Investigation into an improved modular rule-based testing framework for business rules

Jodie Wetherall

For the award of

Doctor of Philosophy

for which the thesis is submitted in partial fulfilment of its requirements

University of Greenwich

October 2010

Declaration

I certify that this work has not been accepted in substance for any degree, and is not concurrently being submitted for any degree other than that of Doctor of Philosophy being studied at the University of Greenwich. I also declare that this work is the result of my own investigations except where otherwise identified by references and that I have not plagiarised another's work

Student: JODIE WETHERALL

Signature:

Date:

Supervisor: Dr Steve Woodhead

Signature:

Date:

Acknowledgments

First and foremost I would like to express my deepest gratitude to my supervisor, Dr Steve Woodhead, for firstly agreeing to supervise a PhD of this nature but most importantly for providing the encouragement and space to work through this process over the nine long years that it has taken. In particular I would like to thank Steve for his work in proof reading the thesis which resulted in a document to be proud of.

My academic career has evolved significantly over the duration of the PhD. As a Senior Lecturer at the School of Engineering, University of Greenwich, I have been privileged to work with a number of very supportive colleagues who have offered encouragement and advice along the way. Thank you in particular to David Israel, Mike Sharp, David Armour-Chelu, Ndy Ekere and Alan Reed.

There have been many changes to my academic role over the years, from developing new programmes and courses to the engineering of software systems to support the School, each of these things has provided the opportunity for personal development which has contributed toward the success of my PhD. Not forgetting the countless students who I have had the pleasure of supporting graduate during this time. In particular the class of 2009 who provided me with the inspiration to push forward and complete when enthusiasm was running dry. Thank you all.

A lot has happened over the course of this PhD which makes tackling it part-time a particular challenge. Firstly I got married and my wife and I bought a new house, two of the most stressful things to do in life, within the same year. We then became foster carers for several years before deciding to expand our own family from 3 to 6 members. Whilst my hair is yet to recede, the gray is now showing. For the most enjoyable rollercoaster ride and the perfect distraction I would like to thank my wife, Jocasta, and our 4 children, Jacob, Sophie, Mae and Elena.

Abstract

Rule testing in scheduling applications is a complex and potentially costly business problem. This thesis reports the outcome of research undertaken to develop a system to describe and test scheduling rules against a set of scheduling data. The overall intention of the research was to reduce commercial scheduling costs by minimizing human domain expert interaction within the scheduling process.

This thesis reports the outcome of research initiated following a consultancy project to develop a system to test driver schedules against the legal driving rules in force in the UK and the EU. One of the greatest challenges faced was interpreting the driving rules and translating them into the chosen programming language. This part of the project took considerable effort to complete the programming, testing and debugging processes. A potential problem then arises if the Department of Transport or the European Union alter or change the driving rules. Considerable software development is likely to be required to support the new rule set.

The approach considered takes into account the need for a modular software component that can be used in not just transport scheduling systems which look at legal driving rules but may also be integrated into other systems that have the need to test temporal rules. The integration of the rule testing component into existing systems is key to making the proposed solution reusable.

The research outcome proposes an alternative approach to rule definition, similar to that of RuleML, but with the addition of rule metadata to provide the ability of describing rules of a temporal nature. The rules can be serialised and deserialised between XML (eXtensible Markup Language) and objects within an object oriented environment (in this case .NET with C#), to provide a means of transmission of the rules over a communication infrastructure. The rule objects can then be compiled into an executable software library, allowing the rules to be tested more rapidly than traditional interpreted rules. Additional

support functionality is also defined to provide a means of effectively integrating the rule testing engine into existing applications.

Following the construction of a rule testing engine that has been designed to meet the given requirements, a series of tests were undertaken to determine the effectiveness of the proposed approach. This lead to the implementation of improvements in the caching of constructed work plans to further improve performance. Tests were also carried out into the application of the proposed solution within alternative scheduling domains and to analyse the difference in computational performance and memory usage across system architectures, software frameworks and operating systems, with the support of Mono.

Future work that is expected to follow on from this thesis will likely reside in investigations into the development of graphical design tools for the creation of the rules, improvements in the work plan construction algorithm, parallelisation of elements of the process to take better advantage of multi-core processors and off-loading of the rule testing process onto dedicated or generic computational processors.

Table of Contents

Table of Figures

1 INTRODUCTION

This thesis has been constructed to summarise the results of research into a rule based testing system for use in scheduling applications. The document will begin by looking at the current research areas that fall into or around this research domain in order to define the state of the art in this area and identify the target original contributions for the thesis.

1.1 Background

The problem domain which this research tackles was spawned from a consultancy project undertaken with Kings Ferry Coaches Ltd (KFC) (Wetherall, 2002), a local award winning Coach Company. Where possible KFC try to implement technology to automate processes within its organisation with the long-term view of reducing the overall cost of running the business whilst maintaining a high quality level of service to their customers.

One area of the company's business which they have had the most difficulty in automating is the scheduling of the vehicles and drivers to their appointments. The company employ a relatively highly paid domain expert to carry out the scheduling process whilst taking into account the legal driving rules (Transport, 1998) which all drivers of domestic passenger journeys are legally bound by. The cost of doing this is fairly substantial with a high turnover of experts in short periods of time.

The company have conducted their own investigations into existing solutions for automating this scheduling process but have found that nothing exists to schedule the drivers and vehicles to appointments whilst, at the same time, taking into consideration the legal driving rules. This is mainly because the legal driving rules are a complex set of detailed and varied rules which have to take into consideration multiple sets of temporal data.

1 INTRODUCTION

The initial project took on the approach of using a Genetic Algorithm (GA) to carry out the scheduling process. GA's are renowned for their ability to schedule timed data in an efficient way and, at that time of the consultancy project, research identified this to be the most appropriate way forward. The system was called the Genetic Allocation System (GAS) relating to the scheduling algorithm used.

One of the greatest challenges faced during the project was taking the rules, as described by the Department of Transport (Transport, 1998), and implementing them in Visual Basic 6 (the language previously chosen for prototyping the solution for the consultancy project). During the scheduling process, these rules are called upon to ensure the legal driving rules have been adhered to. This part of the project took a long time to complete and required a considerable amount of testing in order to prove the implementation met the rules both without bugs and to the correct interpretation of the legal rules.

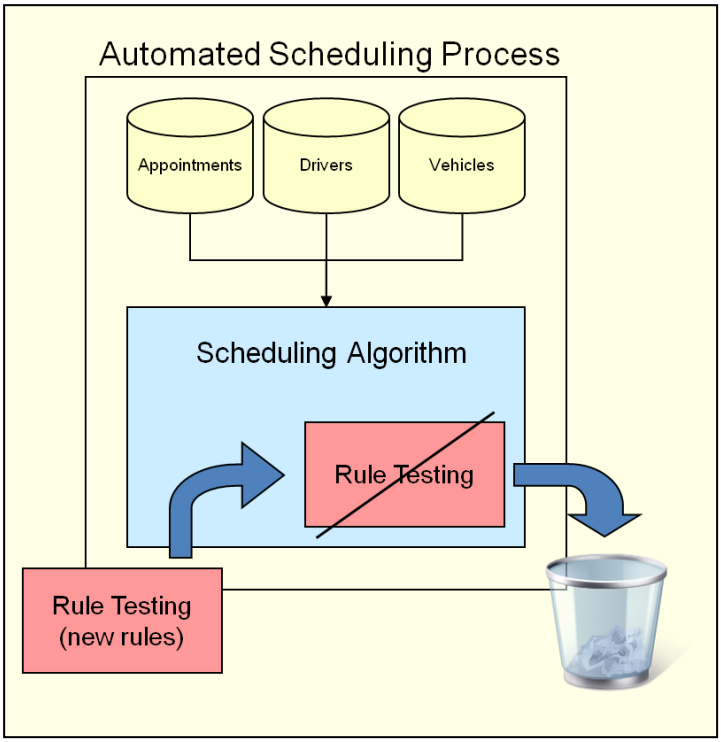


Figure 1-1 - A diagram illustrating the problem with embedding the rule testing algorithm within the scheduling process.

As illustrated in Figure 1-1, the problem now comes when the Department of Transport decide to alter the legal driving rules or change them completely to an alternative scheme. This will require further months of work for a software developer to implement a new set of driving rules in code which, for a coach company, will significantly compromise the business case for the software investment.

The outcome of this research has been designed to solve this problem. By proposing a solution which will allow either the Department of Transport or the company itself to describe the rules in a human readable form and let the translation of the rules equate to computer executable code for fast testing.

Solutions exist to do similar types of things to that proposed, but none are suitable in bridging the gap between rule-based testing and scheduling.

The outcome of this research is not specifically focused on scheduling, but rather, rule-testing in scheduling. Part of the problem in the separation of the two problem domains is that rule-testing rarely considers the continuous nature of scheduling and the fact that events are timed and need to be aligned in a multi-dimensional way. More often rule testing looks at rules in a simplistic sense of the data and the rule to apply and does not consider the time sensitive nature of scheduling in its approach.

For clarity, this thesis looks at business rules. (Ambler, 2004) states that a business rules "defines or constrains one aspect of your business that is intended to assert business structure or influence the behaviour of your business", such as the legal driving rules within the transport scheduling scenario. Some example business rules provided by (Ambler, 2004) include:

- Tenured professors may administer student grades.
- Teaching assistants who have been granted authority by a tenured professor may administer student grades.
- All master's degree programs must include the development of a thesis.

One of the main issues when looking at rule-engines, is that a rule engine tends to require a considerable amount of additional logic in order to understand how to deal with the rules and data being tested with it. For some applications this approach can be fairly complex, especially where the rule provider is not an expert in software development, which defeats the idea of using an 'automated' system for scheduling and rule testing. This thesis considers the need for a simple modular approach when defining a rule engine in order that the proposed solution can easily be integrated within a range of host systems without the need for a large quantity of integration logic.

1.2 Objectives

The objectives of this thesis are to:

- investigate the existing domains within which the topic of this thesis spans to help identify the requirements for a rule testing system for use within a scheduling environment.
- propose a mechanism by which rules can be changed within a scheduling process without the need for the redevelopment of the scheduling algorithm or the redevelopment of further software.
- identify a means of improving the computational performance of the rule testing process by compiling rules into executable code.
- identify the required metadata to associate with a set of rules to support the process of organising a set of data into a work plan.
- define a simple interface to support the integration of the proposed solution within existing or future scheduling systems by designing the solution to be modular.

These objectives are looked at in greater detail within the Requirements chapter.

1.3 Domain Boundary

To summarise the scope of the problem it is important to identify the domain boundary which allows the research topic to restrict the areas of investigation.

To refresh, the research is titled "*Investigation into an improved modular rule-based testing framework for business rules*". The output of the research aims to be relatively generic and adaptable to fit into a number of different systems hence the need to make it Modular. This project aims to look specifically at rule testing for scheduling applications. Many research areas (see Section 0) have previously looked at generic rule testing approaches but this research takes into account temporal data used in scheduling along with the need for a high computational performance for a continuous scheduling process.

1.4 Contribution

This research is considered to provide a number of contributions within its field. The following bullet points summarise the research contributions:

- This research will bridge the gap between rule testing and scheduling systems providing a solution where previously an appropriate solution has not existed.

- It will describe a potential new rule testing definition language that has been designed around the need for describing rules which relate to temporal problem domains such as scheduling.

- It will demonstrate a high performance approach to rule testing by compiling rules to optimise the speed for frequent testing scenarios such as scheduling.

- It will provide an example approach to modularising the rule testing engine in a way which will allow it to be fully integrated with existing systems.

- It will define a method of benchmarking the resulting solution in order that further improvements to future developments have a means of comparison.

1.5 Document Structure

This thesis has been divided into a number of sections to aid in its readability.

Literature Review

The Literature Review illustrates the extensive research undertaken within the various distinct research areas surrounding the domain of this research. The purpose of this section is to establish where existing research has been undertaken and to then identify the target original contributions for the work reported in this thesis.

Requirements

The Requirements chapter unpacks and defines the problem for which this research aims to provide a solution. It set outs to clearly identify the boundary of the problem domain with the aid of user and system requirements.

Inception & Elaboration

The Inception & Elaboration chapter shows the outcome of the first two phases of the Rational Unified Process (RUP), the methodology used to help solve this research problem. This chapter shows the evolution of the requirements as they develop into elements of software that can be use to demonstrate sections of the overall solution.

Construction

The Construction chapter looks at the construction phase of the RUP, illustrating how the outcome of the inception and elaboration phases are brought together and a working prototype of the candidate solution is realised.

Testing

The Testing chapter looks at the overall testing process for the candidate solution as well as the design of an automated test harness designed for carrying out the testing process.

Initial Results

The Initial Results chapter provides some discussion of the results obtained from the initial testing process, considering the computational performance and memory usage of the prototype solution as well as its measurable organisational value.

Further Results

The Further Results chapter expands on the previous Testing and Initial Result chapters by testing the performance of the proposed solution within an alternate rule testing domain. It also considers the solutions performance under various conditions such as across multiple platforms and execution runtimes. The Further Results chapter also demonstrates the ability to debug rules to eliminate rule definition errors.

Conclusion

The Conclusion chapter draws together and summarises the results obtained throughout the testing process, comparing the outcome with the contributions outlined in this Introduction.

References and Appendices

The References chapter provides a list of citations made throughout the thesis and the Appendices provide a range of additional information from a set of code listings through to the raw data obtained during testing.

1.6 Summary

This introduction has attempted to summarise the purpose of this research, providing a background to the problem domain and giving context to the target original contributions.

Most importantly, the introduction has outlined the original contributions made by this research. The goal for the rest of this document is to demonstrate that these contributions are novel, to show how the contributions have been made and to suggest ways of moving forward other research within this field.

2 LITERATURE REVIEW

2.1 Introduction

As discussed in Section 1.1, the problem the research in this thesis endeavours to present a solution for originated from a consultancy project (Wetherall, 2002). This project required the scheduling of drivers and vehicles to appointments within a transport scheduling environment. The client had, themselves, searched for an existing system but resorted to academic support when a solution could not be found.

The problem the client had was not that a scheduling system could not be sourced to support their activity, but the scheduling process undertaken by the existing scheduling systems (specifically the PHC system already in use at Kings Ferry) were not able to take into account the legal driving rules. Similarly, the client was able to source rule testing systems to support the testing of drivers schedules against the legal driving rules (Transport, 1998), however, these system were unable to perform the required scheduling task and were not designed with the kind of optimisation required for an already computational expensive scheduling process.

The initial project resulted in the implementation of a Genetic Algorithm (GA) for carrying out the scheduling activity whilst the legal driving rules were carefully interpreted directly into the code of the application. This approach was fine, at first, until changes were made to the driving rules resulting in a need for significant further development time.

It was at this point that the research reported in this thesis began, in order to look at a method that would limit the amount of additional development required each time a set of rules were to change. The first step in this research was the completion of the literature survey described in this chapter.

2 LITERATURE REVIEW

This literature review has been conducted to determine the current state of research in a numbers of areas which fall within the problem domain. It is not until the current state of research is better understood that it is possible to look at alternative approaches to solving the rule testing problem in transport scheduling environments.

The literature review has been divided into a number of different sections as the topic of interest incorporates a number of distinct areas.

- The research starts by breaking down the scheduling process currently used within the solution developed for the consultancy project, specifically looking at the GA, in order to understand the importance of testing the scheduling rules at optimal speeds.

- The second area investigated looks at the field of transport scheduling; the domain which best seems to contain the problem this research attempts to solve.

- The third area expands the investigation into the more general field of scheduling by looking at the current state of generic scheduling research, not specific to transport scheduling.

- The fourth area investigates the field of rule-based testing in order to determine current approaches to testing rules, in an attempt to identify an approach suitable for the transport scheduling domain.

- The fifth area investigates the field of software execution, in terms of dynamic compilation, scripting and interfacing, to determine current practices of gaining optimal performance of execution with a view to optimising the performance of a rule-based testing solution for use in transport scheduling.

2.2 Genetic Algorithms

As previously stated the consultancy project from which this research has spawned used a GA as its scheduling mechanism. A GA is "a computational model of biological evolution" (Forrest, 1996). Such algorithms get their name from the theory of manipulating the genetic makeup of a being to produce a superior version of that being. In this specific application, if there were a number of work schedules the GA will take these and try to combine and manipulate them to produce a better, more fitting work schedule. By continually manipulating a set of work schedules with each other it is possible to improve the final result.

2.2.1 Genetic Algorithms In Relation To GAS

This sub-section describes how the GA was used as part of the scheduling process in the Genetic Allocation System identified in Section 1.1. Detail provided in section 2.2.1 has been derived from (Wetherall, 2002).

A gene describes an individual element of a work schedule, in this case, an appointment and its driver and vehicle allocations. Figure 2-1 illustrates the elements of a gene. It shows that a gene represents a single appointment and the appointment has drivers and vehicles associated with it.

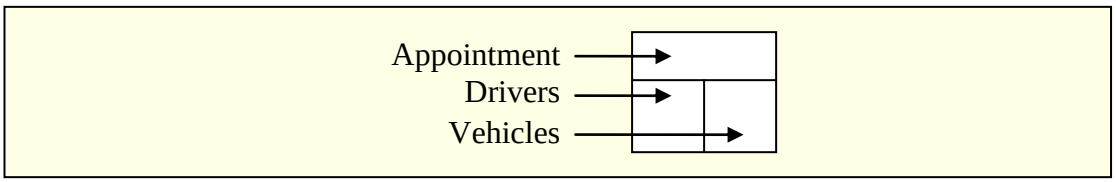


Figure 2-1 – Gene Structure

A chromosome describes a daily work schedule and is a collection of all of the Genes for any given day. Figure 2-2 shows the layout of a chromosome.

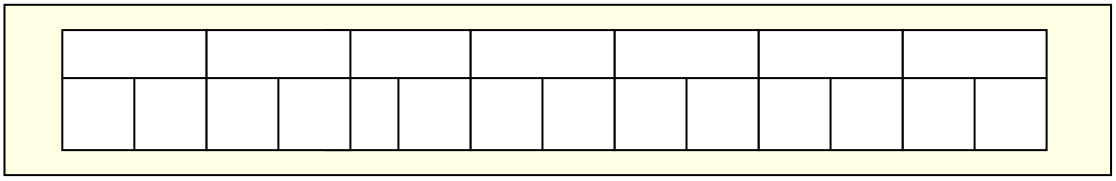


Figure 2-2 – Chromosome Structure

One of the key elements to the GA is the use of fitness functions as a tool to measure a work schedule in order to determine its suitability. Once a chromosome has been produced its fitness is measured in order to make a comparison against other chromosomes developed for a given day, to determine which is the most suitable. This level of fitness is based upon predetermined criteria and, in this example, there are 7 factors that contribute to the level of fitness of a chromosome.

The way the GA sequence works is by continuously performing a sequence of operations to try and improve upon the previous best schedule. Figure 2-3 shows the sequence undertaken in order to achieve this.

Initial number of chromosomes to generate (A) = 50
Working chromosome set size (B) = 10
Mutating frequency (C) = 20%

Step 1
Generate (A) pre-evolved chromosomes and evolve them into allocated chromosomes.

Step 2
Test all chromosomes for their fitness values.

Step 3
Remove the lowest fitness chromosomes so that the best (B) remain.

Step 4
Enter a loop to begin a repetitive cycle of actions.

Step 5
Keep the best chromosome to one side, in case we cannot find anything better than this.

Step 6
Perform a genetic cross-over on pairs 1-2, 3-4, 5-6, 7-8, 9-10 by randomly chopping the chromosomes into two at a random location and attach the first

chromosomes beginning with the second chromosomes end, and the second chromosomes beginning with the first chromosomes end to produce 2 new chromosomes. If the cross-over fails (i.e. produces an invalid chromosome) for any pair of individual chromosomes, generate new evolved chromosomes to replace them. Add the resulting 10 crossed-over chromosomes to the best chromosome kept from the previous cycle.

Step 7

Every (C) of the cycles of the loop, at a random point, mutate elements of the chromosome, for example, changing a driver for a different driver. This will cause a mutation that could be a positive or a negative reaction. If the mutation fails, put the element that was altered back to the way it was before we tried mutating.

Step 8

Test all chromosomes for their fitness values and remove the lowest fitness chromosomes, leaving (B) to continue to manipulate.

Step 9

Loop back to Step 4 until the user decides that they do not wish to continue (by seeing no improvement over time in the level of fitness).

Figure 2-3 – Genetic Algorithm Process

The approach described throughout section 2.2.1 works efficiently in many applications and situations, but the challenge comes when there is little or no slack in the process. For example part of the problem identified in using this approach in the consultancy project was that there were not enough drivers to choose from, i.e. all of the drivers were busy with an appointment at some point throughout a scheduled day. Trying to operate a scheduling system with tight restrictions such as these is very challenging.

It is also important to note at this point that the rules requiring testing are not related to the fitness functions. These fitness functions are simply used to determine, during the scheduling process and after rule testing has confirmed that a schedule meets the legal driving rules, which is the best work schedule from those produced.

2.2.2 Other Genetic Algorithm Approaches

Although GA's are not the core of this research it is the application which takes advantage of such scheduling approaches that may benefit the most from having a rule-testing mechanism geared toward it's application area.

(Tongchim & Chongstitvatana, 2002) describes an approach in optimising the performance of a parallel GA by altering its parameters based upon its observed performance. With an algorithm such as this which is working hard to improve the overall performance of the scheduling system there would be substantial benefit to have an accompanying rule testing mechanism that was also optimised for speed, understand the temporal nature of scheduling tasks and with the flexibility for integration.

There are a number of papers that look at using distributed GA's to improve the computational performance of the scheduling process but if any of these approaches also require the need for rule testing as part of the scheduling cycle then that too would need optimising. These research papers include (Alba & Troya, 2002), (Pavel, Ivan, & Jan, 1996), (Pinto., Monterio, & Rosa, 1999) and (Goncalves, Mendes, & Resende, 2008).

2.2.3 Summary

GA's are a common and often used tool for carrying out the scheduling process in various problem domains. Research is continually being undertaken to get more from these algorithms and specifically improve their performance. It is clear from reading these papers that performance is a key issues and something that a rule-testing solution, that is embedded into a scheduling process, needs to take into consideration.

2.3 Transport Scheduling

Whilst transport scheduling was the domain in which this research spawned, the aim of this research is not only limited to this domain. There are a variety of applications areas which can benefit from rule testing in scheduling such as timetable scheduling for educational institutions which have to consider contractual teaching rules.

The area of transport scheduling is vast and covers a wide range of activity areas including the management of the transport infrastructure (for example roads, railways, airways, waterways, canals, pipelines, airports, railway stations, bus stations and seaports), vehicles (for example automobiles, bicycles, buses, trains and airplanes) and operations (for example traffics signals, ramp meters, railroad switches, air traffic control, tolls and gasoline taxes).

At first glance, one paper which appears to be fairly close to this research topic is (Weerdt, 1999). This paper takes a mathematical look at approaches to resolving a transport scheduling problem. The specific problem examined in the paper, however, is based upon the movement of a product from Point A to Point B as opposed to the movement of passengers and vehicles which has a significantly different set of governing regulations, although both require the consideration of driving rules. Surprisingly the paper does not discuss how the legal driving regulations for the driver of the goods vehicle impacts on the process. Instead it focuses on the specific algorithmic scheduling method which could be used for this type of scenario.

One of the most relevant papers to this research topic is the paper (Wren, Fores, Kwan, Kwan, Parker, & Proll, 2002), which summarises the process of scheduling drivers to bus and train jobs by means of shift patterns. According to the paper the research group has many years of experience in driver scheduling for bus and train shifts but it does not discuss the approach taken to manage the labour rules, although they are highlighted as part of the problem.

Instead the group's research focuses on scheduling using shift patterns and the bus and train scenarios and the rules which accompany the scheduling process they use are a secondary issue. (Kwan, Kwan, & Wren, 2001) and (Fores & Proll, 1998) explain a number of approaches the research group have looked at in optimising the scheduling algorithms used which include Integer Linear Programming (ILP) and Relief Chains. The majority of approaches considered are variants of the ILP concept which, similar to the problem previous highlighted, tend to embed the specific rules into the scheduling algorithm which would eventually cause issues if the rules themselves were to later change.

In the paper (Freling, Huisman, & Wagelmans, Models and Algorithms for Integration of Vehicle and Crew Scheduling, 2003), the authors take a look at combining the overall scheduling process for the vehicle and crew and propose a mechanism for scheduling these together as opposed to the independent scheduling of one followed by the assignment of the other. Whilst this is an interesting papers that does consider many of the potential transport scheduling issues it is admitted that the rules which govern driver scheduling have not been considered as they provide additional complication to the overall scheduling process. This is the type of research which highlights the need for a modularised rule-testing solution that will integrate with existing systems.

Another paper to come from this research group is (Freling, Huisman, & Wagelmans, Applying an Integrated Approach to Vehicle and Crew Scheduling in Practice, 2000), which provides further details regarding the approach taken in looking at the rule testing side of the driver scheduling problem. The approach taken in the paper is to consider the legal driving rules as part of the algorithmic process of producing the scheduling which leads to a number of restrictions. Firstly the rules cannot be updated without also considering the redesign of the overall scheduling process so that if the rule issuing authority changed the legal driving rules the system would no longer be valid; secondly the rule testing process cannot be detached from the specific driver scheduling algorithm and reused in an alternative scheduling system,

which also requires rule testing for temporal style rules. This helps to identify the challenges to be addressed.

