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>none</td>
<td>pass</td>
</tr>
<tr>
<td><strong>Actual Result</strong></td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>fail</td>
<td>fail</td>
</tr>
</tbody>
</table>

**G.2.3 Scene three - EQ test**

Nodes: a b c d

Relationships:

- ab = EQ
- ac = DC
- ad = DC
- ba = EQ
- bc = DC
- bd = DC
- ca = DC
- cb = DC
- cd = EQ
- da = DC
- db = DC
- dc = EQ

<table>
<thead>
<tr>
<th></th>
<th>Create Matrix</th>
<th>Check Matrix 1</th>
<th>Check Matrix 2</th>
<th>sorted</th>
<th>visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>s(ab)s(ab)s(cd)e(cd)</td>
<td>pass</td>
</tr>
<tr>
<td>Actual Result</td>
<td>pass</td>
<td>pass</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
</tr>
</tbody>
</table>

G.2.4 Scene four - DC scene

Nodes: a b c d
Relationships:

- ab = DC
- ac = DC
- ad = DC
- ba = DC
- bc = DC
bd = DC
ca = DC
cb = DC
cd = DC
da = DC
db = DC
dc = DC

dc = DC

<table>
<thead>
<tr>
<th>-</th>
<th>Create Matrix</th>
<th>Check Matrix 1</th>
<th>Check Matrix 2</th>
<th>sorted</th>
<th>visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>s(a)e(a)s(b)e(b)s(c)e(c)s(d)e(d)</td>
<td>pass</td>
</tr>
<tr>
<td>Actual Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
</tbody>
</table>

G.2.5 Scene five - Inputting invalid scene

Nodes: a b c

Relationships:

ab = EQ
ac = EQ
ba = EQ
bc = DC
ca = DC
cb = DC

<table>
<thead>
<tr>
<th>-</th>
<th>Create Matrix</th>
<th>Check Matrix 1</th>
<th>Check Matrix 2</th>
<th>sorted</th>
<th>visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>none</td>
<td>pass</td>
</tr>
<tr>
<td>Actual Result</td>
<td>pass</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
</tr>
</tbody>
</table>
Problems: bugs

<table>
<thead>
<tr>
<th></th>
<th>Create Matrix</th>
<th>Check Matrix 1</th>
<th>Check Matrix 2</th>
<th>sorted</th>
<th>visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>none</td>
<td>pass</td>
</tr>
<tr>
<td>Actual Result</td>
<td>pass</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
</tr>
</tbody>
</table>

G.2.6 Scene six - PO scene

Nodes: a b c d

Relationships:
- ab = PO
- ac = PO
- ba = PO
- bc = PO
- ca = PO
- cb = PO
- cd = PO

<table>
<thead>
<tr>
<th></th>
<th>Create Matrix</th>
<th>Check Matrix 1</th>
<th>Check Matrix 2</th>
<th>sorted</th>
<th>visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>s(a)s(b)e(a)s(c)e(b)e(c)</td>
<td>pass</td>
</tr>
<tr>
<td>Actual Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
</tbody>
</table>

G.2.7 Scene seven - incomplete scene with one incomplete realtionship

Nodes: a b c d

Relationships:
• \( ab = PPI \)
• \( ac = PPI \)
• \( ba = PP \)
• \( bc = PPI \)
• \( ca = \)
• \( cb = PP \)

<table>
<thead>
<tr>
<th>-</th>
<th>Create Matrix</th>
<th>Check Matrix 1</th>
<th>Check Matrix 2</th>
<th>sorted</th>
<th>visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>( s(a)s(b)s(c)e(c)e(b)e(a) )</td>
<td>pass</td>
</tr>
<tr>
<td>Actual Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
</tbody>
</table>

G.2.8 Scene eight - majaourly incomplete scene

Nodes: a b c d

Relationships:
• \( ab = \)
• \( ac = PPI \)
• \( ba = \)
• \( bc = PPI \)
• \( ca = \)
• \( cb = PP \)

<table>
<thead>
<tr>
<th>-</th>
<th>Create Matrix</th>
<th>Check Matrix 1</th>
<th>Check Matrix 2</th>
<th>sorted</th>
<th>visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Result</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
<td>( s(a)s(b)s(c)e(c)e(b)e(a) )</td>
<td>pass</td>
</tr>
<tr>
<td>Actual Result</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
<td>fail</td>
</tr>
</tbody>
</table>
Appendix H

Appendix H - Proposed Exception Structure

Here is the exception structure for the software architecture defined within chapter 3. Note the Exception class belongs to the native language the architecture is created with, and the custom exception classes are inherited from this.

Figure H.1: UML class diagram representing exception structure.
Appendix I

Appendix I - Additional Psuedo Code

Here is some pseudo code that was developed for the "checking correctness of a qualitative scene" algorithm 4.1.1.
**Program 4** Proposed pseudo code for building a relationship matrix from a description of a scene

SET startReference, endReference to 0
SET tempRelation, matrix[][] to null

//get human defined relations...
FOR each relation within the scene description
    SET tempRelation to relation
    SET start node to tempRelation start node
    SET end node to tempRelation end node
    SET relationship to tempRelation relationship

    IF tempRelation start node is not in the matrix
        COMPUTE expand matrix dimensions by one
        SET matrix[LENGTH-1][0] start node in the matrix
        SET matrix[0][LENGTH-1] start node in the matrix
    END IF

    IF tempRelation end node is not in the matrix
        COMPUTE expand matrix dimensions by one
        SET matrix[LENGTH-1][0] end node in the matrix
        SET matrix[0][LENGTH-1] end node in the matrix
    END IF

FOR each node in the matrix[0]
    IF node is startnode
        SET startReference to the matrix column index
    END IF
    IF node is endnode
        SET endReference to the matrix column index
    END IF

WRITE relationship in matrix[startReference][endReference]

    IF relationship is symmetric
        WRITE relationship in matrix[endReference][startReference]
    ELSE
        WRITE relationship inverse in matrix[endReference][startReference]
    END IF
END FOR
END FOR
**Program 5** Proposed pseudo code for a function which takes two edges of a "triangle" and uses that information with composition table to get the third edge

FUNCTION choose relation(relation 1, relation 2):

  SET n to the number of JEPD relationships in language + 1.
  SET array_of_possible_relations to null
  SET startReference, endReference to 0

  FOR i to n - 1
    IF matrix[i][0] is relation 1
      startReference = i;
    ENDIF
    IF matrix[j][0] is relation 2
      endReference = i;
    END IF
  END FOR

  SET array of possible relations to composition matrix[startReference][endReference]

  IF array of possible relations length is 1
    RETURN relation
  ELSE
    RETURN array of possible relations
  ENDIF

END FUNCTION