Another paper which looks at transport scheduling problems is (Fischetti, Lodi, Martello, & Toth, 2001). This paper takes a look at alternative mathematical approaches used in describing simplified parts of transport scheduling and, as with many of the other papers discussed, does not tend to look at the rule testing element in any detail.

This review of recent transport scheduling research identifies an obvious gap between rule testing and transport scheduling. Many more research papers can be identified that continue toward a solution to transport scheduling problems, all without the specific objective of optimising the rule testing element of the scheduling problem. Examples of other papers include (Martijn, van der Heijden, & van Harten, 2007), (Peters, de Matta, & Boe, 2007) and (Malachy & Crawford, 2007).

2.3.1 Summary

The research currently being undertaken within the transport scheduling field illustrates an ideal opportunity for a new potential solution to fill this gap, with a method of dealing with rules in a scheduling system. There may, however, be existing solutions available outside of the transport scheduling domain, therefore, the scope of research needs expanding to consider other areas of scheduling.

2.4 Generic Scheduling

The scheduling problem is not specific to a particular industry but can be seen as an issue for project managers in allocating tasks to personnel and plant through to the creation of timetables for bus, train and coach scheduling, which is the direction from which this work is being driven.

(Burke & Petrovic, 2002) presented a summary of the state of the art in timetabling research providing a useful grounding in the field of timetabling. Wide ranges of scheduling approaches were discussed providing numerous potential alternatives to the use of GA's used within the prior consultancy work. The approaches summarised in this paper focus on the algorithmic process of timetabling, leaving enough scope for the integration of a rule-testing system into a range of the algorithms discussed (if appropriate). It may be the case, such as with the constraint-based scheduling, metaheuristics and objective functions, that the rules themselves form part of the scheduling algorithm and are not separately distinguishable. In these cases an external rule testing component may not be appropriate. However, the flaw with these types of approaches, where the rules themselves are embedded within the algorithm, is that if the rules were to change, an expert in defining a new algorithm for the new rules will need to be employed, at a relatively high cost to an organisation, in order to bring the new set of rules into the scheduling process.

Real-time scheduling is an advancement made in real-time computation technology allowing more processes per second and, in turn, the ability to schedule in real-time. A recent paper (Lu, Stankovic, Tao, & Son, 2001) looks at a means of mathematically modelling a real-time scheduling process, taking feedback during the various stages and feeding it back in to the process.

Another progressing form of technology is Grid computing (Dail, 2002) which looks at taking advantage of large-scale parallel computing to carry out tasks at greater speeds. One of the main challenges with this type of approach is the need for a specialist distributed computing configuration which would not work for a normal organisation with standard computing equipment.

One approach, which was considered early in the literature review, looked at an alternative mathematical means of identifying the solutions to scheduling problems (Beck & Fox, 1998). It took the approach of using constraints to narrow down the scope of possible solutions. Whilst extremely useful, this would provide an alternative to the GA as opposed to providing a solution to

the problem of changing rulesets in the rule testing phase of the scheduling process.

Following on from this, another constraint-based scheduling paper (Liebowitz, 1997) looked at a prototype system developed using the constraint based approach to scheduling. The work carried out is now a little out of date but it would be a useful starting point to build on if scheduling were to be part of this research.

One research group (Subramanian, Katz, & Franklin, 2000) looks at ways of grouping together common types in the scheduling process. Their work aims to show that, by aggregating the scheduling tasks in this way, they can improve the throughput of scheduling tasks. After some investigation into the results it seems that this technique, although interesting, mainly applies to scheduling activities which need to be carried out immediately, like the scheduling of processes in a computer system or the control of bandwidth in large-scale network systems.

On investigation into existing scheduling research, looking specifically at systems which carry out the scheduling process, (Bartak, 2003) was of particular interest as it discussed a solution which used constraint based scheduling (Beck & Fox, 1998) to carry out the scheduling process. One of the elements missing from this research was identified as the testing of business rules as part of the scheduling process.

It is not uncommon for scheduling systems to be developed for a specific industrial sector. The paper (Cesta, Oddi, & Susi, 2000) describes a scheduling system produced called OO-scar whose purpose is to schedule tasks for the Italian Space Agency. This paper helps identify the need for specialist scheduling systems amongst specific industrial sectors.

The article (Green, 2001) contained a list of twenty scheduling systems which meet part of the requirements of a scheduling system but, as found in many of the research papers, the systems tend to be designed for a specific task and are

not adaptable for scheduling different sets of activities and they do not offer the flexibility to incorporate fast rule based testing into their processes.

A specific example that identifies a need for rule testing in its scheduling process is identified in (Abramson & Adela, 1991). The school timetabling problem has to take into consideration the rules set by the school themselves in terms of the time periods for the beginning and end of the teaching session and the lunch breaks but also the union agreed teaching loads for teachers. These rules are complex, and without a dedicated mechanism of updating these in an automated system, when the rules are changed, the system cannot properly perform the scheduling process.

2.4.1 Summary

Whilst looking at the field of generic scheduling approaches, it was clear that there was a divide between the approaches taken to manage and solve the scheduling problem and the ability to test rules as part of this process. Even in the paper (Proth, 2007), which attempts to summarise the current state of research in the field of scheduling, the application of rules to this process isn't taken into consideration.

It may be that in many cases the testing of business rules as part of the process isn't a bottleneck or the rules used do not require regular change. This does identify scope for this research to undertake further investigation.

2.5 Rule-Based Systems

The rule testing problem has been extensively researched in the past with contributions still being highlighted in a variety of areas. Rule testing engines have been developed that provide the opportunity for software developers to incorporate the testing of rules into a number of different application areas to solve the rule testing problem.

The concept of the rule testing engine has moved from research to commercial product status with a number of leading companies offering competitive rule testing engines which either come fully integrated within their applications (such as the E-mail rules component of Microsoft Outlook) or are ready for integration with host applications (such as Oracle Business Rules).

The contributions identified in this area are not in the development of a new rule testing engine although the means of describing temporal rules for the benefit of scheduling systems and a mechanism of testing temporal rules with a temporally aware rule testing engine that is flexible enough to integrate with existing systems is necessary in order to solve the problems outlined in Section 1.4.

(Hayes-Roth, 1985) provides a good introductory starting point for understanding rule based systems although it does tend to look at inference rules, with an if-then approach, such as if the condition is met then perform an action as a result. This approach differs from the idea of rule testing as required by the legal driving rules (Transport, 1998) as these rules require no action; if the driving rules are broken then the driver cannot drive with the allocated schedule. Within the driver scheduling domain there is no feedback process for the rules or any process by means of actions occurring. The idea behind this research is that driver schedules may be produced by means of, for example, GA's, and then they must be tested against the legal driving rules in order to determine if the schedule is legal or not. This is not to say that the rule test occurs at the end of the scheduling process as it may occur as part of the process in determining if it is possible to allocate a driver to a given job.

The other thing that the paper does not look at is the concept of temporal rules where one appointment is followed by another which then may be followed by rest time. Without this consideration the rule testing process for scheduling applications becomes complex for the scheduling algorithm developer who has to take on board a rule testing engine and provide additional functionality to cater for those temporal style rules.

(Mak & Blanning, 2003) provides a mechanism for understanding how to interpret the business rules used within the business process in order to determine what the rules are that may need automation. The process would be appropriate for a Systems Developer who may take on board the process of developing a scheduling system that uses the rule testing approach described in this research in better understanding what the initial business rules are.

A closely related research area to the one being proposed is the area of RuleML. RuleML is a markup language used to describe rules for rule testing systems. RuleML is substantially supported by the research community in defining a shared Rule Markup Language with a large number of papers published that support its overall goal (RuleML). It was spawned as the result of the movement toward the Semantic Web (Shadbolt, Berners-Lee, & Hall, 2006), the concept that allows data to be shared and reused across applications, enterprise and community boundaries. The underlying technology that has made RuleML possible is the eXtensible Markup Language (XML) (Bray, Paoli, Sperberg-McQueen, & Maler, 2000), a flexible technology for describing data.

RuleML provides the strong basis for rule description permitting both forward (bottom-up) and backward (top-down) rules in XML. It is based on the XML standard resulting in rules which are transportable over the Internet and are in a nearly human readable format.

```
<?xml version="1.0" encoding="UTF-8"?>

<RuleML
xmlns="http://www.ruleml.org/1.0/xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ruleml.org/1.0/xsd
http://www.ruleml.org/1.0/xsd/datalog.xsd"
>

  <Assert mapClosure="universal">

    <!-- This example rulebase contains four rules. The first and second rules
    are implications; the third and fourth ones are facts. -->


    <!-- The first rule implies that a person owns an object if that person
    buys the object from a merchant and the person keeps the object. -->

    <Implies>
```

```
      <if>
        <!-- explicit 'And' -->
        <And>
          <Atom>
            <op><Rel>buy</Rel></op>
            <Var>person</Var>
            <Var>merchant</Var>
            <Var>object</Var>
          </Atom>
          <Atom>
            <op><Rel>keep</Rel></op>
            <Var>person</Var>
            <Var>object</Var>
          </Atom>
        </And>
      </if>
      <then>
        <Atom>
          <op><Rel>own</Rel></op>
          <Var>person</Var>
          <Var>object</Var>
        </Atom>
      </then>
    </Implies>

    <!-- The second rule implies that a person buys an object from a merchant
    if the merchant sells the object to the person. -->

    <Implies>
      <if>
        <Atom>
          <op><Rel>sell</Rel></op>
          <Var>merchant</Var>
          <Var>person</Var>
          <Var>object</Var>
        </Atom>
      </if>
      <then>
        <Atom>
          <op><Rel>buy</Rel></op>
          <Var>person</Var>
          <Var>merchant</Var>
          <Var>object</Var>
        </Atom>
      </then>
    </Implies>


    <!-- The third rule is a fact that asserts that John sells XMLBible to
    Mary. -->
    <Atom>
      <op><Rel>sell</Rel></op>
      <Ind>John</Ind>
      <Ind>Mary</Ind>
      <Ind>XMLBible</Ind>
    </Atom>


    <!-- The fourth rule is a fact that asserts that Mary keeps XMLBible. -->
    <Atom>
      <op><Rel>keep</Rel></op>
      <Ind>Mary</Ind>
      <Ind>XMLBible</Ind>
    </Atom>

  </Assert>

</RuleML>
```

Figure 2-4 - Simple RuleML example describing the rules involved in the relationship between a buyer and a seller

The example RuleML (Boley, Paschke, Tabet, & Grosof, 2010) presented in Figure 2-4 demonstrates a set of 4 rules that describe the relationship between a merchant (the seller) and a person (the buyer). The first 2 rules are implications that help to describe the relationship between the merchant and the person and the last 2 rules are the fact, or data, that can be validated against. The example RuleML can be validated, using a schema validator (Thompson, Tobin, & Connolly, 2005), to determine whether the rules can be broken, without the need for an inference engine, however simple tests show that this approach can be slow.

One of the major benefits provided in using XML as a foundation for a rule description language is its simple integration with modern programming languages like Java and C#. Objects in these modern languages can have their data serialised into XML for transportation across networks or saving to a text based file. This means that the rule provider, for example the Department of Transport, can describe the driving rules and serialise the rules ready for distribution across the Internet. A transport companies' rule testing system can then automatically pick up a copy of the serialised rules and deserialise them back into rule objects ready for testing. This saves the software developer the effort of writing additional code to read in textual data and interpret it into a format the software understands.

One of the contributors to RuleML, in their paper (Pan, 2005), describes three important elements which a Semantic Web rule language should have (a decidable language, support for datatype predicates and support for weights). These proposed features provide a useful base foundation from which a rule markup language can be derived and are visible within RuleML as a result.

Some of the most prominent publications to be made from the RuleML initiative include (Wagner, 2002), (Boley, The Rule Markup Language: RDF-XML Data Model, XML Schema Hierarchy, and XSL Transformations, 2001), (Boley, Tabet, & Wagner, Design Rationale of RuleML: A Markup Language for Semantic Web Rules, 2001) and (Lee & Sohn, 2003).

The research papers (Chen, Wetherall, & Doncheva, Performance optimisation of rule-based testing in scheduling within a distributed processing environment (A), 2005) and (Chen, Wetherall, & Doncheva, Performance optimisation of rule-based testing in scheduling within a distributed processing environment (B), 2005), are the product of a research area spawned from the same original problem domain as this research. Rather than tackling the problem of rule description and execution of rules as this research considers, those papers look at a means of distributing rules across a corporate network in order to further increase the computational performance of the rule testing process. Their research uses an existing rule testing engine as described in (Dietrich J. , 2003), to carry out the testing of the rules although this rule testing engine does not have support for temporal rules and is designed specifically for systems developed with the Java programming language.

2.5.1 Summary

The field of rule-testing is one that is expanding with the continued drive towards the development of artificially intelligent, automated systems. The RuleML initiative has set out some important foundations in the area of rule definition with the aid of XML, however the work undertaken here does not provide the ability to describe temporal style rules leaving a substantial amount of work for a scheduling applications software developer to try and retro fit this solution into their application.

This research has identified some good practice in the way RuleML describes rules which can be taken on board although it also identifies a gap in the description of rules for use within a scheduling domain.

2.6 Dynamic Compilation / Scripting / Interfacing

The need for a performance optimised rule-testing solution that can be integrated into an existing scheduling system has been identified following the

investigation of the scheduling and rule-testing domains. Section 2.3.1 identified the need for this research to offer a solution that can be integrated into existing systems whilst Section 2.2.3 identifies the need for a performance optimised solution.

Many rule testing engines take the rules as their input and interpret them rather than compiling rules into some form of executable code. This, in some ways, is comparable to the approach of scripting programming languages against compiled programming languages. The compiled approach in programming languages results in an increase in computational performance, reducing execution times, over the scripted approach. This same philosophy can apply to rule testing.

An active research group within the RuleML initiative have developed a system called TAKE (Dietrich, Hiller, & Schenke, 2007) which demonstrates a means of rule compilation. Their results demonstrate the advantages of improved speed and rule verification by compilation.

The TAKE projects utilises the JSR199 standard (JSR-199) which provides a compiler API for scenarios such as JSP pages, where on the fly compilation is desirable (although limited to the Java platform). One of the considerable limitations of the approach taken by TAKE to rule compilation is that during its compilation process, the compiler emits Java source code in order that the source can then be compiled into Java bytecode for execution. This additional step in the compilation process potentially hinders the computational performance of the TAKE solution compared to a more direct compilation approach. There are alternative methods that are not part of the standard Java base classes, such as the BCEL (Byte Code Engineering Library) which would support the direct compilation of RuleML into Java byte code but the two approaches have not yet been combined.

Equivalent compilation features to that of BCEL exist built-in to other languages and software frameworks, such as the Reflection.Emit capabilities encapsulated within the Common Language Infrastructure (ISO/IEC23271,

2006). This feature provides the ability to avoid the intermediate compilation phase altogether by compiling rules from a rule markup language directly into the bytecode equivalent in ISO/IEC 23271, the Common Intermediate Language (CIL). Additional benefits of compiling rules from a rule markup language into CIL include the ability to use the compiled rules from a large range of programming languages and, like Java, across multiple platforms with the aid of projects such as Mono (de Icaza & Jepson, 2002).

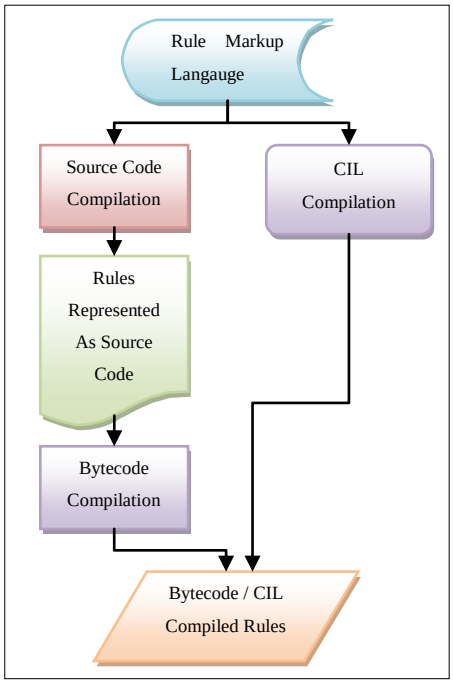


Figure 2-5 - The two approaches, required by TAKE using Java and alternatively using the CLI

(Krintz, Grove, Lieber, Sarkar, & Calder, 2001) discusses the different approaches of carrying out the compilation, in a programming language environment, not a rule testing environment, although the same concepts can apply to both. This paper looks at the different approaches of compilation and the most appropriate means of compiling Java programs from Java byte code into machine executable code.

Some important statements to support the idea that compiled code is faster than interpreted are presented in (Kerningham & Van Wyk, 1998). It states “Compiled code usually runs faster than interpreted code: the more a program

has been ‘compiled’ before it is executed, the faster it will run”. Taking this into account, a software framework such as the CLI provides two levels of compilation (firstly into CIL and secondly into machine code), potentially reducing the overall execution time.

Additionally, the thesis (Newhall, 1999) presents a method for measuring the differences between interpreted, dynamically compiled and JIT (Just-In-Time) compiled execution of software. This thesis looks at the Java language specifically; however, the theoretical concepts described apply to any language and support the view that compilation provides greater performance.

An important aspect that may need consideration is the provision of a clear application programming interface (API) to provide external systems with the ability to consume a rule engines functionality. (JSR-94), the Java Rule Engine API, provides an interface for rule engines to integrate into Java based applications in a standardised way although is not itself a rule engine. This common interface is, however, a good example of a standardised integration interface. (Whaley, Martin, & Lam, 2002) discusses mechanisms of automatic abstractions of object oriented component interfaces and (Ammons, Bodik, & Larus, 2002) looks at a mechanism for data mining for the generation of specifications.

2.6.1 Summary

This investigation into the various software execution processes has led to the belief that compilation of rules will provide the greatest performance, previously identified as a requirement in Section 2.2.3. The TAKE project provides the closest example of rule compilation for rule testing however the need for the additional step of emitting Java source code for compilation is clearly a performance hindrance whereas the use of the CLI would eliminate that step from the compilation process.

2.7 Conclusion

The literature review has outlined an important area of further research that falls across two distinct areas, rule-based testing and scheduling. A number of papers looked at under the transport scheduling area touched briefly on the rule-testing parts of the process; however, overall, there were no papers which effectively spanned both scheduling and rule-testing.

As a result of the literature review it is possible to identify a number of key contributions that this thesis aims to address.

- This research will bridge the gap between rule testing and scheduling systems providing a solution where previously an appropriate solution hasn't existed.

  The research papers in Section 2.5 which discuss RuleML have helped to identify good practice in representing rules in a human readable format. The approach taken by (Boley, The Rule Markup Language: RDF-XML Data Model, XML Schema Hierarchy, and XSL Transformations, 2001), can be extended further to represent temporal rules.

- It will aim to develop a potential rule testing definition language (see Section 3.2) that has been designed around the need for describing rules which relate to temporal problem domains such as scheduling.

  The research in Section 2.2 into GA's has helped to identify scheduling algorithms which require continuous rule testing iterations placing an increased burden on a rule testing engine where existing research in Section 2.6, such as the TAKE project, hasn't focused specifically on the efficiency of the rule execution.

- It will seek to develop a high performance approach to rule testing by compiling rules (see Section 3.3) to optimise the speed for frequent rule testing scenarios such as scheduling.

  The literature research conducted into existing scheduling systems in Sections 2.3 and 2.4 has helped to identify a need for a generic method for handling temporal rules in a solution which can be integrated into a number of systems, not just the one it was originally designed for, as other systems can clearly benefit from this type of approach.

- It will provide an example approach to modularising the rule testing engine (see Section 3.1) in a way in which would allow it to fully integrate with existing and new systems.

  Approaches looked at within Section 2.6, such as (JSR-94) and (Dietrich, Hiller, & Schenke, 2007), have helped to identify some good practice in the area of API's and modularised software integration techniques that can be considered in potential solutions to this research problem.

- It will define a method of benchmarking the resulting solution (see Section 6.1) in order that further improvements to future developments have a means of comparison.

  One of the key limitations of the existing research is the lack of an appropriate rule testing solution for interchangeable rules within a scheduling environment. As a result of this limitation no appropriate test scenarios exists which can be used as a basis for comparison of a new solution.

The bullet points outlined above have highlighted the areas of contribution this thesis will continue onward to identify a potential solution for.

2.8 Summary

This literature review has looked at a broad range of research areas in and around the domain for which this problem resides. With the aid of many references to existing publications from the various fields it has been possible to draw on the existing state of the art research in identifying both the originality of the problem domain and also potential areas of research which can support and provide the basis for identifying a solution.

The next step is to identify the specific requirements, as can be discovered during the Analysis stage of a software engineering process. Once the problem is better understood, it is possible to progress forward into identifying a means of solving the problem.

3 REQUIREMENTS

The context of the research reported in this thesis has now been identified and an extensive literature review has been carried out to clearly identify the areas in which an original contribution can be made and how this relates to existing research areas. This chapter of the thesis aims to identify, more clearly, the requirements that must be met, in order that an original contribution can be demonstrated.

A number of different approaches could have been taken to design a system which would demonstrate the contributions set out in Section 1.4 but it was considered helpful to start by looking at the various requirements first.

There are two types of requirement that need to be considered prior to the design of a potential solution (Sommerville, 2006).

1. The user requirements describe the services the system is expected to provide and the constraints under which it must operate.
2. The system requirements set out the system's functions, services and operational constraints in detail.

This chapter will mainly focus on system requirements as the research outcome is more of a component that requires integration with existing systems rather than something which will be used directly by an end user. However, Section 3.1 will take a brief look at user requirements from the perspective of a host system being the user.

3.1 Modular

The solution must be able to work with existing systems and not just those systems for which it is developed. As such it is important that the solution is a modular framework with a clear interface as opposed to a closed solution which does not require any additional integration with alternative systems.

In order to move closer to a solution it is important to look at a model which would be able to fit into various size holes in existing systems. Figure 3-1 depicts the concepts of each system that may wish to use a means of rule based testing requiring different methods of integration with the solution.

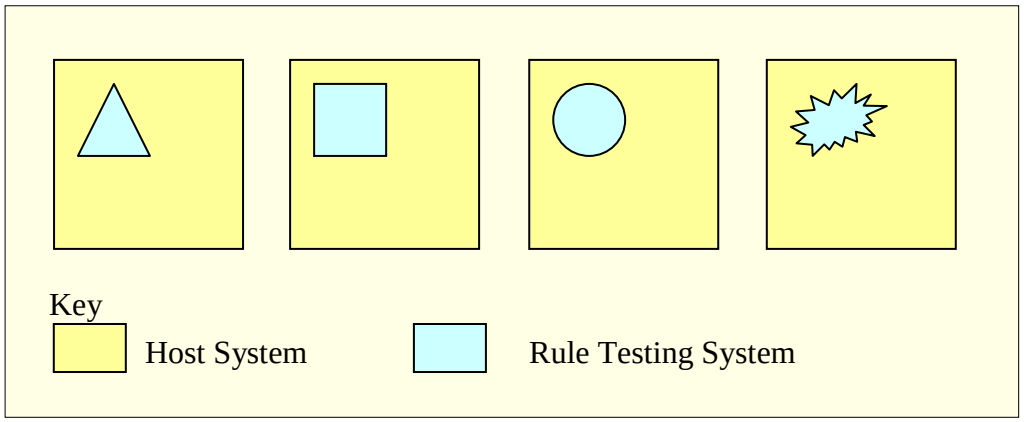


Figure 3-1 – Example of the need for a flexible interface for host system integration

The important thing about the integration of the rule testing system with the scheduling system is that it needs to provide the additional functionality without a performance decrease as a result of poor integration.

There are various integration technologies which can be used to connect systems together such as the use of standard object oriented methods for inheritance and polymorphism, although this often only works for software written in common technologies such as .NET to .NET integration or Java to Java. Alternatively the Common Object Model (COM) approach provides for cross technology integration, although this is often only useful for applications designed for Microsoft operating systems.

One of the main advantages with using the .NET technology for implementation over the Java solution is that .NET developers tend to have a wider choice of languages within which to write their software. If a Visual Basic developer wished to integrate with a rule testing system developed in C# (ISO/IEC23270, 2006), the .NET Framework provides the common object

model with the flexibility of making this possible. Also, the .NET technology has a shared source implementation known as Rotor (Stutz, Neward, & Shilling, 2003) which has been designed to work on multiple operating systems and is based on the Common Language Infrastructure standard (ISO/IEC23271, 2006).

3.2 Rule Definition

Whilst rule definition is not the specific focus of this research it is essential to consider it in order to work towards a method of rule testing which works with temporal rules. Solutions exist to represent rules, such as RuleML (Mak & Blanning, 2003) although the main problem with these existing solutions is that they do not consider the testing of temporal events.

Temporal rules need a number of custom attributes which may not be found with rules that aren't of this nature. For example, in the driver scheduling system (Wetherall, 2002), consider a weekly work period which describes the rules governing a driver's work schedule for a week. The weekly work period is dependent upon a number of daily work periods and daily rest periods which contribute to the composition of the weekly work period. The daily work period is a composition of appointments which describe the drivers work pattern. Figure 3-2 shows a graphical representation of this.

Each of the dependent work and rest periods have their own rules associated so it is easy to see that the testing of rules of this nature can quickly become very complex.

With a complex temporal problem such as this, standard rule definition languages are insufficiently rich to provide the mechanism to represent these rules. Part of the challenge would be to organise the raw data into some form of work plan in order that the data is sorted and managed in time order and then the rule testing engine would need an understanding of the breakdown of the different types of periods each having their own sets of rules.

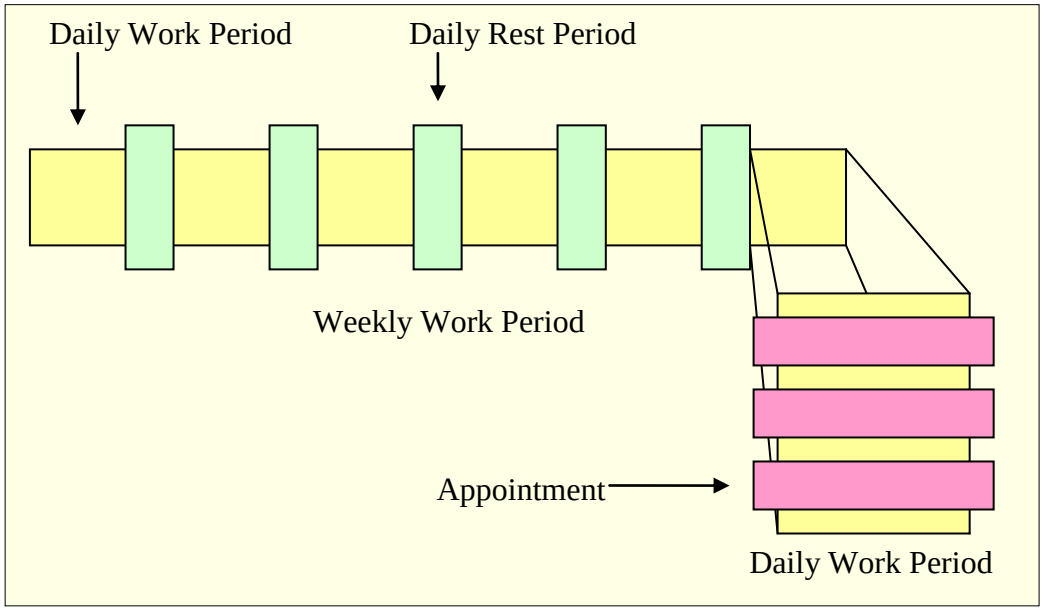


Figure 3-2 – Example of the complexity in the organisation of temporal rule data

If the rule testing engine is not built to be aware of temporal data, the client application hosting the rule testing system would have to implement the functionality to organise the data and prepare it for testing, thus making the solution less efficient. There are a number of different application areas which require the use of a temporally aware rule testing engine and without a rule testing system that is aware of the challenges posed with temporal data, each would have to implement its own individual approach.

3.2.1 XML

The use of the eXtensible Markup Language (XML) (Bray, Paoli, Sperberg-McQueen, & Maler, 2000) has become widespread and is quickly becoming the standard format for representing any type of data in the computer industry. Its primary purpose is to facilitate the sharing of data across different information systems, particularly via the Internet, which certainly meets the goals of this research, for the sharing of rules.

There are a number of advantages to using XML as the underlining rule definition technology. One of the most important reasons is that it is both human readable and computer readable at the same time. This provides the ability for the rules to be defined by both an automated rule designing tool as well as a providing the ability for the rules to be created manually using a simple text editor.

Additionally XML is a free open standard and as such makes the task of defining the structures of the rule transmission data much simpler than if a proprietary language were defined to represent this information. Additionally other consumers of the rule definition data such as alternate rule testing engines can easily understand the structure of the data as it adheres to a standard whereas if a proprietary standard, such as a binary representation, were used, specification documents of the rule definition format would need to be defined and the task of consuming the rule definition data would become far more complex.

There are disadvantages in the use of XML as a rule representation technology. The most significant disadvantage is that the syntax contains redundant information and can become large when compared to a binary representation of the same data. This redundancy would result in greater transmission costs when sending the data over a network as well as additional processing costs in reading and writing the XML.

RuleML (Wagner, 2002) uses XML effectively to represent rules. As previously stated, the main problem with the RuleML approach is that it does not support the representation of temporal rules; rules which are dependent upon data to be structured in a timed manor.

3.2.1.1 Serialisation

As far as serialisation technologies in computer languages are concerned, XML provides a standard method for converting data stored in modern object

oriented programming languages memory to something which can be saved to disk or transported across networks. It has become common place in the development of technologies such as the Service Oriented Architecture (SOA) and specifically Web Services to use XML to represent information transported from one system to another.

In the latest generation programming technologies including .NET and Java, there is built-in support for using XML and the serialisation of object data to XML. By designing the software correctly, a rule definition language for temporal rules can fall out of the bottom of the class design process.

The serialisation of data from the software object notation to XML is quite popular in modern languages. Each language tends to be supported by its own implementation of the XML standard parser as well as the functionality for serialising and deserialising between XML and objects. (Hericko et al., 2003) discusses object serialisation and carries out an analysis between the two current main software development technologies, .NET and Java.

An example of object serialisation can be found in Appendix 11.1.

3.3 Compilation Over Interpretation

Unlike many other rule testing systems, computational performance is one of the primary concerns for the requirements of this research. Existing rule testing systems are designed to allow for the one-off testing of a set of business rules whereas the rule testing process required in a scheduling application requires a high volume of rule testing iterations to occur.

Various technologies exist to permit the execution of rules within a system however existing solutions generally look towards the interpretation of rules using a rule testing engine to control the overall process. The interpretation of these rules adds an additional speed implication to testing which does not work towards an increase in computational performance.

In order to gain the maximum performance from the testing process it is necessary to look at alternative methods. Software development technologies offer a number of potential opportunities to solve this problem. If software is compiled then execution time is much faster than found with an interpreted language and, in fact, compiled software cannot run any faster as it is in machine code and is running directly on the processor itself. Looking at the problem from this angle it is possible to conceptualise that compiling business rules to executable code offers a number of potential speed increases.

Whilst compiling rules provides an opportunity for a greater improvement in rule testing performance, the process of compilation provides an additional overhead. A solution to this problem is presented when considering the use of caching the compiled rules and only recompiling if the rule set changes. Using an approach such as this provides the benefits of faster performance execution whilst only requiring recompilation when rules change, which in the majority of applications is not frequent. In the testing of driving rules (DoT, 1998), the rules do not change from one year to the next and, in examples such as this, the benefit of caching is self evident.

3.3.1 Compilation Approaches

There are a number of methods available for compiling rules into executable code. The first is to use a language compiler that matches the destination language of the rule testing engine. This would require the generation of code targeted for a specific programming language that conforms to the necessary language syntax and semantics and then to call on the compiler to compile the code into executable code. This approach would work although isn't very flexible or portable since it would require code generators for a variety of languages and the accompaniment of the compiler and its relevant dependency files along with the rule testing engine.

An alternative could be to target machine specific code and to look at creating an executable file directly. This, in itself, would result in a whole range of issues such as the need to ensure that the processor to which the executable code is being targeted is supported by the custom compiler and does not allow for much machine interoperability or future proofing.

A more desirable solution is to use the approach provided by the .NET Framework. The framework provides a standardised common language infrastructure and runtime supporting the development of software in any number of different languages. Executables compiled for the .NET Framework are not compiled into machine code but into an Intermediate Language (IL) which is Just-In-Time (JIT) compiled into processor specific machine code either in advance or on demand of its execution.

Another benefit to using the .NET Framework approach is that a set of classes are provided using a technology called Reflection which allows for the generation of an executable assembly from within a program written for the .NET Framework. This works by emitting OpCodes (IL instructions similar to that of assembly language) for the various types of operations being carried out.

An additional advantage to this approach is that common functionality which may be useful to varying temporal rule testing applications can be provided as a set of classes in a class library compiled for the .NET Framework which can be linked to by the generated executable.

For clarity, the term executable that has been used in this section refers to a file that contains executable binary code. This may include an executable file with a ".exe" extension as well as other files such as dynamic link libraries with a ".dll" extension, found on Microsoft platforms. .NET uses the term Assembly to define a file containing the executable code.

3.4 Support Functionality

There will most likely be the need to provide some additional support functionality in addition to the rule testing engine and rule representation language. For example, when looking at the driver scheduling problem (Wetherall, 2002), there are common entities which require functionality that may best be provided by the rule testing engine as opposed to the host application, such as the description of a Work Period and a Rest Period.

Also, when looking at the organisation of the raw data such as the appointments in Figure 2-1, rather than the host application having to provide the functionality to organise sets of complex data into a work plan, it may be appropriate to provide a generic method of carrying out this functionality for all host applications. By factoring such needs in from the start an optimal solution can be designed.

If this type of common functionality is not included in the rule testing engine it means that every system which requires the use of the functionality would have to provide its own means of handling these sorts of processes which would require additional development time for the developer of the host application and the potential of additional bugs occurring in the overlap between the host and the rule testing engine.

3.5 Domain Boundary

Existing scheduling systems and approaches provide the ability to schedule with the use of optimised algorithms such as GA's. The requirements therefore do not need to include the design of a scheduling system, although what is essential is that the design takes into account that the solution will need to integrate with other systems in order to provide the additional functionality that it will have to offer. This means that a clearly defined application

programming interface (API) is essential in order to provide integration with other systems.

The space within the domain boundary will need to include the design of a rule representation language such as RuleML in order to represent temporal rules. Also, and more importantly, included within the domain, there will need to be a temporally aware compiled rule testing engine designed for use with scheduling applications.

The rule representation language is required in order that a rule provider has a means of distributing rules electronically to client systems. The rules themselves have to contain the breakdown of temporal datasets as described in the original rules (for example it is the Department of Transport, 2002, which define a Daily Work Period and a Daily Rest Period, and this definition could change along with the rules).

A compiled rule testing engine is required in order to convert the rule representation language into executed code, capable of executing the rules as at greater speeds. This resulting computational performance improvement would benefit a range of application areas including scheduling applications that use, for example, the GA, where rule testing takes place frequently.

Finally, a set of support functionality should be included to provide some of the basic repetitive functionality that may be common in a number of different rule testing scheduling applications. The need for this has already been explained but, in summary, this provides a means of supporting the host application by providing more efficient integration overall.

An area which would be desirable to include within the domain boundary would be a method of graphically describing the rules prior to them being saved into the rule representation language. This would provide a much simpler means for an end user or non-IT professional to create the rules for themselves. Considering the already large scope of the domain boundary this will need to be something that is considered for further research although the

design of a simple rule editor may be needed for the benefit of developing test rules.

The diagram in Figure 3-3 provides a graphical representation of the domain boundary of the research, to aid its description.

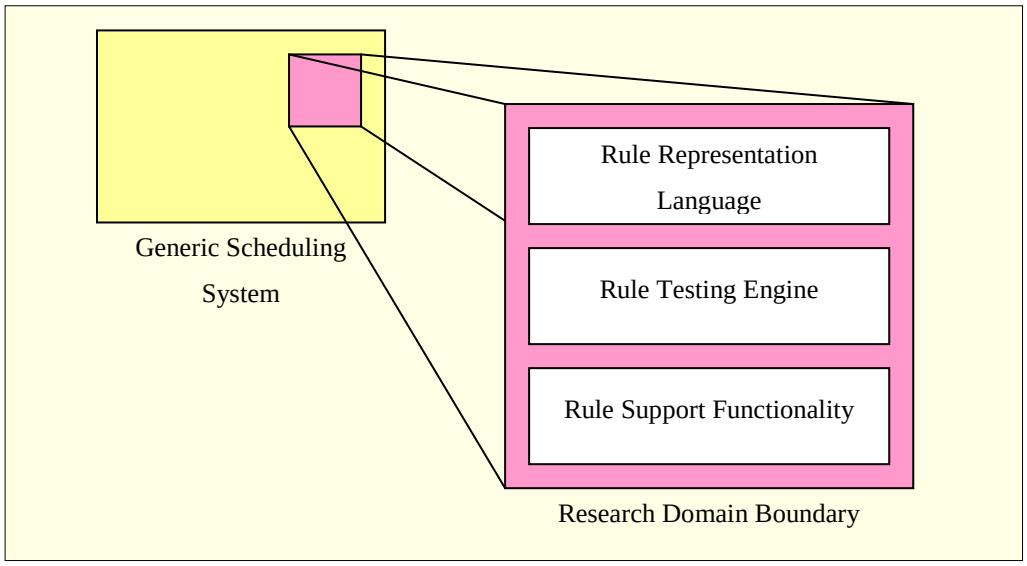


Figure 3-3 – A visual representation of the research's domain boundary

It is worthy of note at this point that the principle original contribution of the research reported in this thesis is not considered to be in the development of a rule testing engine, the compilation of rules, the representation of rules in a markup language or the additional rule support functionality. The principle original contribution is considered to be in combining all of the various areas into a single problem domain and taking into account the temporal nature of the data in scheduling systems, with a view to improving the performance of the overall system.

3.6 Design Methodology

The design methodology which has been chosen to tackle this problem is the Rational Unified Process (RUP) (Bittner K. , 2006). There are a number of

substantial reasons for taking this approach to work towards the development of a solution and these are outlined in some detail in this section of this thesis.

3.6.1 Rational Unified Process

In many traditional software engineering projects, the waterfall model, or a derivative, is used to manage the overall project process. In the waterfall process a number of disciplined stages exist, which tend to be followed serially. Figure 3-4 illustrates the stages involved in a waterfall managed software project. This clearly provides a structured approach to managing the overall software development effort.

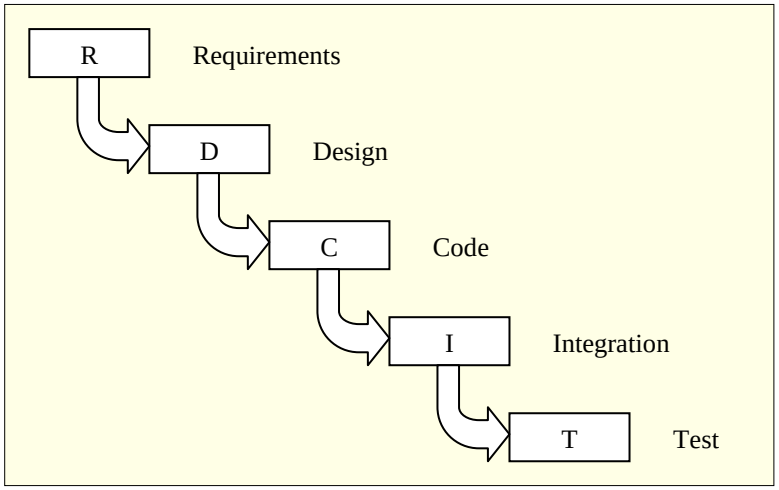


Figure 3-4 – Visual representation of the stages and flow of the waterfall process

The waterfall process works well in cases of software development where little or no innovation is required, as the level of risk is less than, for example, a research project where there is a greater amount of unknown and, as such, the level of risk is much higher. In instances such as this, the waterfall process isn't the most appropriate method to take and the RUP provides a software development model that is better suited for this type of development.

The RUP proposes an iterative development model where many of the phases used in the waterfall process occur within each iteration of the software

development lifecycle. Figure 3-5 provides a graphical representation of the RUP at a high level, illustrating that there can be many phases to the software development process.

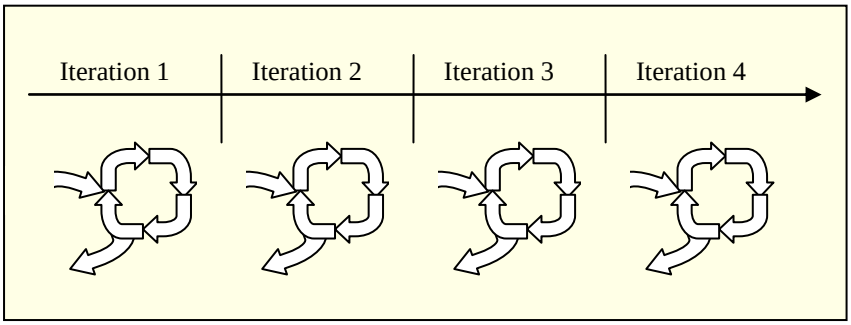


Figure 3-5 - Visual representation of the iterative nature of the Rational Unified Process

One of the main purposes of the RUP is to eliminate risks within a software project. These risks can occur at all stages of the project lifecycle and therefore the RUP works toward attacking the risks at each stage, as soon as possible, as opposed to the waterfall process which does not work towards risk elimination explicitly and therefore problems can be found at the Integration or Testing stages which would have been better dealt with much earlier in the projects lifecycle.

The RUP divides a project into four main phases:

- Inception – This stage deals with identifying the projects scope and objectives and brings the business risks under control.
- Elaboration – This stage looks at stabilizing the product plans and brings the architectural and technological risks under control.
- Construction – This stage deals with building the product whilst keeping the logistical and project execution risks under control.
- Transition – This stage looks at delivering the product keeping the roll-out risks under control.

3.6.2 Unified Modelling Language

As identified earlier in this chapter, a modern programming language with the ability to emit executable code would be appropriate as a means of implementing the desired solution. The two main contenders for this implementation would be Java and most of the .NET Framework supported languages. Both of these approaches are object oriented and, as such, an object oriented modelling language would be highly suitable to help in designing an appropriate solution.

The use of the Unified Modelling Language (UML) for the research reported in this thesis is important as this design methodology provides a mechanism for unpacking the various areas of the problem domain and a means of progressing them on toward a working solution.

The research reported in this thesis has benefited from an important research paper that helps describe UML as it was meant to be. (Rumbaugh, Jacobson, & Booch, 2004) introduces the idea of Use Cases and Class Diagrams which are essentially design methods for Object Oriented (OO) software development.

3.6.3 Design Methodology Summary

In summary, the chosen method for managing the process of this research is the Rational Unified Process (RUP) with the aid of the Unified Modelling Language (UML) as a method of documenting the designs in a standardised way.

The research reported in this thesis uses the Inception and Elaboration phases of the RUP to continue the Analysis phase of the project, whilst, at the same time, some work can continue in implementing various elements of a potential solution in order to reduce technological risks at a later stage. The main purpose being to ensure that when a solution has been designed and is ready to implement, many of the various technical risks of the implementation have

already been resolved. This should reduce the potential for progress being hindered by technical challenges and provide a means of continually moving forward.

3.7 Summary

This chapter of the thesis has gone into some detail in an attempt to better define the requirements for the problem. A number of distinct areas have been identified and their complexities unpacked in an attempt to provide a clearer picture as to the solution which will need to be designed. It has also gone some way towards identifying an appropriate design methodology to be used to help progress the research forward.

The next section of this thesis moves into the outcome of the Inception and Elaboration phases of the project where the feasibility and business risks are identified and the architectural and technological risks are brought under control. It begins to consider various elements of the system's architectural design and some of the technical challenges of its implementation.

4 INCEPTION AND ELABORATION

The preceding chapters of this thesis have attempted to provide an extensive literature review in order to clearly identify a set of contributions that the reported research can make. In addition with these chapters the requirements of the problem have been better identified and a suitable design methodology has been chosen.

The inception and elaboration stages, documented in this chapter, focus on the feasibility of the project and the need to assess the business risks and bring the architectural and technological risks under control.

4.1 Use Cases

A Use Case is a technique for capturing the functional requirements of systems and systems-of-systems. Use Cases avoid the jargon of technology and tend towards using terminology understood by the end user or domain expert (Bittner K. &., 2002).

Figure 4-1 shows a Use Case scenario for a human scheduler working in a transport scheduling environment (Wetherall, 2002), using an automated scheduling system to carry out the scheduling process. This example attempts to clearly identify the processes involved in the automated scheduling system and how the previously identified domain boundary falls outside of that process, whilst adding additional value. This example includes the need for testing the Legal Driving Rules (Transport, 1998) as part of the scheduling process of allocating drivers to appointments.

For clarity, the vehicle scheduling constraints shown within Figure 4-1 are not considered part of this research domain as the vehicle constraints relate to the appropriateness of vehicle in terms of seating capacity and amenities, and not temporal constraints which this thesis focuses on.

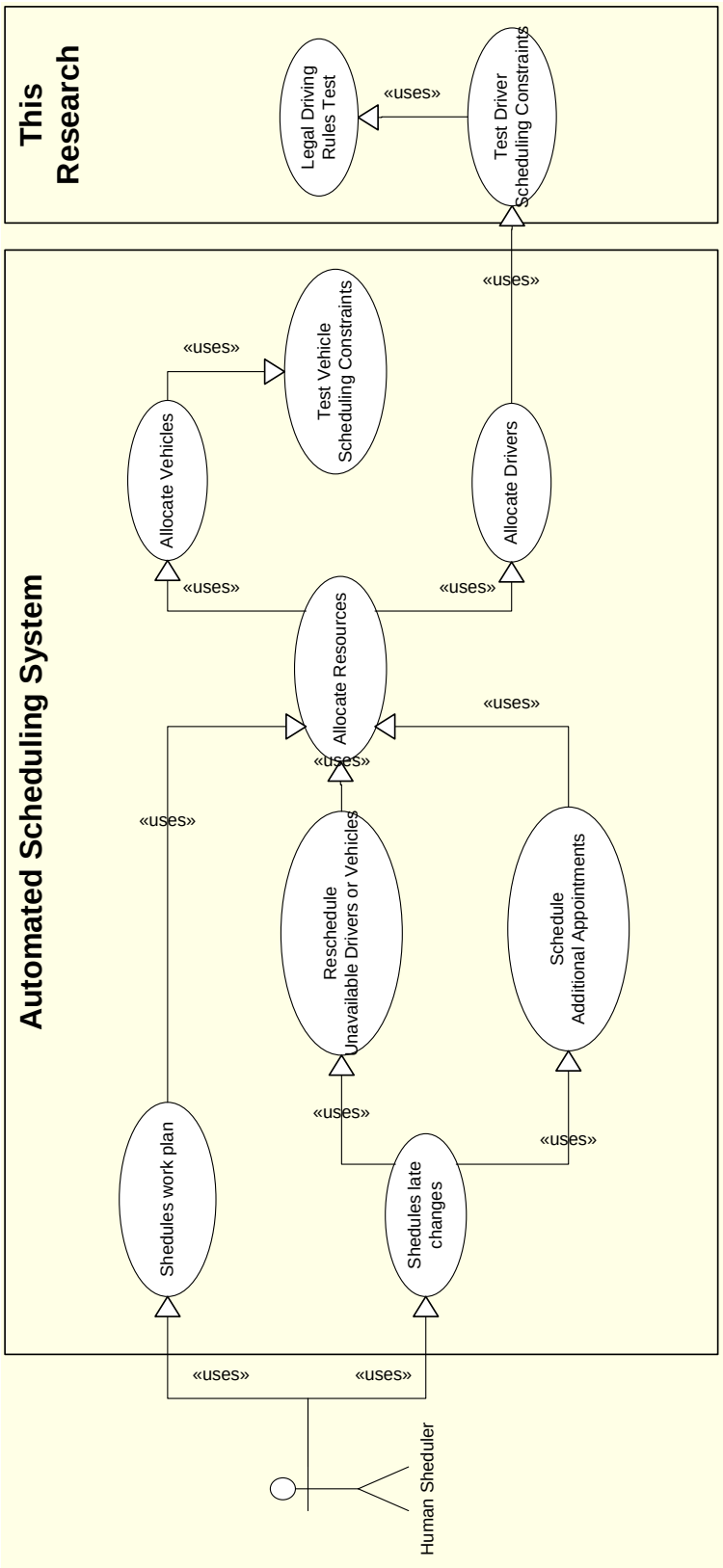


Figure 4-1 – Use Case scenario of a human scheduler using transport scheduling system

In a scenario where a GA is used for carrying out the scheduling process, the rule testing element would need to be carried out frequently in order to test a large range of possible combinations of a driver's schedule. Speeding up the rule testing element of the system will, in turn, improve the overall computational performance of the scheduling process.

Figure 4-2 illustrates a Use Case to show how an automated rule testing process would work within a scheduling system. There are a number of separate parts involved within the overall process.

To start with there is the Rule Provider who has the responsibility for providing the rules themselves. In the example used the Rule Provider is the Department of Transport. At this stage it is envisaged that the rules would not only be provided as a human readable document (Transport, 1998) but also provided in a description language which can be used within a rule testing system, in order to test the rules automatically.

The scheduling system would need some form of awareness of the need for rules in order for the user to be able to specify a location for the system to automatically access them, so they can be downloaded and passed on to the rule testing engine for processing. Clearly this is not something that can be hard-coded into the rule testing engine as the engine needs to be able to test generic rules from many different Rule Providers and, as such, wouldn't be aware of the application that it is integrated into, only that it processes the rules provided to it.

The systems integration section is vital to the overall process as it would ensure that the engine can be fully integrated with existing or future scheduling systems and therefore must be fairly dynamic to cater for a large range of integration requirements. This section would need to be clearly defined and documented for systems designers and integrators.

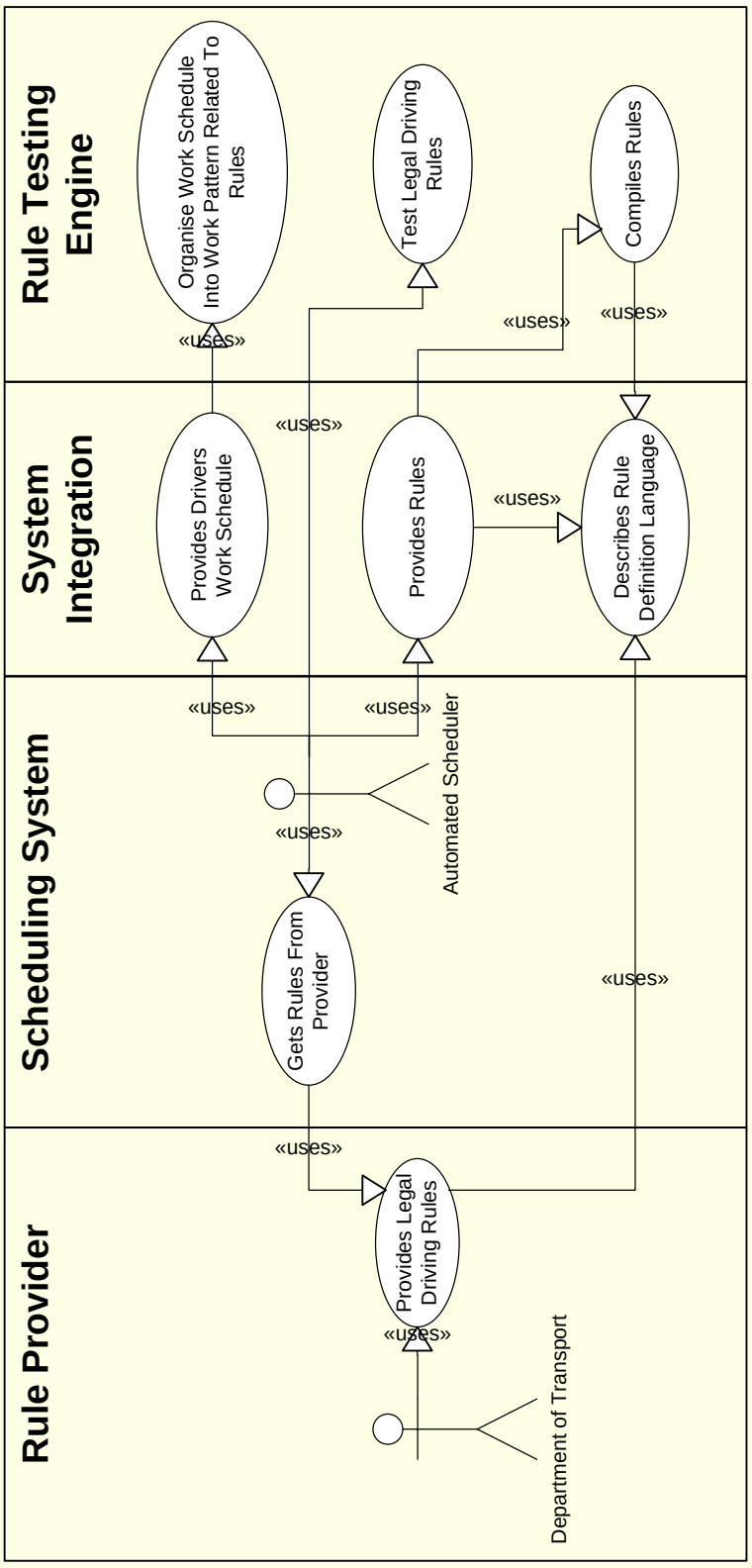


Figure 4-2 – Use Case identifying the components of an automated scheduling system requiring rule testing

The integration section also contains the description of the rule definition language. This language may be the key to the overall effectiveness of the description of the rules and then compilation of those rules into something which is executable and, in turn, can run quickly. There is the need to be able to describe the data to be tested against those rules in a way in which the integrating scheduling system and the rule testing engine can both understand. For example in a scheduling system temporal information, such as the start and end of appointments, would be fairly generic and something which the rule testing engine should be able to comprehend, so it is probably best for this data description to exist in the integration section.

The rule testing engine has some complex yet simple to describe use cases interacting with the integration section. The rule testing engine needs to be able to compile the rules because, as discussed in the Section 3.3, compiled rules will execute much faster than interpreted rules.

The rule testing engine will need to be able to organise the data provided by the scheduling system into some form of work pattern. The legal driving rules (Transport, 1998) describe rule dependencies and the organisation of data into daily, weekly and fortnightly work and rest rules; the rule testing engine would need to be able to organise the data in this way in order to be able to test the data against those rules.

4.2 Design

Following on from some of the literature review,(Lee & Sohn, 2003)(Boley, Tabet, & Wagner, Design Rationale of RuleML: A Markup Language for Semantic Web Rules, 2001) and the analysis of the problem, it is found that there is a significant difference between the rule testing process in this research and that of other rule testing systems. This is largely due to the need to deal with temporal rules for scheduling applications and, as such, there needs to be a certain level of design focus on this during the Inception and Elaboration phases.

4.2.1 Rule Support

In order to have a generic rule testing system it is important to provide a mechanism to permit its integration with a host application. This application could be a scheduling system but may also be other types of system requiring the ability to test temporal rules, so the integration mechanism needs to be flexible.

Figure 4-3 shows the Class Diagram for part of the integration support functionality looking at the `IPeriodElement` interface and how, for example, in a transport scheduling application, an Appointment may inherit it as an `Appointment` has a `StartTime` and `EndTime` and needs to describe the `RestLength` and `WorkLength` to the rule testing engine.

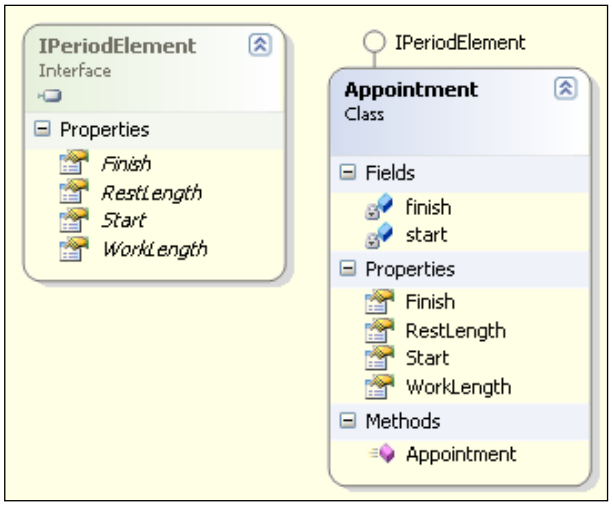


Figure 4-3 – Class diagram illustrating an example Appointment class which inherits from a generic IPeriodElement interface

The `IPeriodElement` interface can be used by any sort of temporal event to describe start and end times. If the event is a rest period then the `RestLength` would be the difference between the `Start` and `Finish`, whereas if the event is a work period then the `WorkLength` would be the difference.

In the given example the `Appointment` is used as the data to be tested against a set of rules (see Section 6.1.1.2). If the scheduling system were to test, for example, timetabling for schools, the data may not been an `Appointment` but may be a taught subject and therefore it would still need to describe a start and end time and also the `WorkLength` and `RestLength`.

4.2.2 Rule Definition

Once integrated with a host application such as a scheduling system it is necessary to describe a set of rules for the data to be tested against. Figure 4-4 shows the Class Diagram for the mechanism of rule description and compilation of rules at the highest level. These classes are used to begin the description of rules and, as can be seen in the base class `Period`, contain an attribute called `Rules` which is an array that contains those rules associated with either a Work Period (`WP`) or a Rest Period (`RP`).

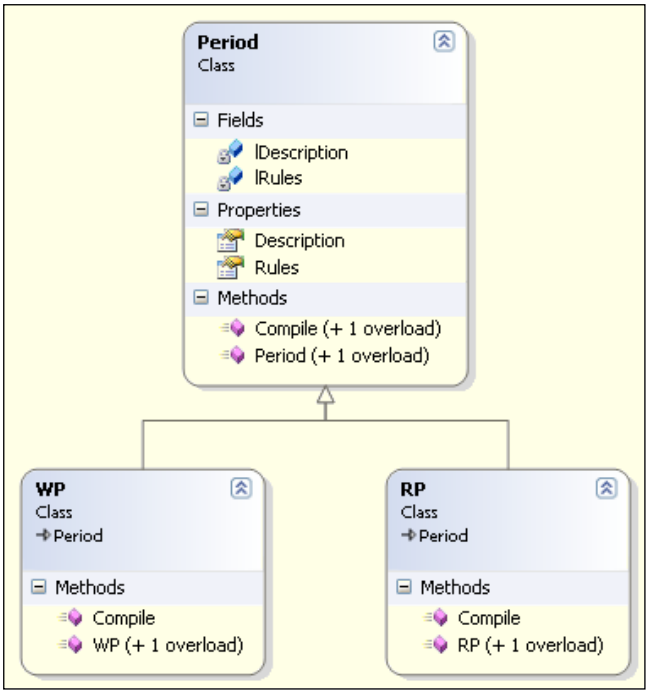


Figure 4-4 – Class Diagram describing the concept of a Period and its subclasses, WP (Work Period) and RP (Rest Period)

Another important thing to note about the `Period` class is the method called `Compile` which is designed to be called when the rules are to be compiled. The process of compilation is fairly simple; at the highest level the rule is told to compile, which in turn, calls the `Compile` method of the various rules stored within the `Rules` attribute. Those sub-rules call the `Compile` method of the rules stored within them and so on, in a recursive fashion, until each rule or sub-rule of a rule is compiled and has had the opportunity to emit its functionality to the resulting executable library.

It is important to acknowledge again, at this point, that the rule testing process does not apply the if-then approach as there are no actions. The approach required for the legal driving rules example is to determine if the driving schedule breaks the rules; if it does then we cannot use the given schedule, otherwise the schedule is valid. This means that during the testing phase the rule has to result in a Boolean value, `true` or `false`, depending on whether or not the rule is broken. If the result of the test is `true`, the rules were tested and the result was that the rules were not broken, if the value returned was `false`, the rules were broken.

Figure 4-5 identifies the different types of rules that exist that may result in a Boolean value. These all inherit from the abstract class `Comparitor` which describes the ability to make a generic comparison. Each class also has a `Compile` method to provide the opportunity for it to emit the relevant executable code to perform its designated operation.

The `AndCondition` class provides a mechanism for having two or more rules that have to be met in order that the overall rule can be met. A transport scheduling example is if you can drive up to 10 hours in a day and up to 6 days a week, these may be two separate rules but they can be concatenated by an And operation.

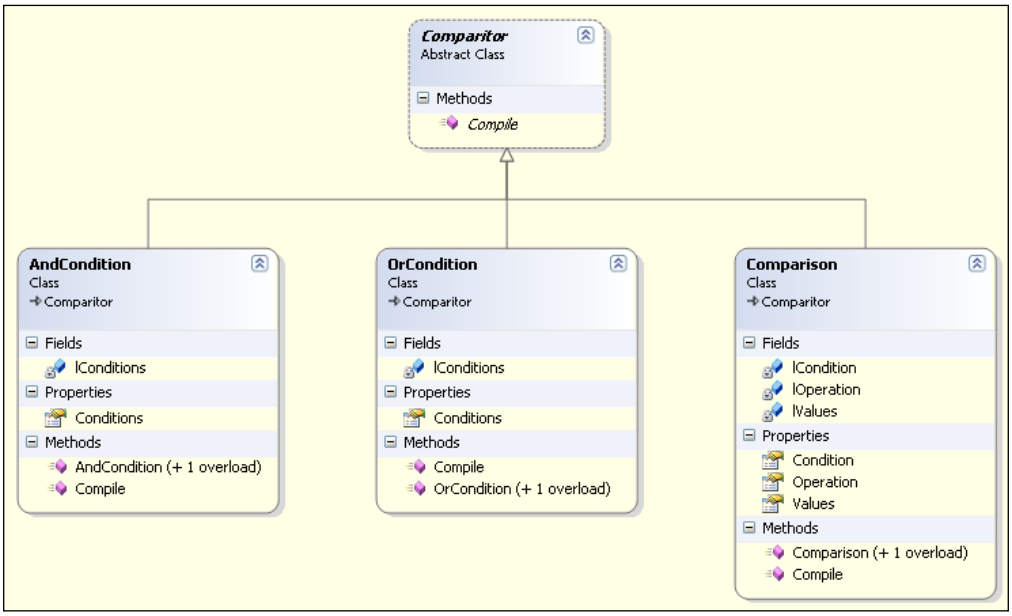


Figure 4-5 – Class Diagram illustrating the various types of rule

The `OrCondition` class carries out an `Or` operation on the various rules contained within it and the `Comparison` class is used to provide slightly more advanced non-boolean comparisons such as if a number value is greater or less than, similar to those which are available to 'if' statements in common programming languages.

The sorts of values that can be used in making a comparison range from numeric values to dates and times and even string values. Most importantly parameter values derived from the data itself, such as the `StartTime` of an `Appointment`.

In order to describe a generic value that has some form of data the `Value` class is used. Figure 4-6 shows the `Value` class which has a `Compile` method to allow it to compile as part of the overall rule compilation process. Figure 4-6 also identifies the `Math` class which is derived from the `Value` class. The `Math` class provides the ability to perform mathematical operations on two or more `Value` objects by specifying the `Value` objects and the type of `Operation` to be performed such as `Addition`, `Subtraction`, `Multiplication` and `Division`.

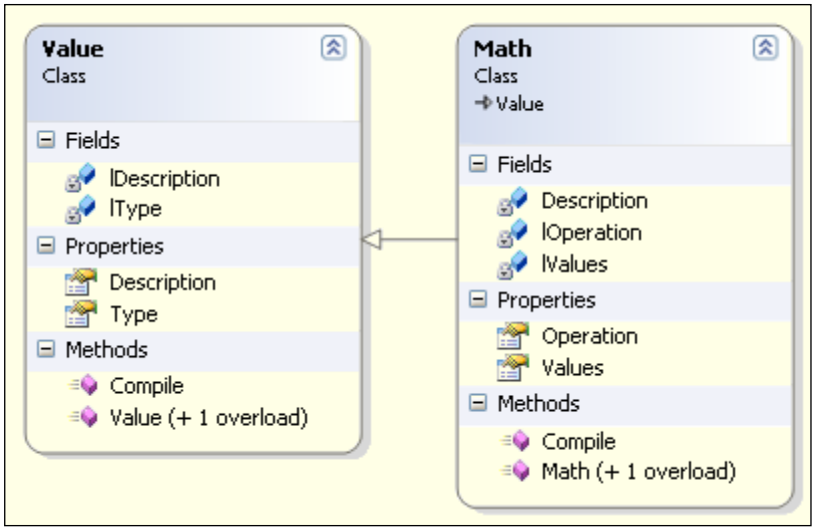


Figure 4-6 - Value and Math Operations

All of the different classes which go toward describing a set of rules must have the ability to be serialised into XML. This may require that certain attributes are placed on the properties of a class, or even the class itself in order to correctly control the serialisation process.

4.2.3 Work Plan Organisation

To test against temporal rules such as those in driver scheduling the rule testing software needs the data to be organised into a work plan. A work plan would have the data arranged into various types of period, for example a Daily Work Period, Weekly Work Period, Daily Rest Period or Weekly Rest Period.

This type of work plan requires that data be ordered in a particular way, for example, there may be two appointments that, when combined, contribute toward a Daily Work Period and the remaining appointments fall under a different Daily Work Period and the two Daily Work Periods are separated by a Daily Rest Period. These two Daily Work Periods together with the Daily Rest Period may form a Weekly Work Period.

A method for handling the organisational description of rules is in the `PeriodDescription` class that can be seen in Figure 4-4 as a member of the `Period` class, as its `Description` property.

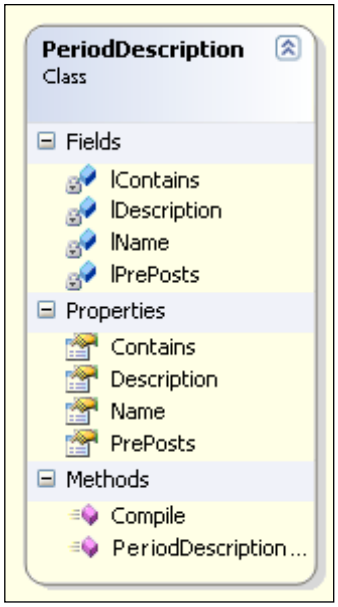


Figure 4-7 - Period Description Class

Figure 4-7 shows the `PeriodDescription` class in detail. The first things to note are the `Name` and `Description` attributes which provide a means of describing this type of rule, the name being a short form of the rule and the description providing a more detailed description. An example could be that the name is "`DWP`" and the description is "`Daily Work Period`". The name would be used to help reference the type of `Period` such as when using the `Parameter` type in the `Value` class; to reference the start of the Daily Work Period, the shortened form of '`DWP.StartTime`' would be used.

The `Contains` attribute is used to describe what types of periods can be contained within the `Period`. For example the Daily Work Period would contain Appointments and the Weekly Work Period would contain Daily Work Periods and Daily Rest Periods.

The `PrePosts` attribute provides a mechanism for describing the sorts of periods that come before and after the given period. For example an Appointment can come before or after another Appointment. A Daily Work Period can come either before or after either a Daily Rest Period.

With this level of detail being assigned to a period of time it is possible to organise the period appropriately so that a work plan is constructed. By providing the facility to construct the data into a work plan it takes away the burden from the host program and allows for an optimised method of managing the process of the work plan.

The construction of the work plan is not the main focus of the research reported in this thesis and, as such, a fairly simple approach to constructing a work plan can be used with the ability in future research to provide a less processor intensive, more optimised construction method.

4.3 Implementation

When designing the classes of the various elements of the rule testing engine, it is possible that the definition of the rule description language can occur as a direct result of the class design, due to the ability for objects in modern object oriented languages to be serialised into XML.

Figure 4-8 gives an example of an instance of the `WorkPeriod` class serialised into XML. The data associated with the class is also serialised providing a hierarchical description of the rule in a human readable language. At a later point this XML can be deserialised back into object data in a program where the rule engine can compile these rules into executable code and then test data against them by organising the data into a valid work plan.

The example illustrated in Figure 4-8 is a simple one. A comparison is made to the work period in order to ensure that the overall `WorkLength` of the work

period is less than or equal to 8 hours. This is one of the legal driving rules and demonstrates that this approach can be used in real world applications.

```
<?xml version="1.0"?>
<Period xsi:type="WP">
  <Description>
    <Name>DWP</Name>
    <Description>Daily Work Period</Description>
    <PrePosts>
      <PrePost>
        <Pre />
        <Post>DRP</Post>
      </PrePost>
      <PrePost>
        <Pre>DRP</Pre>
        <Post>DRP</Post>
      </PrePost>
      <PrePost>
        <Pre>DRP</Pre>
        <Post>WRP</Post>
      </PrePost>
      <PrePost>
        <Pre>WRP</Pre>
        <Post>DRP</Post>
      </PrePost>
      <PrePost>
        <Pre>DRP</Pre>
        <Post />
      </PrePost>
    </PrePosts>
    <Contains>
      <Contain>
        <Name>Appointment</Name>
      </Contain>
    </Contains>
  </Description>
  <Rules xsi:type="Comparison">
    <Operation>LessThanOrEqualTo</Operation>
    <Values>
      <Value>
        <Description>WorkLength</Description>
        <Type>Parameter</Type>
      </Value>
      <Value>
        <Description>8</Description>
        <Type>Hours</Type>
      </Value>
    </Values>
  </Rules>
</Period>
```

Figure 4-8 – An example of a Work Period object serialised into XML

An important thing to note in Figure 4-8 is that there is a lot of XML used to provide the description of the rule which is a serialised form of the class described in Figure 4-7. It is this description that allows the rule testing engine to organise the data into a valid work plan.

4.4 Feasibility

As identified in the Literature Review chapter, there are a number of research fields that work around the area of rule-based testing for scheduling systems, but there isn't a specific solution which can meet the requirements outlined in Chapter 3, or that are consistent with the contributions outlined in Section 1.4.

The analysis work carried out identifies the Use Cases involved in an automated scheduling system that requires rule testing and provides the first step towards a solution to prove the identified contributions. The designs provide a mechanism by which the concepts outlined in Section 1.4 can be implemented and highlight an approach that can, once implemented, demonstrate the increase in speed and efficiency of the overall process in comparison to a manual scheduling process, such as Figure 4-1.

4.5 Summary

This section of the thesis has managed to identify many areas of the problem and requirements that need to have their architectural and technological risks brought under control and has attempted, with the aid of UML, to unpack and expand on the various aspects of the problem. Diagrams have been produced that can help to move forward to the next phase of the development process, to construct a solution.

The next phase, the Construction Phase, is the point where the main implementation will take place, with the support of the work completed so far.

5 CONSTRUCTION

In the preceding chapters this thesis has presented an extensive literature review and has then attempted to identify the original contributions made by the reported research. It has also attempted to break down the research requirements in order to more clearly define the problem; some analysis and design has also been presented with the aid of the RUP design methodology and UML.

During the Construction phase the aim is to bring the logistical risks of the project under control. This phase is the largest phase within the project development lifecycle and, by the end of the Construction phase, a working solution should be available.

Based upon the results from the Inception and Elaboration phases it is possible to move forward further into the implementation of a number of different elements of the design, whilst at the same time continuing some of the design work that has yet to take place, often due to the need to have implemented some parts of the system first.

This chapter presents the outcome of a large amount of design and implementation work by discussing various components of the proposed solution. It looks into the process of rule compilation and looks in detail at the technology employed and the specific implementation of the various software classes designed to support this process.

In addition to the various elements described within this Chapter, Appendix 11.2 provides a set of complete class diagrams that describe the overall solution and Appendix 11.3 provides a complete set of code from which a number of code snippets are taken and explained in detail during this Chapter.

5.1 Rule Definition

As a result of the Inception and Elaboration phases, designs were produced to help describe a suitable method of implementing the rule definition functionality of the system. As previously stated, providing this is implemented correctly, the XML version of the rules should result directly from the class implementation.

In order to help describe the various classes and their functionality we will start by looking at the system from the perspective of the compiler and then gradually move through the process of compilation to see how the other classes carry out their compilation functionality.

The reader should note that from this point forward the term ‘period’ is used to describe a temporal rule containing a rule definition and its additional metadata describing how the given temporal rule relates to other temporal rules. This has been described in some detail in the previous chapter. The term ‘data’ refers to the units of information to be tested against the rules, such as the Appointment, previously seen in the transport scheduling scenario.

It is also worth noting at this point that a ‘period’ is designed to describe temporally organised sets of data and the rules within a given period support the organisation of data into a form of work plan. For the testing of typical business rules that do not require work plan organisation, a standard rule engine, such as Mandarax (Dietrich J. , 2003) or TAKE (Dietrich, Hiller, & Schenke, 2007), may be more appropriate.

5.1.1 Compiler

The `Compiler` class comes under the `RuleCompiler` namespace and is designed with a single goal in mind, to provide a mechanism for the host system to take the rules and turn them into an executable assembly. The

`Compiler` class is not designed to have an instance created; it is designed with a single static method called `Compile` that, when called, turns the rules into an executable assembly.

The first thing that the `Compile` method does is to determine how the types of data to be tested against the rules relate to the rules themselves and how the various periods and data relate to each other. It is important to determine this first as it can eliminate a number of compilation errors that could potentially occur later in the process.

An essential part of the compilation process is to define the new assembly where the compiled code will be placed. This is currently fixed for the purpose of simplicity in demonstrating the concepts of the system, although this can be easily changed later to be more dynamic.

Once the new assembly is defined, as demonstrated in Figure 5-1, the compiler iterates through all of the periods in the period array, instructing them to compile one at a time, each injecting its IL into the assembly. Each period creates its own class within the assembly using the short name given in its description as the class name. The classes produced can be created and executed at will and are later returned as a result of the call to the `Compile` method of the Compiler class.

```
// Declare the necessary assembly builder variables for use
// in this method.
AssemblyName assemblyName = null;
AssemblyBuilder builder = null;
ModuleBuilder module = null;

// Intialise the new Assembly and declare it is run only for
// the time beging. Later in may be worth look at ways of
// caching the compiled assembly to prevent recompilation but
// for the time being, we will recompile.
assemblyName = new AssemblyName();
assemblyName.Name = "TestAssembly";
builder = AppDomain.CurrentDomain.DefineDynamicAssembly(assemblyName,
AssemblyBuilderAccess.RunAndSave);

// Define the module within the Assembly that will contain
// the new types.
module = builder.DefineDynamicModule("TestModule", "Test.dll");

// Define an array to store the new types, one type per
// period, then, using a loop, get each period to compile
// itself.
Type[] theTypes = new Type[periods.Count];
```

```
for(int i = 0; i < periods.Count; i++)
        theTypes[i] = periods[i].Compile(module,
typeMapping[periods[i].Description.Name]);

// Save the compiled rules into a dynamic link library
builder.Save("Test.dll");

// Save an XML representation of the rules along with
// the DLL to allow for debugging later.
return theTypes;
```

Figure 5-1 - Code snippet illustrating the process of constructing a new rule assembly

5.1.2 Period

The `Period` class comes under the `RuleDefinitionLanguage` namespace, along with the remaining classes under section 5.1, and was designed to describe a temporal element which contains rules. An example of this from (Transport, 1998) would be a Daily Work Period or Weekly Rest Period. The class design provided in Figure 4-4 shows the concept of the `Period` class with a relatively simple implementation. It is not until the implementation is tackled that the complexity shows.

The first step in compiling the `Period` class, as shown in Figure 5-2, is to define a new public class using the periods' short name. From this a new default constructor is created whose task it is to call the constructor of its base class. This is because the class being defined has an equivalent runtime base class which is used to provide additional functionality, saving the compilation process of the rules from having to include some of the standard rule testing functionality.

```
// Define the variables used within this method.
TypeBuilder newType = null;
ILGenerator ilGenerator = null;
ConstructorBuilder constructor = null;
MethodBuilder testMethod = null;

// Define the new class for this particular period type basing
// it on the base class provided.
newType = moduleBuilder.DefineType(Description.Name, TypeAttributes.Class |
TypeAttributes.Public, baseType);

// Define the constructor for the new object and emit the code
// to get the constructor to call the base classes constructor
// which should initialise everything properly. Also get the
// new types Description object to inject its intial setup
// values inserted into the constructor.
constructor = newType.DefineConstructor(MethodAttributes.Public,
```

```
CallingConventions.Standard, new Type[0]);
ilGenerator = constructor.GetILGenerator();
ilGenerator.Emit(OpCodes.Ldarg_0);
ilGenerator.Emit(OpCodes.Call, baseType.GetConstructor(new Type[] {}));

Description.Compile(baseType, typeMap, ilGenerator);
ilGenerator.Emit(OpCodes.Ret);
```

Figure 5-2 – Code snippet illustrating the creation process of a new class

The new class also needs to store the description information for its rule which is essential for helping the class to later determine how to generate the work plan, such as which periods can be stored within or around it. This is achieved by adding custom class attributes to each period class generated. The benefit of using attributes is that a new instance of the period class does not need to be instantiated to access this information at runtime resulting in improved performance.

Figure 5-3 provides a code example showing how the custom attributes for the period description are emitted into the compiled executable to support the work plan generation process.

```
// The customAttribute local variable is used to store the
// CustomAttributes that will be defined for this rule type.
CustomAttributeBuilder customAttribute;

// The values local variable is used to store an array of
// string to be held by the customAttribute once compiled.
List<string> values;

// First of all construct array of the items this type can
// contain, then construct a ContainsAttribute for this type.
values = new List<string>();
foreach (string contain in typeMap.Contains)
      values.Add(contain);
customAttribute = new
CustomAttributeBuilder(typeof(RuleSupport.ContainsAttribute).GetConstructor(new
Type[] { typeof(string[]) }), new object[] { values.ToArray() });
newType.SetCustomAttribute(customAttribute);

// Now construct an array of the types that can contain this
// type, then construct a ContainedWithinAttribute for this
// type.
values = new List<string>();
foreach (string containWithin in typeMap.ContainWithin)
      values.Add(containWithin);
customAttribute = new
CustomAttributeBuilder(typeof(RuleSupport.ContainedWithinAttribute).GetConstruc
tor(new Type[] { typeof(string[]) }), new object[] { values.ToArray() });
newType.SetCustomAttribute(customAttribute);

// Now construct a list of the items than can come before
// and after this type, then construct a PrePostAttribute
// for this type.
values = new List<string>();
foreach (PrePost prePost in typeMap.PrePost)
{
      values.Add(prePost.Pre);
```

```
        values.Add(prePost.Post);
}
customAttribute = new
CustomAttributeBuilder(typeof(RuleSupport.PrePostAttribute).GetConstructor(new
Type[] { typeof(string[]) }), new object[] { values.ToArray() });
newType.SetCustomAttribute(customAttribute);
```

Figure 5-3 – Code snippet demonstrating how the period description is included with the compiled rule, as attributes to the rules class

The next task is to define the `Test` method. The `Test` method is used to start the testing process of the given period, returning a Boolean value indicating the outcome of the test. The contents of the test method are, however, not yet defined, apart from the point of returning a value. The definition of the Test method comes from the `Rules` collection contained within the period.

It is possible to see, from the two code snippets, Figure 5-2 and Figure 5-4, how simple it is to start emitting IL into the assembly. Once the class has been created it is a simple case of a method call to `DefineMethod` to define a new method within the class. From the returned `MethodBuilder`, it is a simple case of calling its `GetILGenerator` method to return an `ILGenerator` object which is used to emit IL into the method.

```
// Now define the Test method which returns a boolean value,
// true meaning that the test was successful, false it is was
// not.
testMethod = newType.DefineMethod("Test", MethodAttributes.Public |
MethodAttributes.Virtual, CallingConventions.Standard, typeof(bool), new
Type[0]);

// In order to generate the necessary IL code for this method,
// we get the rules themselves to inject their sections of the
// code into. Each section of the rule will normally inject
// its code into seperate scopes to keep the different sections
// seperated.
ilGenerator = testMethod.GetILGenerator();
LocalBuilder retVal = ilGenerator.DeclareLocal(typeof(bool));

ilGenerator.BeginScope();
Rules.Compile(retVal, ilGenerator);
ilGenerator.EndScope();
ilGenerator.Emit(OpCodes.Ldloc_S, retVal);
ilGenerator.Emit(OpCodes.Ret);

// Return the newly created type to the calling object.
try
{
        return newType.CreateType();
}
catch(Exception e)
{
        throw new RuleCompiler.RuleCompilerException("CreateType Failed: " +
e.ToString());
}
```

Figure 5-4 – Code snippet illustrating the creation of the Boolean returning Test method

5.1.3 Comparitor

The `Comparitor` class is designed as the base class for all types of rule as it provides a means of carrying out a test and returning a Boolean value.

The `Compile` method of the `Comparitor` class can be seen in Figure 5-5. It shows that the return value generated as a result of the condition is stored in the `retVal` parameter, the first argument of the Compile method, and the functionality of the comparison is emitted into the executable through the `ilGenerator`, the second argument of the Compile method.

There are three main classes which inherit from the `Comparitor` class, as previously seen in Figure 4-5, providing a means of building up a set of more complex rules. These are the `AndCondition`, `OrCondition` and `Comparison` classes.

```
/// <summary>
/// Comparitor Class. Used to represent an operation which will
/// result in either a true or false answer.
/// </summary>
public abstract class Comparitor
{
    /// <summary>
    /// This method, when consumed by a parent class, is used for
    /// the parent to emit its IL.
    /// </summary>
    /// <param name="retVal">
    /// The retVal is the boolean value where the operation will
    /// store the result of the comparison.
    /// </param>
    /// <param name="ilGenerator">
    /// The ilGenerator is to be used for emitting the IL to.
    /// </param>
    public abstract void Compile(LocalBuilder retVal, ILGenerator ilGenerator);
}
```

Figure 5-5 – Code snippet illustrating the abstract base class, Comparitor, and its Compile method

5.1.4 AndCondition

The `AndCondition` rule is designed to concatenate a number of sub-rules together to form a more complex rule. All of the sub-rules Anded together must result in a positive outcome in order for the `AndCondition` to return `true`.

Figure 5-6 shows the `Compile` method implementation for the `AndCondition`. Firstly, an array of local variables are defined to store the result of the sub-rules of the `AndCondition`. Each sub-rule then has its `Compile` method called causing it to emit its IL, storing its Boolean result into one of the predefined local variables.

Once all of the sub-rules have been emitted, each return value is compared, one-by-one with the previous, to determine if the results indicate a failed rule. If a failed rule is detected then execution branches out of the loop to where the `endLabel` has been marked, at the end of the method, otherwise execution branches to the `nextLabel` and the next Boolean result of the next sub-rule, until all of the results have been compared.

```
LocalBuilder[] retVals = new LocalBuilder[Conditions.Length];

for(int i = 0; i < Conditions.Length; i++)
{
        retVals[i] = ilGenerator.DeclareLocal(typeof(bool));
        ilGenerator.BeginScope();
        Conditions[i].Compile(retVals[i], ilGenerator);
        ilGenerator.EndScope();
}

Label endLabel = ilGenerator.DefineLabel();
ilGenerator.Emit(OpCodes.Ldc_I4_0);
ilGenerator.Emit(OpCodes.Stloc_S, retVal);
ilGenerator.Emit(OpCodes.Ldloc, retVals[0]);

for(int ii = 1; ii < retVals.Length; ii++)
{
        ilGenerator.Emit(OpCodes.Ldloc, retVals[ii]);
        Label nextLabel = ilGenerator.DefineLabel();
        ilGenerator.Emit(OpCodes.Beq_S, nextLabel);
        ilGenerator.Emit(OpCodes.Br S, endLabel);
        ilGenerator.MarkLabel(nextLabel);
}

ilGenerator.Emit(OpCodes.Ldc_I4_1);
ilGenerator.Emit(OpCodes.Stloc S, retVal);
ilGenerator.MarkLabel(endLabel);
```

Figure 5-6 - Code snippet illustrating the compilation process of the And class

5.1.5 OrCondition

The `OrCondition` works in a very similar way to the `AndCondition`. The main difference is that it is not checking for a Boolean And comparison, but for a Boolean Or. Figure 5-7 provides the code for the `Compile` method of the `OrCondition`.

5.1.6 ComparisonOperation

The `ComparisonOperation` enum provides a programmatic list of operation types that can be carried out within a comparison between two values to produce a Boolean result.

```
LocalBuilder[] retVals = new LocalBuilder[Conditions.Length];

for(int i = 0; i < Conditions.Length; i++)
{
        retVals[i] = ilGenerator.DeclareLocal(typeof(bool));
        ilGenerator.BeginScope();
        Conditions[i].Compile(retVals[i], ilGenerator);
        ilGenerator.EndScope();
}

Label endLabel = ilGenerator.DefineLabel();
ilGenerator.Emit(OpCodes.Ldc_I4_0);
ilGenerator.Emit(OpCodes.Stloc_S, retVal);
Label trueLabel = ilGenerator.DefineLabel();

for(int ii = 0; ii < retVals.Length; ii++)
{
        ilGenerator.Emit(OpCodes.Ldc_I4_1);
        ilGenerator.Emit(OpCodes.Ldloc, retVals[ii]);
        ilGenerator.Emit(OpCodes.Beq_S, trueLabel);
}
ilGenerator.Emit(OpCodes.Br_S, endLabel);

ilGenerator.MarkLabel(trueLabel);
ilGenerator.Emit(OpCodes.Ldc_I4_1);
ilGenerator.Emit(OpCodes.Stloc_S, retVal);

ilGenerator.MarkLabel(endLabel);
```

Figure 5-7 - Code snippet illustrating the compilation process of the Or class

The `ComparisonOperation` enum is designed to work with the `Comparison` class to describe the type of comparison to make against multiple values.

```
/// <summary>
/// The ComparisonOperation enum describes the type of
/// comparison to make on two true or false values.
/// </summary>
public enum ComparisonOperation
{
    /// <summary>
    /// Are the two values equal?
    /// </summary>
    Equals,

    /// <summary>
    /// Are the two values not equal?
    /// </summary>
    NotEquals,

    /// <summary>
    /// Is the first value less than the second? Can only be
    ///  applied to values of a numeric nature.
    /// </summary>
    LessThan,

    /// <summary>
    /// Is the first value less than or equal to the second?
    /// Can only be applied to values of a numeric nature.
    /// </summary>
    LessThanOrEqualTo,

    /// <summary>
    /// Is the first value greater than the second? Can only
    /// be applied to values of a numeric nature.
    /// </summary>
    GreaterThan,

    /// <summary>
    /// Is the first value greater or equal to the second?
    /// Can only be applied to values of a numeric nature.
    /// </summary>
    GreaterThanOrEqualTo
};
```

Figure 5-8 - Code snippet illustrating the definition of the comparison operations

5.1.7 Comparison

The `Comparison` class describes the means of making a comparison between two or more values. The result of a comparison, as with all other basic rule types, is a Boolean value indicating the outcome of the comparison. The `Compile` method of the `Comparison` class is designed to emit the IL to deal with the various types of comparison as seen in Figure 5-8.

The code snippet provided in Figure 5-9 shows that the first thing to be done is to emit the IL for all of the values and to store their results in an array of variables. The next step is to set up the return value for the `Comparison` to `false`, that way if at any point the comparison has a negative result,

execution can branch to the `endLabel`, at the end of the method and false is returned.

```
LocalBuilder[] retVals = new LocalBuilder[Values.Length];

for(int i = 0; i < Values.Length; i++)
{
        ilGenerator.BeginScope();
        Values[i].Compile(ilGenerator, ref retVals[i]);
        ilGenerator.EndScope();
}

Label endLabel = ilGenerator.DefineLabel();
ilGenerator.Emit(OpCodes.Ldc_I4_0);
ilGenerator.Emit(OpCodes.Stloc_S, retVal);
ilGenerator.Emit(OpCodes.Ldloc, retVals[0]);
for(int ii = 1; ii < retVals.Length; ii++)
{

        ilGenerator.Emit(OpCodes.Ldloc, retVals[ii]);
        Label nextLabel;

        switch(Operation)
        {

                case ComparisonOperation.Equals:

                        nextLabel = ilGenerator.DefineLabel();
                        ilGenerator.Emit(OpCodes.Beq S, nextLabel);
                        ilGenerator.Emit(OpCodes.Br S, endLabel);
                        ilGenerator.MarkLabel(nextLabel);
                        break;

                case ComparisonOperation.NotEquals:

                        ilGenerator.Emit(OpCodes.Beq_S, endLabel);
                        break;

                case ComparisonOperation.LessThan:

                        ilGenerator.Emit(OpCodes.Bge S, endLabel);
                        break;

                case ComparisonOperation.LessThanOrEqualTo:

                        ilGenerator.Emit(OpCodes.Bgt S, endLabel);
                        break;

                case ComparisonOperation.GreaterThan:

                        ilGenerator.Emit(OpCodes.Ble S, endLabel);
                        break;

                case ComparisonOperation.GreaterThanOrEqualTo:

                        ilGenerator.Emit(OpCodes.Blt S, endLabel);
                        break;

        }

        ilGenerator.Emit(OpCodes.Ldc_I4_1);
        ilGenerator.Emit(OpCodes.Stloc S, retVal);
        ilGenerator.MarkLabel(endLabel);

}
```

Figure 5-9 - Code snippet illustrating the compilation process of the Comparison class

For each of the values stored in the array the `for` loop makes a comparison of the current value against the last with the comparison algorithm being dependant on which `ComparisonOperation` was specified. The switch statement provides a relatively simple means of separating the comparison type for the logic of the comparison by using simple branching statements.

5.1.8 ValueType

The `ValueType` enum is used to describe the different types of value that can be represented by the `Value` class. These can be seen in Figure 5-10. There are two value types that are described in this enum that cannot be represented numerically which are the String type and the Parameter type; all of the others can be represented internally by, for example, the double data type.

```
/// <summary>
/// The ValueType enum is used to describe different types of
/// Values which can be used by a Value object.
/// </summary>
public enum ValueType
{
    /// <summary>
    /// A number which will be of a floating point type to allow
    /// for maximum compatibility as a general number.
    /// </summary>
    Number = 0,

    /// <summary>
    /// A quantity of days as a floating point to allow
    /// fractions of days.
    /// </summary>
    Days = 1,

    /// <summary>
    /// A quantity of months as a floating point to allow
    /// fractions of months.
    /// </summary>
    Months = 2,

    /// <summary>
    /// A quantity of years as a floating point to allow
    /// fractions of years.
    /// </summary>
    Years = 3,

    /// <summary>
    /// A quantity of hours as a floating point to allow
    /// fractions of hours.
    /// </summary>
    Hours = 4,

    /// <summary>
    /// A quantity of minutes as a floating point to allow
    /// fractions of minutes.
    /// </summary>
    Minutes = 5,
```

```
    /// <summary>
    /// A quantity of seconds as a floating point to allow
    /// fractions of seconds.
    /// </summary>
    Seconds = 6,

    /// <summary>
    /// A quantity of milliseconds as a floating point to allow
    /// fractions of days.
    /// </summary>
    Milliseconds = 7,

    /// <summary>
    /// A string value.
    /// </summary>
    String = 8,

    /// <summary>
    /// A property of the given period or a related period denoted
    /// by a seperation of decimal points within the description.
    /// </summary>
    Parameter = 9
};
```

Figure 5-10 - Code snippet illustrating the types of value that can be used by the Value class, as stored in the ValueType enum

5.1.9 Value

The `Value` class provides a fairly simple means of representing different values externally, to the consumer. Internally, however, the code emitted into the executable appears relatively complex.

For the sake of simplicity in the representation of data held by the rule engine, the double data type is used to store a large variety of types of data including numeric values, dates, times and parameter values. This concept of fixed sized data representation isn't uncommon and can be seen in other industrial systems such as MatLab. By using a double, we are able to hold the largest range of numeric values, with an approximate range of $\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$ providing an unrestrictive, future proof, common storage format for all numeric values.

The simplest of the value types to emit code for is the `Number` which can have its value stored as a double with simplicity. The code for emitting IL for dealing with a standard numeric value is seen in Figure 5-11.

```
newLocal = ilGenerator.DeclareLocal(typeof(double));
ilGenerator.Emit(OpCodes.Ldc_R8, double.Parse(Description));
```

Figure 5-11 - Code snippet illustrating the declaration and association of a simple numeric value, stored in a double data type

The management of the `Days` value type is dealt with slightly differently. Internally, all date and time value types are represented as whole and fractional hours. By standardising to a concept such as this, making comparisons between periods of time, further down the line, becomes much faster as it is simply the comparison of two variables of the double data type and not of a complex data type such as the `TimeSpan` or `DateTime` data types.

Figure 5-12 shows the IL for converting the number of days into hours representation and storing them within a double data type. Firstly the number of days is passed in to the static `FromDays` method of the `TimeSpan` object to produce a new `TimeSpan` object. The `TotalHours` property of the `TimeSpan` object is then used to complete the conversion into double precision hours.

```
newLocal = ilGenerator.DeclareLocal(typeof(double));
tmpLocal = ilGenerator.DeclareLocal(typeof(TimeSpan));
ilGenerator.Emit(OpCodes.Ldc_R8, double.Parse(Description));
ilGenerator.Emit(OpCodes.Call, typeof(TimeSpan).GetMethod("FromDays", new
Type[1] {typeof(double)}));
ilGenerator.Emit(OpCodes.Stloc_S, tmpLocal);
ilGenerator.Emit(OpCodes.Ldloca_S, tmpLocal);
ilGenerator.Emit(OpCodes.Call, typeof(TimeSpan).GetMethod("get_TotalHours", new
Type[0]));
```

Figure 5-12 - Code snippet illustrating the declaration and association of a number of days, stored in a double data type

Each of the other date and time value types has its functionality emitted in a similar way to that of the `Days` example in Figure 5-12. The `String` data type emits code similar to that given in Figure 5-11 except that it works with a `String` data type rather than a double data type.

The final and most complex of the value types is the `Parameter` value type. This value type describes a value which is obtained from either an element of rule testing data or of the rules themselves. For example, the required length of the daily rest period between two daily work periods.

Rather than spending the time emitting all of the IL necessary to evaluate the name of the property to the property value, a base class method to the `Period` object, of which the newly emitted class inherits, contains a `GetProperty` method which returns the appropriate value back to the caller. The implementation for invoking this method is shown in Figure 5-13.

```
newLocal = ilGenerator.DeclareLocal(typeof(double));
ilGenerator.Emit(OpCodes.Ldarg 0);
ilGenerator.Emit(OpCodes.Ldstr, Description);
ilGenerator.Emit(OpCodes.Call,
typeof(RuleSupport.Period).GetMethod("GetProperty", new Type[]
{typeof(String)}));
```

Figure 5-13 - Code snippet illustrating the declaration and obtainment of a Parameter value then stored in a double data type

5.1.10 MathOperation

The `MathOperation` enum describes the four basic math operations that can be carried out by the `Math` class. Figure 5-14 shows the code snippet for the `MathOperation` enum.

```
/// <summary>
/// The MathOperation enum describe a type of mathematical operation
/// which can be used with the Math class.
/// </summary>
public enum MathOperation
{
        /// <summary>
        /// The operation of adding two numbers together.
        /// </summary>
        Add,

        /// <summary>
        /// The operation of subtracting two numbers.
        /// </summary>
        Subtract,

        /// <summary>
        /// The operation of multiplying two numbers together.
        /// </summary>
```

```
        Multiply,

        /// <summary>
        /// The operation of dividing two numbers.
        /// </summary>
        Divide
};
```

Figure 5-14 - Code snippet illustrating the math operations permitted for use with the Math class.

5.1.11 Math

The `Math` class is able to emit the executable code to carry out the four math operations on a given set of values. Figure 5-15 shows the code snippet for the `Compile` method of the `Math` class. The purpose is to loop through all of the values provided and store each of them within a local variable. The next step is to loop through each variable, performing the math operation on each, one after the other.

Although the number of math operations that are currently supported is very limited, it would not be overly complex to expand on these by providing additional operations within the `MathOperation` enum and then adding the matching functionality to the `Compile` method of the `Math` class. To establish a proof of concept, however, the four main operations are considered adequate.

```
newLocal = ilGenerator.DeclareLocal(typeof(double));

ilGenerator.BeginScope();

// Start by generating all of the local variables which will
// be used.
LocalBuilder[] valueLocals = new LocalBuilder[Values.Length];
for(int i = 0; i < valueLocals.Length; i++)
{
        Values[i].Compile(ilGenerator, ref valueLocals[i]);
}

// If you are adding numbers, you can start at 0.0,
// but if you are subtracting, of course it wont work!
ilGenerator.Emit(OpCodes.Ldloc_S, valueLocals[0]);

for(int ii = 1; ii < valueLocals.Length; ii++)
{
        ilGenerator.Emit(OpCodes.Ldloc_S, valueLocals[ii]);

        switch(Operation)
        {
                case MathOperation.Add:
```

```
                        ilGenerator.Emit(OpCodes.Add);
                        break;
                case MathOperation.Subtract:
                        ilGenerator.Emit(OpCodes.Sub);
                        break;
                case MathOperation.Multiply:
                        ilGenerator.Emit(OpCodes.Mul);
                        break;
                case MathOperation.Divide:
                        ilGenerator.Emit(OpCodes.Div);
                        break;
        }
}

// Store the result of the math operation in our new local
// variable.
ilGenerator.Emit(OpCodes.Stloc S, newLocal);

ilGenerator.EndScope();
```

Figure 5-15 - Code snippet illustrating the compilation process of the Math class

5.2 Rule Support

The various rule support classes are fundamental to the overall process of making the rule engine function. Whilst much of the rule specific functionality has been designed to compile into an executable module and run, there are certain elements of functionality that it would be an overhead to emit as part of the compilation process and, as such, are included as part of a separate, pre-defined, set of functionality.

All rule support classes fall under the `RuleSupport` namespace. There are some classes which come under the `RuleSupport` namespace that share the same name as some of the classes which fall within the `RuleDefinitionLanguage` namespace although there are some key differences. The `RuleDefinitionLanguage` namespace classes are for use as compile-time classes (classes used for compilation of the rules into executable code) and for rule definition and serialisation into XML. Those classes with similar names that fall under the `RuleSupport` namespace have a different goal as they are used during run-time to provide support to the compiled rules as either base classes or automation classes, carrying out certain elements of the rule testing functionality that it would be inefficient to emit into the rules individually, at compile-time.

5.2.1 IPeriodElement

An important interface for use by the consuming application is the `IPeriodElement` interface. This interface serves as the basis for any type of timed element such as an Appointment, when used with transport scheduling, or a Class, when used in timetable scheduling, as it specifies that a deriving class must have a `Start` and `Finish` as well as a `WorkLength` and `RestLength`. These are essential elements when looking at temporal based rules and by using an interface to define these, it is possible for a consuming application to simply implement the functionality of this interface to provide an integration mechanism for using the rule engine.

5.2.2 ITestable

Another important interface for the rule engine is the `ITestable` interface. This defines an interface for any type that has the ability to carry out a test and return a Boolean value as a result. All rules described using the rule definition language implement the `ITestable` interface and have a `Test` method along with the other components required to be `ITestable`. There is the potential with an interface such as this to allow the consuming application to develop predefined classes that implement the `ITestable` interface, to be tested as part of the rule testing process.

As well as requiring the consuming class to have a `Test` method, the `ITestable` interface also states that the consuming class must provide two events to allow them access to the `IPeriodElement` objects that may be connected to them within a work pattern. Details of this feature are presented in Section 5.3.1.

5.2.3 Period

The `Period` class, which is located under the `RuleSupport` namespace, has been designed to provide runtime support to compiled rules and serves as a base class for the `WorkPeriod` class and the `RestPeriod` class. It provides certain key functionality that many types of `Period` may find useful. The `Period` class implements both the `IPeriodElement` interface, providing `Start` and `Finish` times as well as `WorkLength` and `RestLength`, and also the `ITestable` interface as it provides a `Test` method and the two events used to gain access to other `IPeriodElements` relative to itself in the work pattern. Details of this feature are presented in Section 5.3.1.

One of the most useful and complex methods of the `Period` class is the `GetProperty` method. This method is designed to evaluate property, specified as text in the rule definition, to its value at runtime, similar to the late-binding mechanisms found in some modern programming languages. When, for example, in a rule definition a `Value` type is specified with a `ValueType` enum of `Property` and a value of `PRE.Start`, the `GetProperty` method will be used to obtain the `Start` value of the `IPeriodElement` that comes before the current `IPeriodElement` within the work pattern.

5.2.4 WorkPeriod

The `WorkPeriod` class provides a basic set of features that would be common to a period of time which describes a working element. The `WorkPeriod` class is the base class for all compiled rules described as a `WorkPeriod` and provides property implementations for `Start`, `Finish`, `WorkLength` and `RestLength` based upon the elements contained with it.

As an example, if the `WorkPeriod` rule described a Daily Work Period, in the transport scheduling problem, the `WorkPeriod` object would, for the

`Start` property, return the earliest start time based upon the appointments that would be contained within; the `Finish` property would return the latest finish time of the appointments contained within; the `WorkLength` would return the sum of the `WorkLength`s of the appointments contained within and the `RestLength` would return the sum of the `RestLength`s of the appointments contained within.

5.2.5 RestPeriod

The `RestPeriod` class provides a basic set of features that are common to all periods of time that describe a period of rest. All compiled rules that are described as a `RestPeriod` will extend the `RestPeriod` class.

The `RestPeriod` class implements the `Start`, `Finish`, `WorkLength` and `RestLength` properties but in a different way to the `WorkPeriod` class. A `RestPeriod` would not normally contain any other `Period` objects as it describes a gap between one `WorkPeriod` and another. The `Start` property of a `RestPeriod` is obtained by looking at the previous period in the work plan and obtaining its `Finish` property. Similarly, the `Finish` property is obtained by looking forward in the work plan and obtaining the `Start` property of the `WorkPeriod` which follows. The `WorkLength` property of the `RestPeriod` class also returns a 0 value as there is no work within a period of rest. The `RestLength` returns the difference between the periods `Start` property and the periods `Finish` property.

5.2.6 Specialised Classes

Certain classes provided by the .NET Framework needed to be extended in order to provide additional functionality. The first example of this is the

`TimeSpan` class which describes a period of time. The original `TimeSpan` class found under .NET's `System` namespace does not provide the ability to convert the `TimeSpan` into the `System.DateTime` data type or make comparison between a `TimeSpan` and a `DateTime`. A specialist version of the `TimeSpan` data type was created which can offer this additional functionality.

Similarly, the `List` class found under the `System.Collections.Generic` namespace required additional functionality currently not supported. The `List` class is a generic class providing the ability to store a dynamically sizable array of a specialist data type. The additional functionality required from this was the ability to describe the content of the `List` as 'dirty'. This means that the items within the `List` had changed in some way since the last time the `List` was accessed. An example of this type of functionality is in storing the `IPeriodElement` object contained within a `Period`. If the `List` has not changed then we can return to a calling method the cached results of the `Start`, `Finish`, `WorkLength` and `RestLength` properties to reduce the processing requirement at runtime. If the `List` cannot describe whether its contents have changed then it is not possible to decide whether we can cache various properties. Additionally, the new `DirtyList` class has a fairly advanced `Sort` method that is able to sort its contents based upon a predicate describing the value of either a contained variable, a property or the result of a method call. This can support the process of sorting contained `IPeriodElement` objects within the `List` by the `Start` property, amongst other things.

5.3 Rule Testing

The functionality that carries out the rule testing process is located under the `RuleTesting` namespace. The testing method provided is very basic and not optimised to support advanced methods of producing work plans as it is

not considered the focus of the research presented within this thesis. This is discussed further in Section 9.5.1.

5.3.1 WorkPatternCollection

The `WorkPatternCollection` class provides a range of functionality that supports the organisation of the data, provided by the host application in the form of objects which implement the `IPeriodElement` interface, into a useable work pattern. The class inherits from the `DirtyList` class.

One of the main elements of functionality that is provided by the class is the ability to describe appointments that may come before or after a particular period. For example, the `RestPeriod` class requires the ability to obtain its `Start` property based upon the `Finish` property of the `WorkPeriod` that comes before it. The `WorkPatternCollection` class provides a means for each of the periods within the pattern to access other periods also organised within the pattern, by their organisational relationship.

The class also provides the ability to serialise itself into XML so that the work pattern can be saved to disk, transported across a network or displayed in human readable format to the user. This helps to see how the system has organised, based on the rules, the data into a workable schedule or, alternatively, where the system believes a rule has been broken.

5.3.2 Tester

The `Tester` class is the main class used for managing the rule testing process. Whilst the actual rules themselves are compiled, it is the Tester class that has the task of organising the data provided by the host application into a useable work pattern, with the help of the `WorkPatternCollection` class and the rules themselves. The class takes many different elements of information into consideration relating to the way the rules were originally

described. Each rule provides a description of the data or rules contained within or that come before or after it. The `Tester` class uses this information in order to arrange the data into the work pattern and invoke the compiled rule testing process to determine whether rules have been broken by its organisation.

The `Tester` class has a two stage testing process. The first stage, given in Figure 5-16, is used to get the testing process started by adding the first `WorkPeriod` to the work pattern and seeing if the first element of data can be added to it without breaking the rules. The first stage calls the second stage, a recursive method given in Figure 5-17, to carry out the process of adding the remaining data elements to the work pattern and to build up the work pattern to include different types of period, as necessary.

The result of both stages is a simple Boolean value indicated the success or failure of being able to add an element of data to the work pattern. If the process fails it is because, when compared against the compiled rules provided, the data cannot be organised into the work pattern generated.

The implementation of this algorithm can be found under Section 11.3.4. The two methods are quite simple in design and, as stated in section 9.5, there is future work to be done in this area to further improve the algorithm used.

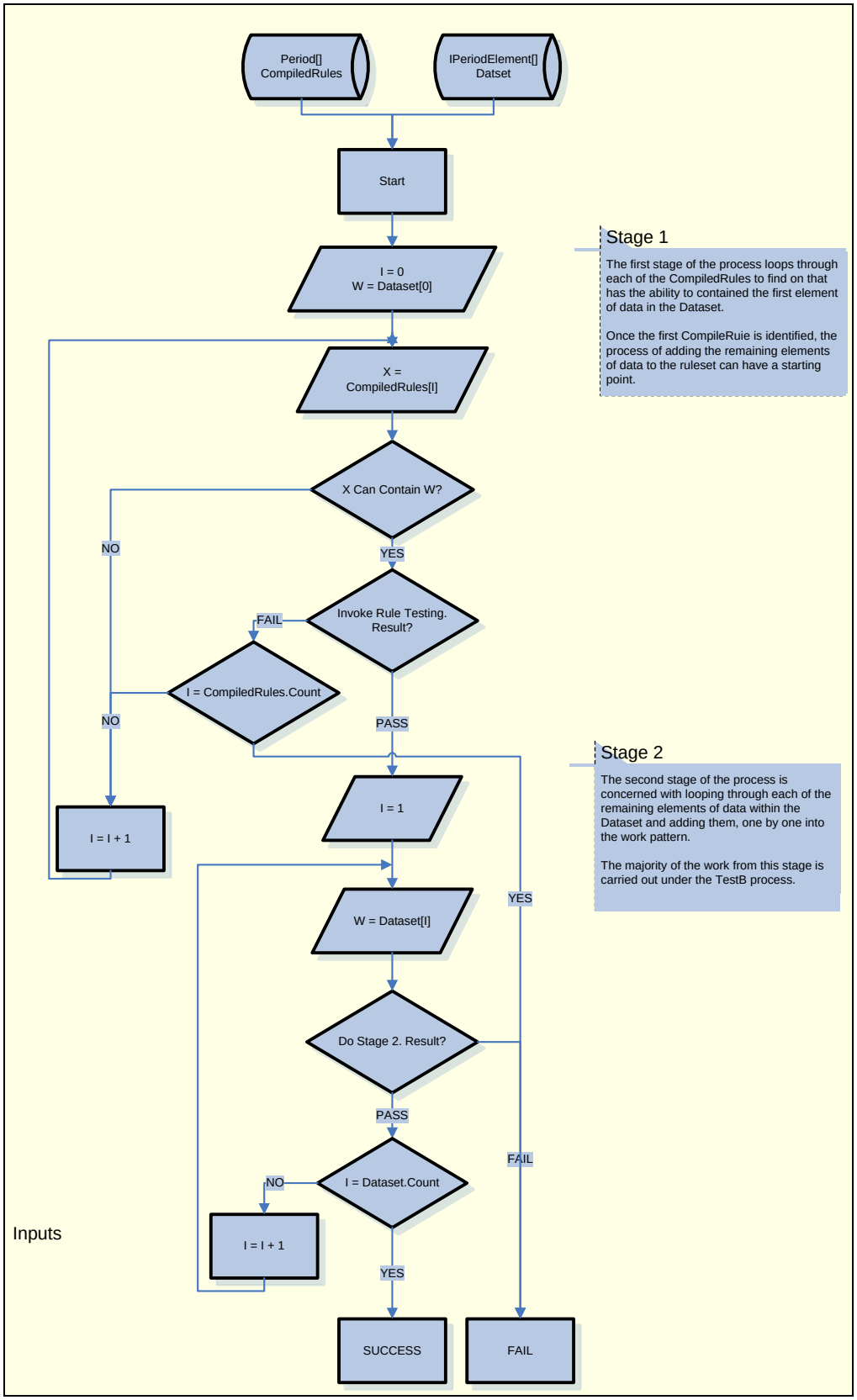


Figure 5-16 – Flowchart illustrating Stage 1 of the automated Testing process

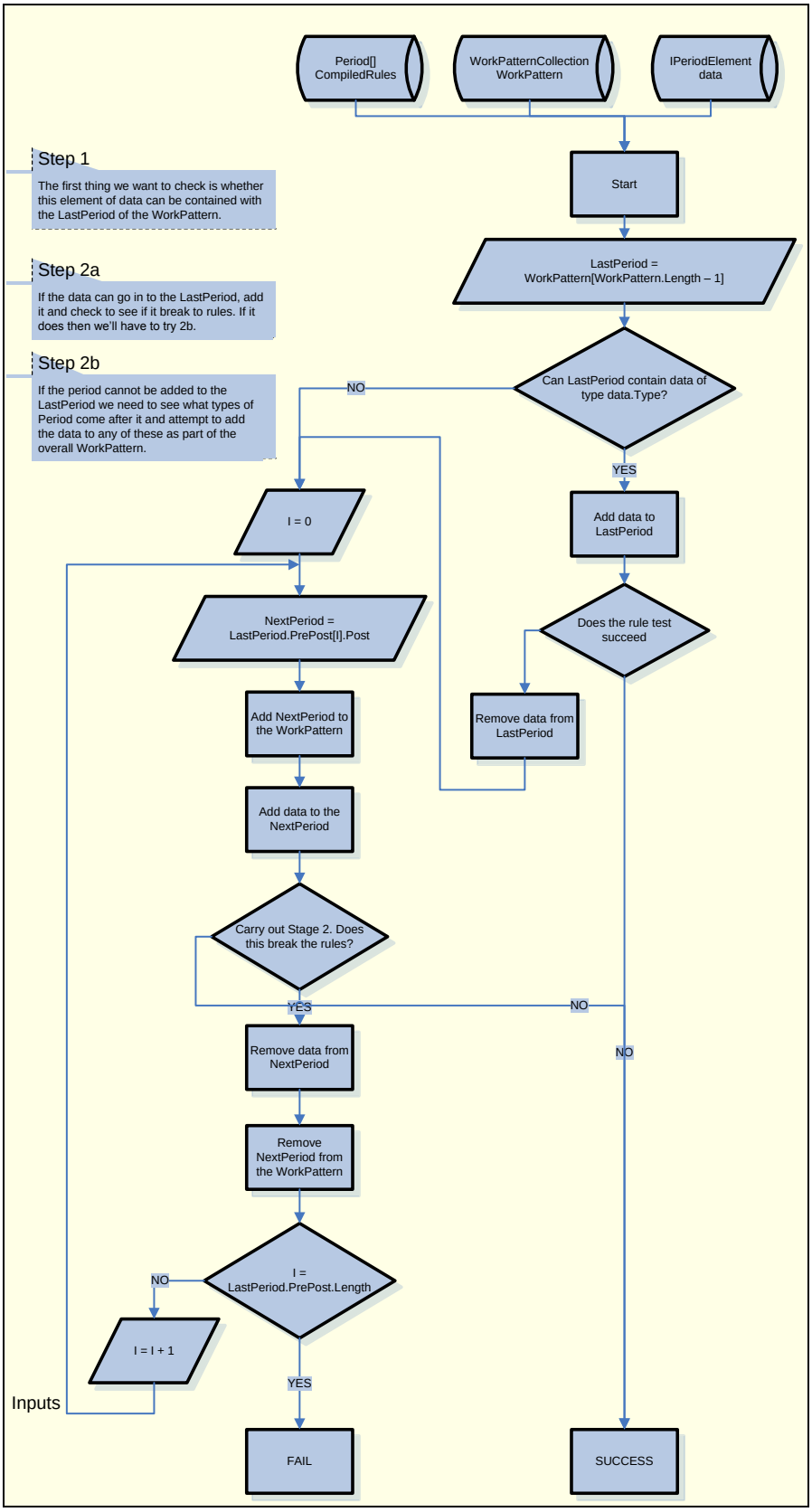


Figure 5-17 – Flowchart illustrating Stage 2 of the automated Testing process

5.4 Summary

This chapter of the thesis has illustrated some of the implementation and further design work that has taken place during the Construction phase of the reported research. It is hoped that this has resulted in a much clearer definition of a more optimal solution for the technical implementation of the research and provides a means of moving forward, to develop a test harness for the collection of statistical results.

The next chapter of the thesis aims to discuss the testing section of the research, both in terms of the development of a test harness as well as a discussion of the results obtained from running the test harness.

6 TESTING

The preceding chapters have set out the key research requirements along with the design and implementation of the rule testing software.

Over the course of the next 3 chapters, Testing, Initial Results and Further Results, this thesis will look at the overall testing process for the given solution. The purposes of these three chapters are to:

- Define an initial set of test procedures to help demonstrate the functionality of the prototype solution.
- Obtain measurements to gauge the performance of the proposed solution across various computing hardware to help determine some form of baseline in performance and to understand idiosyncrasies that affect the solutions performance.
- Analyse the solutions performance under varying conditions including a look at alternate rulesets, execution environments and operating systems in order to determine the effects these things have on the performance of the solution.

The Testing chapter will begin by identifying a programme for testing the proposed solution both in terms of standard software testing, to support the bug removal process, and also to gauge the performance of the proposed solution. Following on from the design of the test programme a design for a test harness is demonstrated that can help automate the main testing process.

The Initial Results chapter presents the findings of the tests carried out to determine the performance of the proposed solution within the initial testing scenario. It draws some conclusions as to how the results obtained differ from what may have been expected and some of the issues that affect the solutions performance. This chapter also discusses the issues found when attempting draw comparison of the results against alternative solutions.

The Further Results chapter looks at the outcome of additional work including a comparison of the performance of the initial testing work with that of a ruleset from an alternate application domain. Additionally this chapter analyses the performance results of tests taken to understand the solutions performance under alternative conditions such as the solutions execution using an alternate execution environment, i.e. Mono instead of the .NET Framework, and the performance comparison across multiple operating systems. The results should help to gauge the optimal conditions under which the proposed solution should operate to receive the optimal performance.

6.1 Test Programme

The test programme is used to define the kinds of results required from the testing process and how these results will be obtained.

Operationally, the testing process needs to identify whether the developed software solution will be able to carry out the functions for which it was designed. It will be important to be able to provide a set of rules and a set of data to the software under test to determine whether the solution can:

a. be integrated into a system requiring rule testing (see Section 3.1)
b. read a set of rules from the external system (see Section 3.2)
c. compile the rules into an executable form (see Section 3.3)
d. test a set of data against the compiled rules

Subject to the software under test being capable of meeting the above criteria, it would be useful to be able to obtain an initial set of data to help understand the proposed solutions overall performance, both in terms of computational performance and memory usage.

As previously highlighted, the need for this research has spawned from the transport scheduling application domain. Therefore, in order to ensure the proposed solutions validity for this real-world application the rules and data to

be tested will be derived from those outlined in (Transport, 1998). Once the initial set of operational tests are completed it will be possible to determine any bottlenecks or other areas of improvement for the proposed solution.

6.1.1 Test Scenario

The initial scenario used for testing the solution and gaining measurable results comes from the transport scheduling field as described in (Transport, 1998).

Rather than making comparisons based upon the entire set of legal driving rules, a small subset of rules have been used. The reason for this is that the complete set of legal driving rules are very complex and usually require a domain expert to understand and implement that complete set manually. For the purpose of the testing programme, a subset is suitable for non-domain experts to be able to determine the operational performance of the given solution. The complete set of rules are highly complex are would potentially detract from the purpose of the testing programme. A more manageable subset of the rules would help to prove operational effectiveness of the programme under test as well as produce a useful baseline as an indication of the expected performance if faced with the complete ruleset.

The test data itself relates to a single drivers schedule, as opposed to a set of drivers and their schedules. This is because, in the legal driving rules scenario specifically, the rules themselves relate to an individual drivers schedule, not a set of driver schedules. In order to test multiple driver schedules, the process could be carried out in series or in parallel as the testing process of driver schedules are not dependent upon each other.

6.1.1.1 Ruleset

The rules used for obtaining results for comparison in this section are as follows:

- A daily work period can be up to 8 hours long.
- A weekly work period can be up to 6 daily work periods long.
- A daily rest period must be at least 8 hours long.
- A weekly rest period must be at least 30 hours long.

Clearly, these rules have additional metadata to explain how they work together and which can be used by the work pattern creation process with the dataset that will be provided.

The metadata for the rules, as well as the rule data itself, can be seen in Figure 6-1 for the four rules described. The metadata provides details of how the rules work together both in what types of rules can come before or after a particular rule as well as those types that can be contained within a particular period.

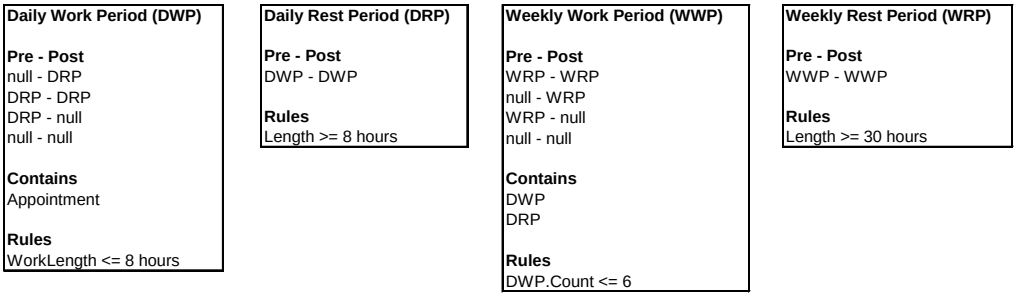


Figure 6-1 – Illustration showing the four testing rules together with their supporting metadata

6.1.1.2 Dataset

The first set of test data used for this experiment consists of a number of appointments, each with a start and finish time. For the benefit of simplifying the testing scenario, it is to be assumed that all appointments require the driver to drive between the start and finish times of the appointment and that no in-between rest breaks will be taken. This simplifies the process further by not requiring the test bench to consider travel times and distances between locations, which are not a factor necessary for consideration when the focus is on the rule testing process and not the scheduling process.

For consistency during the testing process, it is important that all tests conducted start from the same date, in order that oddities such as the number of days in a month, or leap years, do not affect the outcome or the test results. To support this, the start date of each appointment is relative to 01/01/2007, providing an initial start date which is always on a Monday (the start of a normal working week). This provides for a convenient means of automatic appointment generation, relative to a known starting date as well as a fixed consistent set of data which can be replicated automatically under varying conditions, as shown in Figure 6-2.

Appointment	Start Time	Finish Time
1	01/01/2007 09:00	01/01/2007 14:00
2	01/01/2007 15:00	01/01/2007 16:00
3	02/01/2007 09:00	02/01/2007 14:00
4	02/01/2007 15:00	02/01/2007 16:00
5	03/01/2007 09:00	03/01/2007 14:00
6	03/01/2007 15:00	03/01/2007 16:00
7	05/01/2007 09:00	05/01/2007 14:00
8	05/01/2007 15:00	05/01/2007 16:00
9	06/01/2007 09:00	06/01/2007 14:00
10	06/01/2007 15:00	06/01/2007 16:00

Figure 6-2 – One Week Appointment Dataset

This range of appointments takes place over a period of 7 days, with two days off on days 4 and 7. A graphical representation of this data can be seen in Figure 6-3.

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
09:00 - 10:00	1	3	5		7	9	
10:00 - 11:00							
11:00 - 12:00							
12:00 - 13:00							
13:00 - 14:00							
14:00 - 15:00							
15:00 - 16:00	2	4	6		8	10	
16:00 - 17:00							
17:00 - 18:00							

Figure 6-3 – One Week Dataset Represented Graphically

In order to generate large quantities of valid data automatically to carry out tests of the proposed rule testing system, a simple method was designed that

took the 10 appointments outlined above and incremented their dates in order to place them in different weeks. An example of a three week, 30 appointment, dataset can been see in Figure 6-4.

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10	Day 11	Day 12	Day 13	Day 14	Day 15	Day 16	Day 17	Day 18	Day 19	Day 20	Day 21
09:00 - 10:00																					
10:00 - 11:00																					
11:00 - 12:00	1	3	5		7	9		11	13	15		17	19		21	23	25		27	29	
12:00 - 13:00																					
13:00 - 14:00																					
14:00 - 15:00																					
15:00 - 16:00	2	4	6		8	10		12	14	16		18	20		22	24	26		28	30	
16:00 - 17:00																					
17:00 - 18:00																					

Figure 6-4 – The three week driver schedule dataset represented graphically

A large variety of dataset sizes have been chosen to test against the rules. The dataset size ranges from 250 to 10,000 appointments at increments of 250. The upper boundary far exceeds the normal upper limit of appointments typically found in the transport scheduling scenario. The proposed solution does, however, have to be considered for potential future applications which may have large dataset sizes and an understanding of the testing performance for future scenarios is important. Additionally it should provide some interesting results to help understand the systems performance and potential areas for improvement.

6.1.2 Test Plan

With these rules clearly defined a number of different approaches can be taken to generate results for comparison. In the transport scheduling field, the original client would pay a domain expert to look over a drivers work pattern to determine whether there were any infringements to the legal driving rules. As this is the original business case, this must be set as a test scenario. The automated system proposed in this research could, therefore, be testing against the manual process to measure the performance gains in the driver scheduling scenario.

In addition to manual test scenarios, automated test scenarios are necessary to see how the solution performs across a range of modern computer systems.

The areas being used for comparison are based on varying processor speeds, quantities of memory and operating systems.

The table shown in Figure 6-5 provides the technical specifications of the systems used. These systems were chosen due to their availability in an average organisation. The majority of organisations upgrade or replace their computer systems on a three year rolling programme. This set of computer systems indicates a reasonable representation of systems from new to 3 years of age which would typically be the specification of the target system for the proposed solution.

Identifier	Processor Type	Processor Speed	RAM Size	Operating System
A	Intel Pentium 4	3 GHz	1 GB	XP Service Pack 2
B	Intel Pentium 4	3 GHz	2 GB	XP Service Pack 2
C	Intel Pentium 4	2.4 GHz	1 GB	XP Service Pack 2
D	Intel Core Duo (Centrino Duo)	2.33 GHz	4 GB	Vista x86

Figure 6-5 - Computer Specifications Used In Testing

6.2 Test Bench

In order to produce a set of test results which demonstrate the proposed solution can meet the requirements set out in Chapter 3, a test bench had to be implemented which was capable of consuming the functionality of the proposed solution, as if it were a scheduling system, and which could perform some analysis on the results obtained. Using rapid application development (RAD), a relatively simple graphical user interface (GUI) was designed that would be able to call the functions of the Rule Testing System to carry out the testing process and display the results in a useful way. Figure 6-6 shows the simple GUI design.

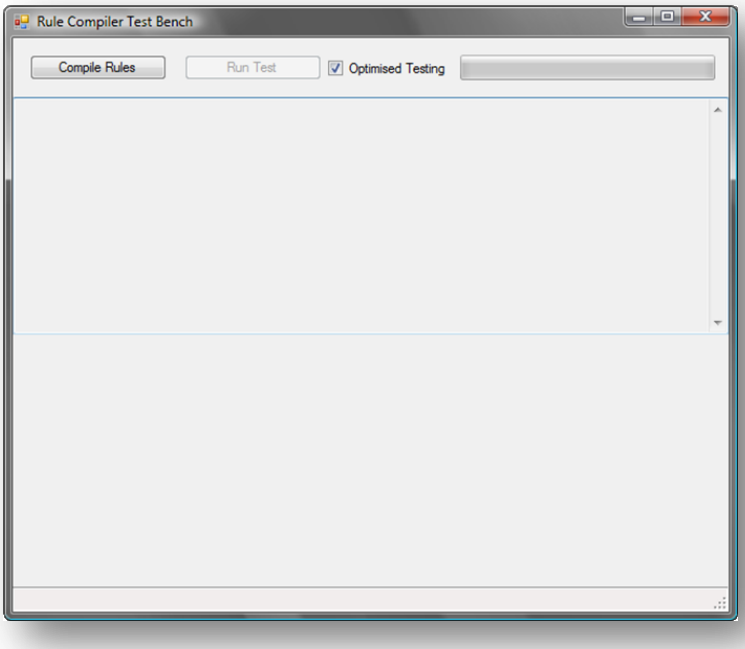


Figure 6-6 – Screen capture of the test bench

The two buttons to the top of the GUI provide the main interface with the program. The Compile Rules button carries out the task of compiling a set of rules from their original descriptive form. Whilst the test bench itself uses rules that are hard-coded, the rules can, in reality, be imported into a system through any medium that can store or transmit XML.

Once the rules have been compiled, a serialised copy of the rules are displayed to the user in the text box which spans the top half of the screen, as can be seen in Figure 6-7.

The next button at the top of the screen begins the rule testing process using the compiled rules. After pressing this button a new thread is created and the testing process begins in the new thread. This frees up the GUI's thread to focus on keeping the graphical interface up to date.

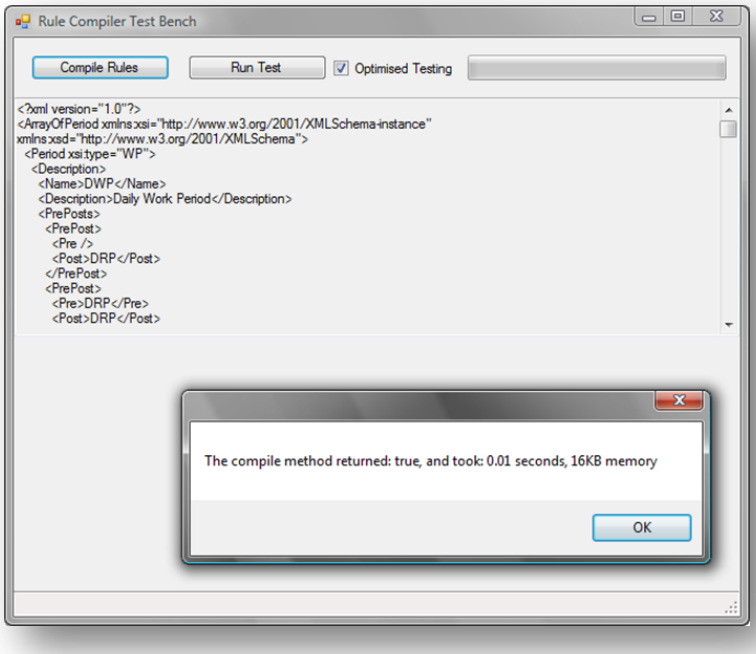


Figure 6-7 - Screen capture of the test bench after rule compilation has occurred

The text box occupying the top half of the screen outputs the progress of the testing process whilst the picture box which occupies the lower half of the screen displays a simple line graph of the test results which have been obtained so far. An example of this stage of execution can be seen in Figure 6-8.

The results being displayed in the text box are being displayed in a format known as Comma Separated Values (CSV) which is a format that can easily be imported into a spreadsheet for statistical analysis and processing at a later point.

Once processing is under way, the user can either wait for the processing to complete (in this tests case, wait for 10,000 elements of data to be processed) or the user can press the Stop Test button (formally Run Test) in order to end the test early and take the test results away for analysis.

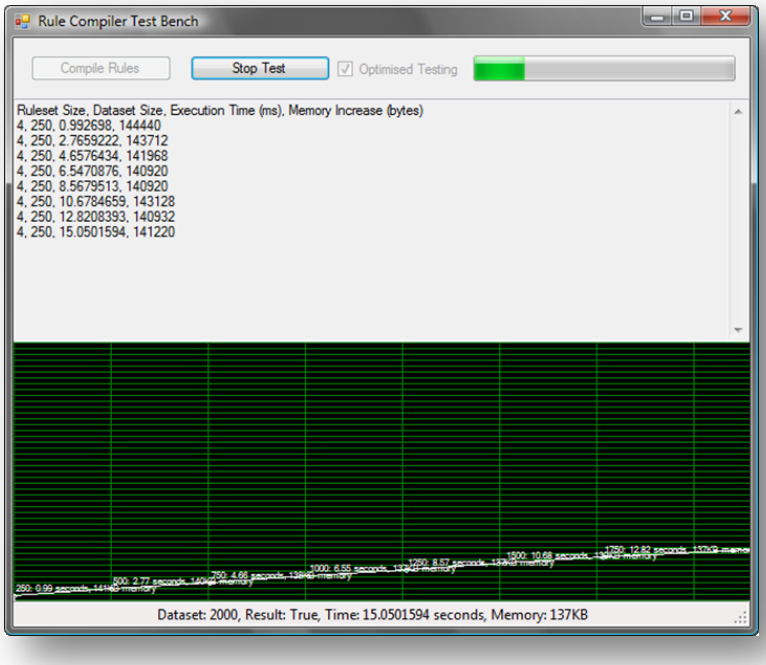


Figure 6-8 - Screen capture of the test bench during the testing process

(Note: the second column represents the increment made, not the dataset size, as indicated in the column header)

At the end of the testing process the result summary of the test is returned to the user in the form of a message box summarising the overall processing time and the information that is left in the text box, as can be seen in Figure 6-9.

The final task, as far as the test program goes, is to copy the data from the text box and paste it into a text editor, such as notepad and save it with the extension .csv, ready to open within a spreadsheet application.

6.2.1 Test Bench Logic

Whilst much of the appearance of the test bench is fairly simplistic, its implementation is far more complex. For example, a new data type had to be defined to implement the `IPeriodElement` interface, described in Figure 4-3. The full implementation of this class can be found in Section 11.3.5.3.

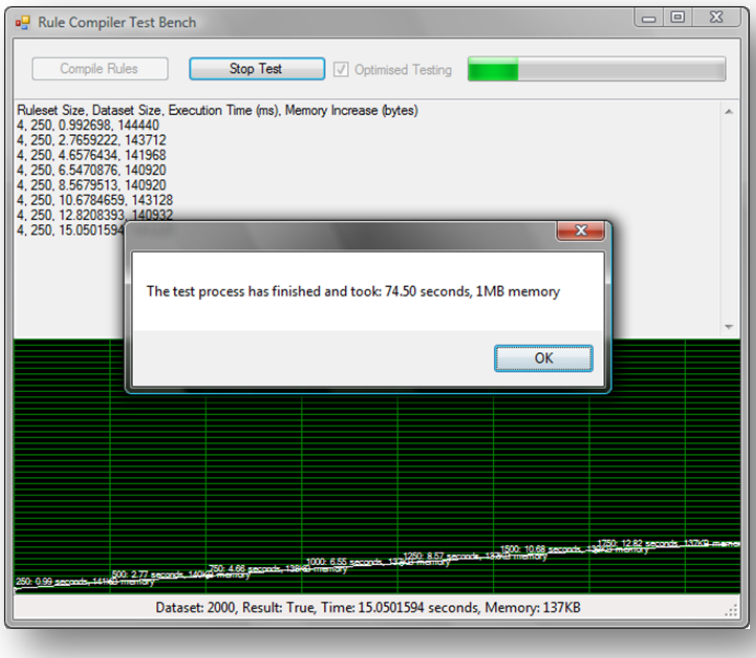


Figure 6-9 - Screen capture of the test bench after the testing process has finished

As previously stated, the rules, in this instance, were hard coded. This was merely for simplicity and could just have easily been provided from an external source in a serialised form. The code snippets which illustrate the implementation of the description of these rules can be seen in Figure 6-10, Figure 6-11, Figure 6-12 and Figure 6-13. An alternate means of describing the same set of rules is using XML and this equivalent can be found seen in Figure 6-14.

```
// A daily work period can be up to 8 hours long.
RuleDefinitionLanguage.WP myDWP = new RuleDefinitionLanguage.WP();
myDWP.Description.Name = "DWP";
myDWP.Description.Description = "Daily Work Period";
myDWP.Description.PrePosts = new RuleDefinitionLanguage.PrePost[] { new RuleDefinitionLanguage.PrePost("", "DRP"), new
RuleDefinitionLanguage.PrePost("DRP", "DRP"), new RuleDefinitionLanguage.PrePost("DRP", ""), new RuleDefinitionLanguage.PrePost("", "") };
myDWP.Description.Contains = new RuleDefinitionLanguage.Contain[] { new RuleDefinitionLanguage.Contain("Appointment") };
myDWP.Rules = new RuleDefinitionLanguage.AndCondition(new RuleDefinitionLanguage.Comparitor[] { new
RuleDefinitionLanguage.Comparison(RuleDefinitionLanguage.ComparisonOperation.LessThanOrEqualTo, new RuleDefinitionLanguage.Value[] { new
RuleDefinitionLanguage.Value("WorkLength", RuleDefinitionLanguage.ValueType.Parameter), new RuleDefinitionLanguage.Value("8",
RuleDefinitionLanguage.ValueType.Hours) }, null), new RuleDefinitionLanguage.Comparison(RuleDefinitionLanguage.ComparisonOperation.GreaterThan,
new RuleDefinitionLanguage.Value[] { new RuleDefinitionLanguage.Value("Appointment.Count", RuleDefinitionLanguage.ValueType.Parameter), new
RuleDefinitionLanguage.Value("0", RuleDefinitionLanguage.ValueType.Number) }, null) });
theRules.Add(myDWP);
```

Figure 6-10 - Code snippet showing how the Daily Work Period rule was defined within the testing bench


```
// A weekly work period can be up to 6 daily work periods long.
RuleDefinitionLanguage.WP myWWP = new RuleDefinitionLanguage.WP();
myWWP.Description.Name = "WWP";
myWWP.Description.Description = "Weekly Work Period";
myWWP.Description.PrePosts = new RuleDefinitionLanguage.PrePost[] { new RuleDefinitionLanguage.PrePost("WRP", "WRP"), new
RuleDefinitionLanguage.PrePost("", "WRP"), new RuleDefinitionLanguage.PrePost("WRP", ""), new RuleDefinitionLanguage.PrePost("", "") };
myWWP.Description.Contains = new RuleDefinitionLanguage.Contain[] { new RuleDefinitionLanguage.Contain("DWP"), new
RuleDefinitionLanguage.Contain("DRP") };
myWWP.Rules = new RuleDefinitionLanguage.AndCondition(new RuleDefinitionLanguage.Comparitor[] { new
RuleDefinitionLanguage.Comparison(RuleDefinitionLanguage.ComparisonOperation.LessThanOrEqualTo, new RuleDefinitionLanguage.Value[] { new
RuleDefinitionLanguage.Value("DWP.Count", RuleDefinitionLanguage.ValueType.Parameter), new RuleDefinitionLanguage.Value("6",
RuleDefinitionLanguage.ValueType.Number) }, null), new
RuleDefinitionLanguage.Comparison(RuleDefinitionLanguage.ComparisonOperation.GreaterThan, new RuleDefinitionLanguage.Value[] { new
RuleDefinitionLanguage.Value("DWP.Count", RuleDefinitionLanguage.ValueType.Parameter), new RuleDefinitionLanguage.Value("0",
RuleDefinitionLanguage.ValueType.Number) }, null) });
theRules.Add(myWWP);
```

Figure 6-11 - Code snippet showing how the Weekly Work Period rule was defined within the testing bench

```
// A weekly rest period must be at least 30 hours long.
RuleDefinitionLanguage.RP myWRP = new RuleDefinitionLanguage.RP();
myWRP.Description.Name = "WRP";
myWRP.Description.Description = "Weekly Rest Period";
myWRP.Description.PrePosts = new RuleDefinitionLanguage.PrePost[] { new RuleDefinitionLanguage.PrePost("WWP", "WWP") };
myWRP.Description.Contains = new RuleDefinitionLanguage.Contain[] { };
myWRP.Rules = new RuleDefinitionLanguage.Comparison(RuleDefinitionLanguage.ComparisonOperation.GreaterThanOrEqualTo, new
RuleDefinitionLanguage.Value[] { new RuleDefinitionLanguage.Value("RestLength", RuleDefinitionLanguage.ValueType.Parameter), new
RuleDefinitionLanguage.Value("30", RuleDefinitionLanguage.ValueType.Hours) }, null);
theRules.Add(myWRP);
```

Figure 6-12 – Code snippet showing how the Weekly Rest Period rule was defined within the testing bench


```
// A daily rest period must be at least 8 hours long.
RuleDefinitionLanguage.RP myDRP = new RuleDefinitionLanguage.RP();
myDRP.Description.Name = "DRP";
myDRP.Description.Description = "Daily Rest Period";
myDRP.Description.PrePosts = new RuleDefinitionLanguage.PrePost[] { new RuleDefinitionLanguage.PrePost("DWP", "DWP") };
myDRP.Description.Contains = new RuleDefinitionLanguage.Contain[] { };
myDRP.Rules = new RuleDefinitionLanguage.AndCondition(new RuleDefinitionLanguage.Comparitor[] { new
RuleDefinitionLanguage.Comparison(RuleDefinitionLanguage.ComparisonOperation.GreaterThanOrEqualTo, new RuleDefinitionLanguage.Value[] { new
RuleDefinitionLanguage.Value("RestLength", RuleDefinitionLanguage.ValueType.Parameter), new RuleDefinitionLanguage.Value("8",
RuleDefinitionLanguage.ValueType.Hours) }, null), new RuleDefinitionLanguage.Comparison(RuleDefinitionLanguage.ComparisonOperation.LessThan,
new RuleDefinitionLanguage.Value[] { new RuleDefinitionLanguage.Value("RestLength", RuleDefinitionLanguage.ValueType.Parameter), new
RuleDefinitionLanguage.Value("30", RuleDefinitionLanguage.ValueType.Hours) }, null) });
theRules.Add(myDRP);
```

Figure 6-13 - Code snippet showing how the Daily Rest Period rule was defined within the testing bench

```
<?xml version="1.0"?>
<ArrayOfPeriod xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Period xsi:type="WP">
    <Description>
      <Name>DWP</Name>
      <Description>Daily Work Period</Description>
      <PrePosts>
```

```
        <PrePost>
          <Pre />
          <Post>DRP</Post>
        </PrePost>
        <PrePost>
          <Pre>DRP</Pre>
          <Post>DRP</Post>
        </PrePost>
```

```
    </Rules>
  </Period>
  <Period xsi:type="WP">
    <Description>
      <Name>WWP</Name>
      <Description>Weekly Work Period</Description>
      <PrePosts>
        <PrePost>
          <Pre>WRP</Pre>
          <Post>WRP</Post>
        </PrePost>
        <PrePost>
          <Pre />
          <Post>WRP</Post>
        </PrePost>
        <PrePost>
          <Pre>WRP</Pre>
          <Post />
        </PrePost>
        <PrePost>
          <Pre />
          <Post />
        </PrePost>
      </PrePosts>
      <Contains>
        <Contain>
          <Name>DWP</Name>
        </Contain>
        <Contain>
          <Name>DRP</Name>
        </Contain>
      </Contains>
    </Description>
    <Rules xsi:type="AndCondition">
      <Conditions>
        <Comparitor xsi:type="Comparison">
          <Operation>LessThanOrEqualTo</Operation>
          <Values>
            <Value>
              <Description>DWP.Count</Description>
              <Type>Parameter</Type>
            </Value>
            <Value>
              <Description>6</Description>
```

```
        <PrePost>
          <Pre>DRP</Pre>
          <Post />
        </PrePost>
        <PrePost>
          <Pre />
          <Post />
        </PrePost>
      </PrePosts>
      <Contains>
        <Contain>
          <Name>Appointment</Name>
        </Contain>
      </Contains>
    </Description>
    <Rules xsi:type="AndCondition">
      <Conditions>
        <Comparitor xsi:type="Comparison">
          <Operation>LessThanOrEqualTo</Operation>
          <Values>
            <Value>
              <Description>WorkLength</Description>
              <Type>Parameter</Type>
            </Value>
            <Value>
              <Description>8</Description>
              <Type>Hours</Type>
            </Value>
          </Values>
        </Comparitor>
        <Comparitor xsi:type="Comparison">
          <Operation>GreaterThan</Operation>
          <Values>
            <Value>
              <Description>Appointment.Count</Description>
              <Type>Parameter</Type>
            </Value>
            <Value>
              <Description>0</Description>
              <Type>Number</Type>
            </Value>
          </Values>
        </Comparitor>
      </Conditions>
```

```
            <Type>Number</Type>
          </Value>
        </Values>
      </Comparitor>
      <Comparitor xsi:type="Comparison">
        <Operation>GreaterThan</Operation>
        <Values>
          <Value>
            <Description>DWP.Count</Description>
            <Type>Parameter</Type>
          </Value>
          <Value>
            <Description>0</Description>
            <Type>Number</Type>
          </Value>
        </Values>
      </Comparitor>
    </Conditions>
  </Rules>
</Period>
<Period xsi:type="RP">
  <Description>
    <Name>WRP</Name>
    <Description>Weekly Rest Period</Description>
    <PrePosts>
      <PrePost>
        <Pre>WWP</Pre>
        <Post>WWP</Post>
      </PrePost>
    </PrePosts>
    <Contains />
  </Description>
  <Rules xsi:type="Comparison">
    <Operation>GreaterThanOrEqualTo</Operation>
    <Values>
      <Value>
        <Description>RestLength</Description>
        <Type>Parameter</Type>
      </Value>
      <Value>
        <Description>30</Description>
        <Type>Hours</Type>
      </Value>
    </Values>
```

```
  </Rules>
</Period>
<Period xsi:type="RP">
  <Description>
    <Name>DRP</Name>
    <Description>Daily Rest Period</Description>
    <PrePosts>
      <PrePost>
        <Pre>DWP</Pre>
        <Post>DWP</Post>
      </PrePost>
    </PrePosts>
    <Contains />
  </Description>
  <Rules xsi:type="AndCondition">
    <Conditions>
      <Comparitor xsi:type="Comparison">
        <Operation>GreaterThanOrEqualTo</Operation>
        <Values>
          <Value>
            <Description>RestLength</Description>
            <Type>Parameter</Type>
          </Value>
          <Value>
            <Description>8</Description>
            <Type>Hours</Type>
          </Value>
        </Values>
      </Comparitor>
      <Comparitor xsi:type="Comparison">
        <Operation>LessThan</Operation>
        <Values>
          <Value>
            <Description>RestLength</Description>
            <Type>Parameter</Type>
          </Value>
          <Value>
            <Description>30</Description>
            <Type>Hours</Type>
          </Value>
        </Values>
      </Comparitor>
    </Conditions>
  </Rules>
```

```
  </Period>
</ArrayOfPeriod>
```

Figure 6-14 – The XML rule representation of the daily and weekly work and rest periods after serialisation

6.3 Summary

This chapter of the thesis has outlined the proposed testing programme to indicate the purpose and method for the initial operational testing process.

In addition, a test bench has been designed that is capable of supporting the testing programme, to assist in the automated generation of statistical results.

As outline in Section 6.1, it was important to show, with the operational tests, that the given solution can:

a. be integrated into a system requiring rule testing
b. read a set of rules from the external system
c. compile the rules into an executable form
d. test a set of data against the compiled rules

Following the completion of the test plans execution, the proposed solution has been shown to meet these operational tests. The solution was successfully integrated into an external system, provided with a set of rules by the external system, compiled the rules into executable code, and tested data against the compiled rules, returning the correct Boolean response.

The next step is to undertake the outlined test plan; the results of which are presented and analysed in the next chapter.

7 INITIAL RESULTS

The testing programme has been set out in the previous chapter of this thesis and the test bench application has been designed and implemented that can be used to carry out the testing of the defined rules and data in order to prove that the solution works and to assess its performance.

There are a variety of result types that can be obtained from research such as this. These range greatly from analysis of the computational performance increases gained from using the system, as opposed to the manual alternatives, through to the measurable organisational value of such a system to a company using this approach as a part of their scheduling process.

The focus of this Chapter is to determine whether the proposed solution can:

a. be integrated into a system requiring rule testing (see Section 3.1)
b. read a set of rules from the external system (see Section 3.2)
c. compile the rules into an executable form (see Section 3.3)
d. test a set of data against the compiled rules

The outcome of this Chapter looks at the various results gained in relation to the above and considers how these results compare to alternative approaches and how they could be improved in the future.

Once the proposed solution is shown to work and meet the initial requirements (set out in Chapter 3), Chapter 8 takes the test of the proposed solution further by seeing how it operates under various condition, for example alternative software frameworks and operating systems, to determine the effects of this change on the overall system performance.

7.1 Test Results

Analysis of the system's performance is one of the methods that can be used to rate the proposed solution. This method used the test case defined in the Testing chapter to generate a set of data that can then be analysed and from which some conclusions can be drawn.

In order to obtain the statistical results for comparison, the test bench uses a stopwatch style approach to capturing data. Before execution of a test the current system time is captured (using the operating systems high resolution timer), then again after execution. The length of time taken to execute is then determined as the difference between the two. This will provide a very accurate look at the computational performance of the solution without interfering with the execution process itself.

In addition the applications current level of memory usage is captured in order to see the changes in memory usage between the start of execution and the end. The main reason for taking this approach and not monitoring throughout execution is that it was important that the computational performance measurements were not interfered with during execution by taking processing time to capture the memory usage.

For clarity when reading the diagrams, as previously illustrated within Figure 6-5, Figure 7-1 shows the specifications of the various systems used during the testing process.

Identifier	Processor Type	Processor Speed	RAM Size	Operating System
A	Intel Pentium 4	3 GHz	1 GB	XP Service Pack 2
B	Intel Pentium 4	3 GHz	2 GB	XP Service Pack 2
C	Intel Pentium 4	2.4 GHz	1 GB	XP Service Pack 2
D	Intel Core Duo (Centrino Duo)	2.33 GHz	4 GB	Vista x86

Figure 7-1 - Computer Specifications Used In Testing

7.1.1 Test Results – Computational Performance

After carrying out the tests on the varying dataset sizes against the rules some interesting results were found. Figure 7-2 and Appendix 11.6.1 provide charts representing the results gained from carrying out the tests on different dataset sizes against the rules. The following conclusions can be made:

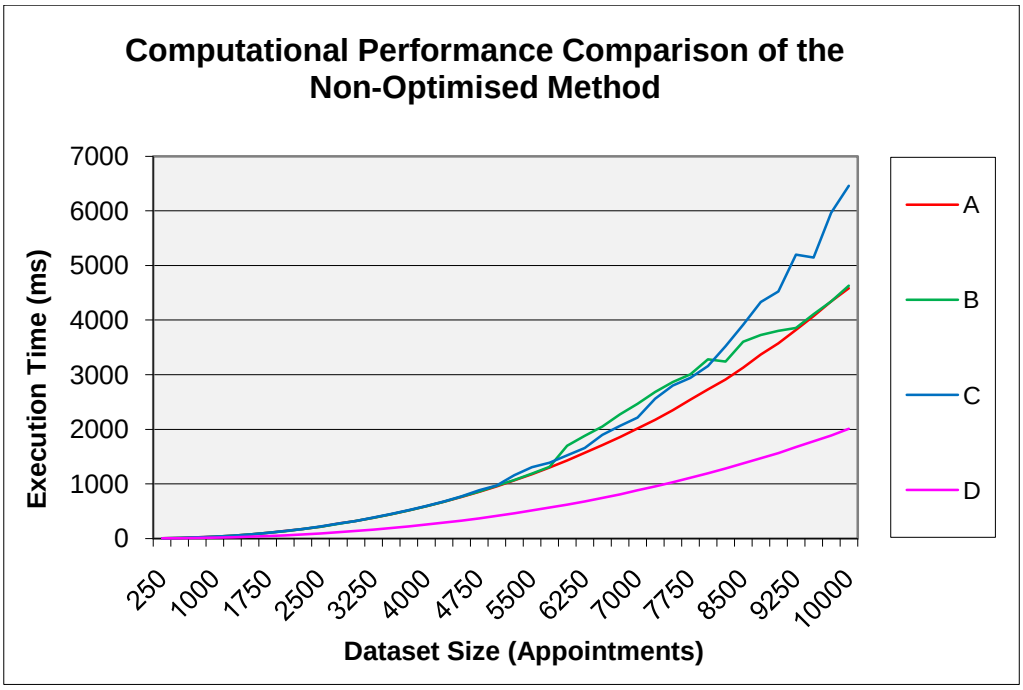


Figure 7-2 - Chart illustrating the computation performance comparison between different system types

The results are non-linear. If the results were linear this would imply that the length of time taken to carry out the rule tests on the data is directly proportional to the increase in dataset size. After an investigation into the reasons for the results found, it seems that the bottle neck in the testing process is not as a result of slow execution of the compiled rules in testing the data but it indicates that there are issues with the work plan generation algorithm in constructing the work plan in an incremental way.

Multi-core processors provide improved performance. The system with the slowest processor speed has outperformed the other system types in

computational performance execution which provides a good indication that its multi-core feature is providing a considerable performance improvement to the overall execution time.

In the article (Geer, 2005) Geer talks about some of the reasons for turning to multi-core processors and states “multi-core chips don’t necessarily run as fast as the highest performing single-core models, but they improve overall performance by handling more work in parallel”. In the case of the proposed rule testing solution, the solution itself has not been designed as a parallel activity which could potentially improve the testing performance (discussed under Section 9.5 as future work). There are, however, other tasks being carried out in the background by the operating system, or even within the .NET framework within other threads, that will be able to operate on the other core, leaving the rule testing algorithm with the full use of its own dedicated core, subject to the resource scheduling process of the operating system.

Larger quantities of RAM provide better computational performance. Two of the systems have the same processor type and operating system however one has twice the RAM available compared to the other. The rule testing system isn’t using an excessive amount of RAM (see Section 7.1.2) however it is clear from the test results provided in the chart that the performance of computer B is better than that of computer A when the dataset size reaches around 6000 appointments.

The reasons for the differences relate to the garbage collection scheme used by the .NET framework. The .NET framework’s garbage collector uses an optimisation engine to determine the best time to perform a collection based on a variety of factors. This means that if a process is performing computationally expensive tasks, the garbage collector will not attempt to perform collection at that point unless the amount of available memory is limited and thus there is a need for the collection. With a system with plenty of available RAM, the rule testing engine can execute for longer, with more uncollected, allocated, memory associated than a system with limited available memory.

This does then have an overall effect of the computational tests undertaken where we may expect to see a difference in computational performance for equivalent systems where one has more memory than the other.

7.1.2 Test Results – Memory Performance

In addition to measuring the computational performance of the rule testing process, measurements were taken to determine how memory usage changed depending upon system architecture. Figure 7-3 and Appendix 11.6.2 provide charts representing the memory usage of the rule testing application as measured before undertaking each test, then after.

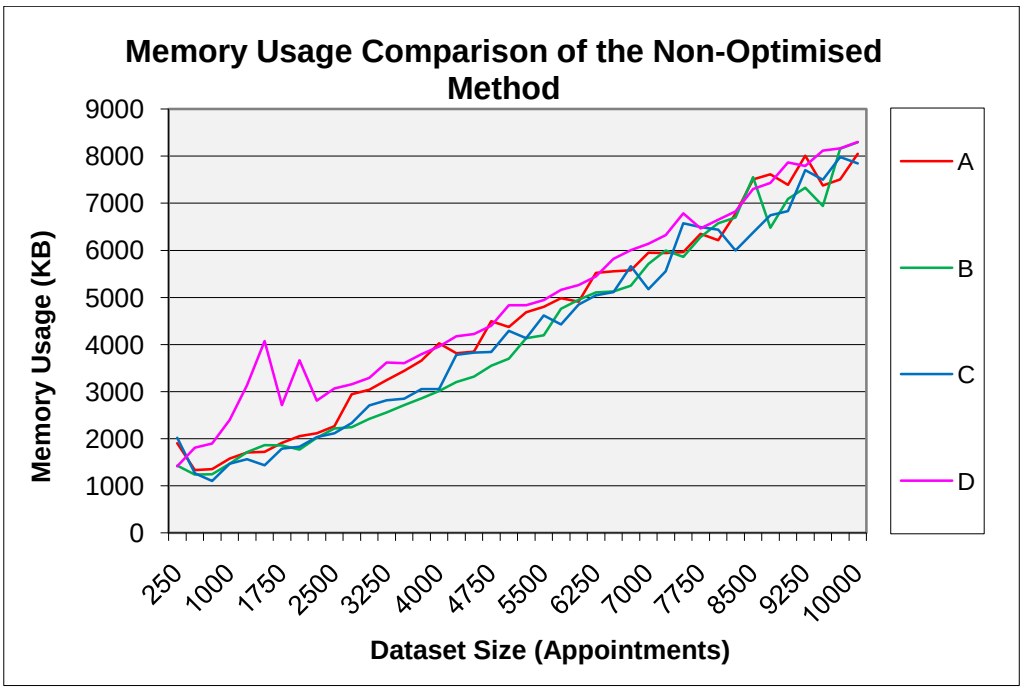


Figure 7-3 - Chart illustrating the memory usage comparison between different system types

Whilst no significant conclusions can be drawn from these results, it can be seen that no matter which system type is considered, the memory usage is roughly the same with the increase in memory over time relating to the increase in the dataset size and resultant work pattern produced following the rule testing process. The .NET Framework's garbage collector is responsible

for managing the applications memory, ensuring that any unused memory is freed appropriately and no memory leaks are occurring.

7.1.3 Additional Improvement to Computational Performance

As a result of undertaking the tests above, it was clear that there were some performance issues relating to the work pattern creation algorithm element of the rule testing process. This was a fairly unexpected result. The original aim of the research was to improve the overall rule testing process for scheduling systems requiring rule testing; the work pattern creation algorithm wasn't originally perceived as one of the more important elements of the rule testing process, however, the results indicated this was an area that could benefit from some additional work.

This is not to say that the approach of compiling rules into executable code is wrong, however, now that this element of the system has been optimised the bottleneck within the system has moved. There are a number of areas that could be considered in order to improve the computational performance of the work plan creation process, as outline in Figure 5-16 and Figure 5-17.

7.1.3.1 Incremental Work Plan Construction

When constructing the work plan, rather than having to recreate the entire work plan when testing additional elements of data, a performance increase could be gained from caching the already constructed work plan and only adding the additional elements of data to the plan during each new test phase.

With some minor modifications to the `Tester` class (previously discussed under Section 5.3.2), a cached version of the work pattern can be taken as an additional output to the testing method and passed back in to the `Test` method during the next iteration. After undertaking this relatively simple modification

and running the testing procedure again, a much better computational performance was found and can be seen in Figure 7-4 and Appendix 11.6.3.

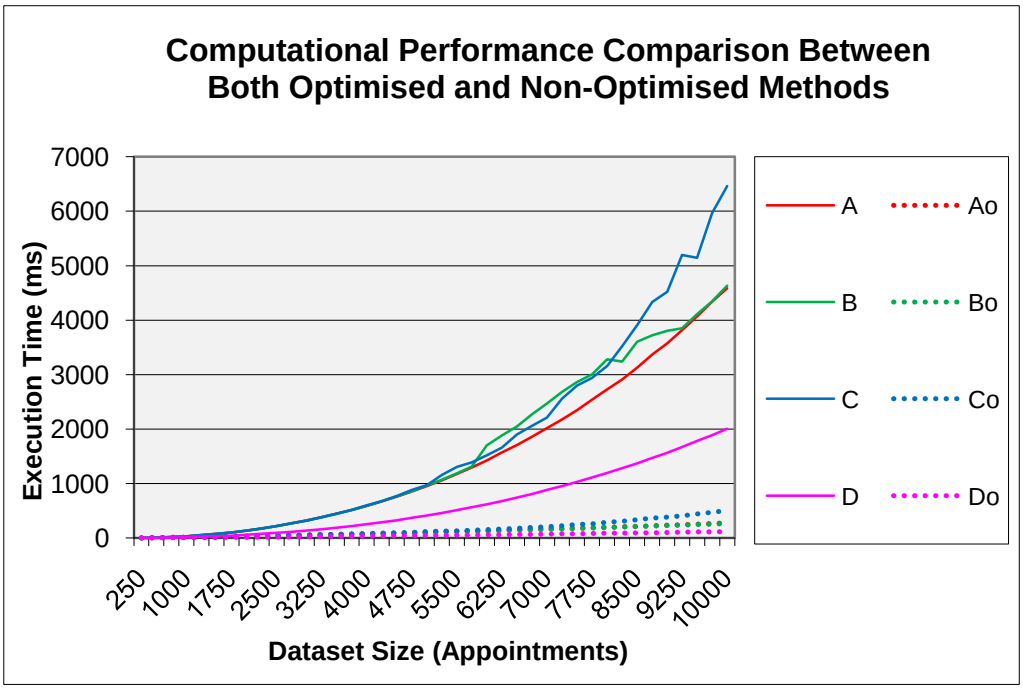


Figure 7-4 - Chart illustrating the computational performance comparison between the original and the incremental testing approaches

(Legend example: A= non-optimised, Ao = optimised)

The incremental approach reduces the execution time of the rule testing process considerably and identifies an important fact, that during a scheduling process the work plan for each of the drivers (in the case of the transport scheduling system) should ideally be cached and reused when attempting to add additional work to a work pattern. There are limitations to this, however, in that only elements of data that occur at the end of the current work pattern can be added using the optimised method as opposed to elements of work that occur at the start or in the middle of the work pattern. For those that cannot be added incrementally, the simplest method would be to reconstruct the work pattern from scratch.

In terms of memory usage, the charts provided in Figure 7-5 and Appendix 11.6.4 show the memory usage between the different approaches barely changes.

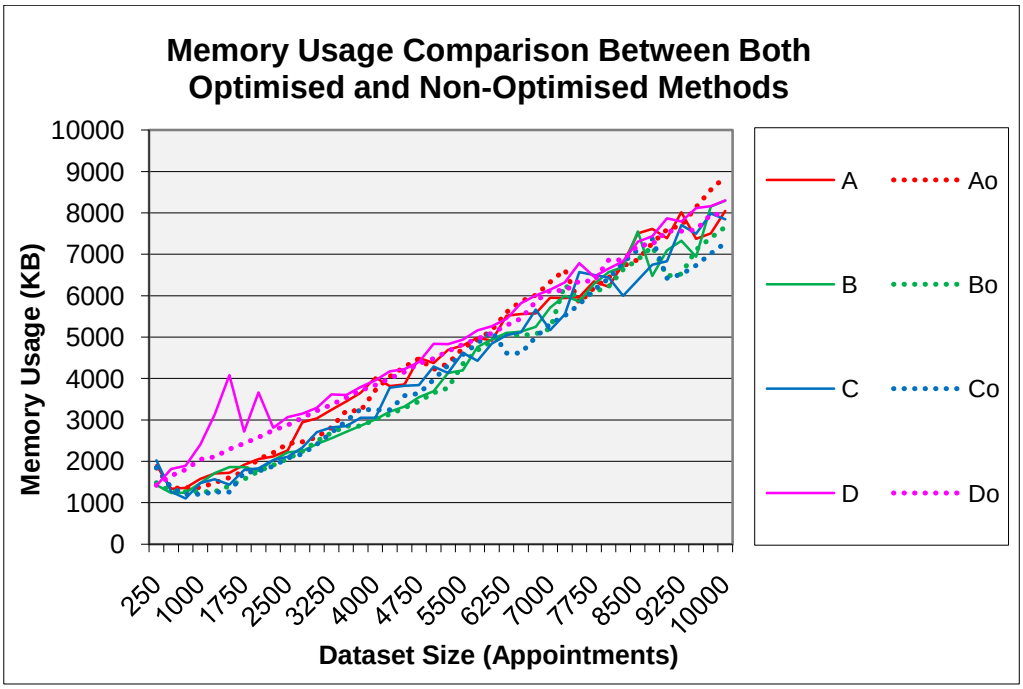


Figure 7-5 - Chart illustrating the memory usage comparison between the original and the incremental testing approaches

(Legend example: A= non-optimised, Ao = optimised)

The approach taken to monitor memory usage without hindering the computational performance of the rule testing process required the capturing of memory usage before and after the rule testing process is carried out. Because the end result of the rule testing process is the same, a complete work plan, the amount of memory used at each point will have been the same.

However, within the non-optimised approach there will have been a significant amount of additional load placed on memory management due to the fact that the work pattern array is being frequently created and destroyed requiring more memory allocations and de-allocations per rule testing cycle, with the same eventual outcome and resultant level of memory usage. This will, in turn, have an impact on the computational performance measurement and will not show up with the memory usage measurements taken.

7.2 Alternate Metrics

For this research problem, the computational performance results aren't necessarily the most appropriate means of measuring the outcomes of this work. The problem domain from which this research has been derived is based on a real world business problem and, as such, it is also important to consider the Measurable Organisational Value (MOV) of the original problem and how the software solution compares using this method of measurement.

7.2.1 Measurable Organisational Value

Existing work, such as (Wetherall, 2002), identifies the need for this research from a business problem point of view. For that organisation specifically, a relatively high paid domain expert had to be employed full-time at each of their branches in order to manually undertake the scheduling process taking into account the legal driving rules. This time consuming task for the scheduler is the manual interpretation and testing of the legal driving rules which is at the core of the scheduling process.

It can therefore be concluded that the financial benefits of having the developed software solution in place within an organisation is significant to the overall operating costs of the scheduling process. When considering that a domain expert, the alternative to an automated system, may be paid a minimum of £30,000 GBP per year, the financial saving of replacing the work of the domain expert based on a non-increasing salary alone is potentially £300,000 GBP over a 10 year period. Not to mention the other employer costs such as human resourcing, national insurance and tax contributions, pension schemes and standard pay increases which contribute toward an even greater overall saving.

Per organisation is one thing, however, when considering that this solution would benefit not just every transport scheduling company who have to

schedule based on legal driving rules, but any other organisation also scheduling with a set of rules, such as timetable scheduling systems, the proposed solution can have a significant impact on the future of rule testing in scheduling.

An alternative overhead employed by a number of organisations that use automated scheduling systems is the use of expert scheduling algorithm developers who use, for example, the constraint based scheduling algorithm approach within their scheduling systems. When the rules within the company's domain change, the expert has to be re-employed to develop a new scheduling algorithm which takes into account a new set of rules at considerable cost to the company. This overhead could be eliminated if the proposed solution was employed instead of an algorithmic rule scheduling process.

7.2.2 Compiled Execution

In addition to the other tests carried out, an important aspect that needs consideration is the claim that compiling the rules results in computationally faster execution than other approaches. A number of experiments to prove this claim have been investigated however the results of these tests have resulted in numerous pit falls.

The main issue faced, when attempting to investigate this, is that there are no suitable rule interpreters for the execution of either IL (Intermediate Language) or C#. Microsoft's implementation, within the .NET Framework, only includes a JIT (Just-In-Time) compiler for runtime execution of an applications IL. This approach is focused on heavily within the standard document of the CLI (ISO/IEC23271, 2006). As a result alternate approaches to creating computational performance comparisons have been investigated.

The first experiment attempted to investigate the concept using the open source implementation of the CLI (Common Language Infrastructure) and

CLR (Common Language Runtime) provided by the Mono project (Dumbill, 2004). In early versions of the Mono project, they used an interpreter called Mint as a first step to porting Mono onto alternate platforms. With current versions of Mono, including the source code distributions, the Mint application has not been maintained and every attempt to compile Mint, including using historic source distributions, has failed.

Additionally, the use of DotGNU(Bollow), another open source implementation has been investigated. DotGNU can be used in two ways, the first with a JIT compiler and the second without, i.e., interpreted. The main issue found when attempting this approach is that the open source project is relatively immature and isn't keeping up with the development of technology in the same way the Mono project is managing to. The approach taken with DotGNU is to develop the project to meet specifically with the standard specification (ISO/IEC23271, 2006) as opposed to providing compatibility with the Microsoft implementation of the .NET Framework. The main problem found in using this approach were incomplete or incompatible implementations of the `System.Reflection` namespace. It was not, therefore, possible to follow this line of investigation further.

The final approach attempted to use a C# interpreter known as PaxScript(Baranovsky) to interpret a C# representation of the rules. The problem found when attempting this approach is that the compiled rule approach proposed depends heavily on the `RuleSupport` functionality provided within the proposed rule testing engine and there are limitations on PaxScripts' ability to interpret class inheritance with classes in existing class libraries.

After substantial investigation, no other appropriate methods of proving the claim that a compiled rule approach is computation faster than a non-compiled approach could be found therefore it is determined that the use of existing work will be required to prove this claim. A number of references have been made in the literature review to existing research papers that support the claim

that compiled code executes faster than interpreted code, as seen in Section 2.6.

7.3 Summary

This chapter of the thesis has presented the test results that were generated as a result of running the test bench in conjunction with the rules and data that were identified in the Testing chapter of this thesis.

The results have been analysed and some conclusions have been drawn on the results. An optimised method of incrementally generating work patterns has been identified which has reduced the execution time of the system as well as reducing the needed continuous memory allocations and de-allocations in the non-optimised method.

The next chapter will present some additional results obtained through further testing work that has been done to investigate the effectiveness of the proposed solution on different software platforms and operating systems, as well as discussing some of the problems found in trying to derive the evidence.

8 FURTHER RESULTS

The previous chapter of this thesis reported the results of the testing process undertaken to show evidence of the contributions proposed in Section 1.4. This chapter of the thesis expands on this work by further illustrating how the proposed solution can be used within alternate application domains and within various environments. This chapter will also look at problems that are faced in designing a set of rules to work with such a complex system and how additional support can be provided to help diagnose problems with rule definitions.

In addition this chapter draws a comparison between the performance execution time between the previous test results given in Section 7.1 and those result obtained from the testing of a new set of rules within an alternate application domain. Finally this chapter will highlight some of the inconsistencies between the two sets of test results and provide justifications for these inconsistencies.

8.1 Additional Test Scenario

The additional testing scenario is derived from the problem of lecturer timetable scheduling. The rules in this scenario differ between Countries, Unions and Institutions so a subset of localised rules has been defined for the purposes of testing.

The rules for testing within this domain are divided into three categories, Shift, Daily Work Period (DWP) and Weekly Work Period (WWP).

A Shift is classed as a period of work, such as a morning, an afternoon or an evening. A Shift cannot be longer than 4 hours long without a break between it and the next shift. A Shift cannot start before 9am and cannot go on passed

9pm. A Shift must be separated by a break of at least 1 hour, called a Shift Break (SB), and there cannot be more that 2 Shifts within a DWP.

Each DWP cannot be longer than 9 hours and must be separated by at least 15 hours of rest, called a Daily Rest Period (DRP). A WWP cannot be more than 5 DWPs long and must be separated by a rest of at least 2 days, called a Weekly Rest Period (WRP), giving the worker a weekend length break.

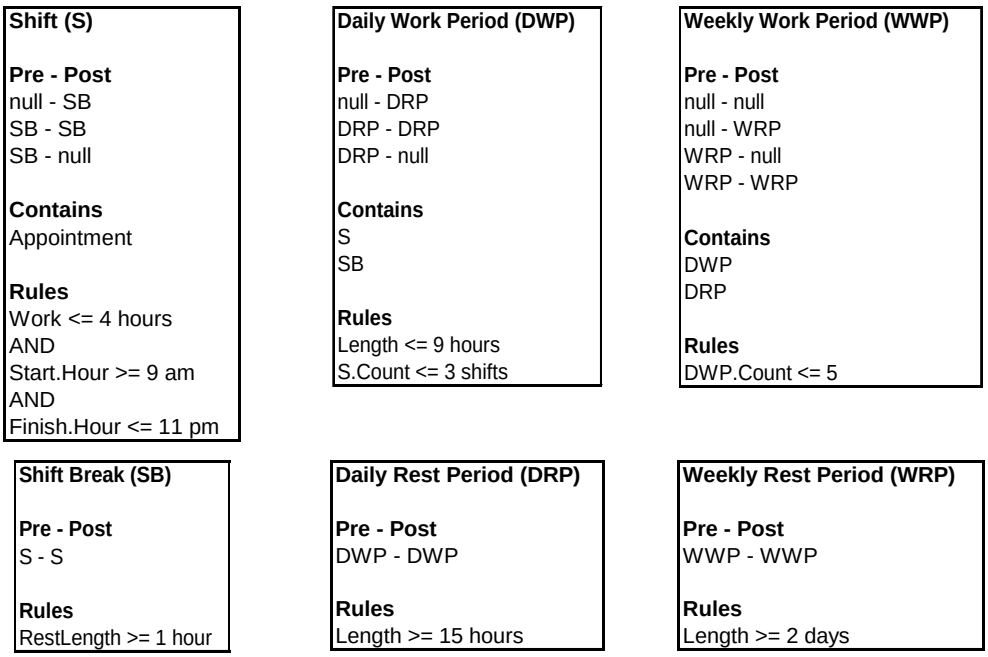


Figure 8-1 - Illustration showing the six testing rules together with their supporting metadata for the additional test scenario

8.2 Rule Description Problems

In a similar way to the previous testing scenario of driving rules, the teaching rules were written into their XML representation for use with the rule testing engine. These can be seen in Section 11.7.2.

Problems were found when executing these rules with unexpected outcomes shown by the rule engine. The problem comes from the fact that whilst

creating the rule descriptions using XML has simplified the process of describing rules, trying to remove "bugs" within the rule description is actually quite complex and difficult to achieve.

In order to overcome this, further improvements were required from the rule compiler and execution engine to support the ability to debug the rules themselves whilst the engine is executing.

8.2.1 Rule Debugging

In traditional software development, the debugging of an algorithm is often carried out within an IDE (Integrated Development Environment), such as Microsoft Visual Studio 2008 (as used for the construction of the rule compiler for this project). Normally, breakpoints can be added to a particular line within the source code of a program and the IDE will halt execution at that line and allow the developer to step through the code and look at the values of variables in order to diagnose problems.

In some ways the XML representation of the rules can be classed as source code for the rule compiler, however, there is not always an XML file as rules can be transferred to the system through an alternate medium such as a direct download via the internet from server to client. Additionally, the XML representation is merely a serialised form of the software objects representing the rules within the software application, differing from traditional source code significantly.

In order to provide an equivalent of source code for the rules for use during the debugging process, it is necessary to output an XML file during rule compilation. In addition to the production of a source file, variables that are used internally by the rule engine will require variable names to support the debugging process.

This extra support has been added to the rule compiler with the addition of a new class to support the addition of the debug information to the compiled rules. This class can be seen in Section 11.3.1.2 . It is responsible for the creation of a file called Rules.xml and integrating the compilation process of the rules to the source code output into the new Rules.xml file.

Once the ability to debug is added to the system, breakpoints can be added into the IDE at particular lines of that source or within the RuleTesting.Tester class. An appropriate place for this would be within the DoTest method of the Tester class, prior to the call to the compiled rules Test method.

Once the rule engine has halted at the line of code prior to calling the Test method on the rules themselves (as seen in Figure 8-2), it is possible, using the debugging features built in to Visual Studio 2008, to step into the newly generated rule file and begin debugging the rules themselves.

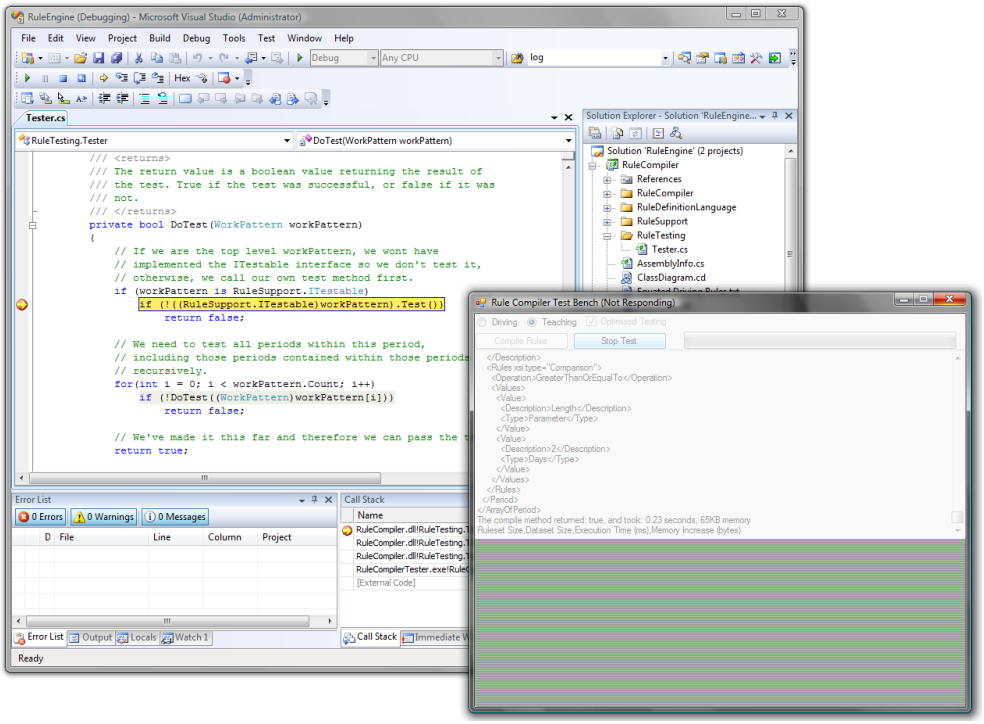


Figure 8-2 – Screen shot illustrating the break in execution of the rule engine prior to the test method

Figure 8-3 shows the result of stepping into the Test method. This jumps from the C# source code for the rule engine's Tester class into the Rule.xml file produced by the rule compiler. It is then possible, in the same way as debugging C# code, to debug the rules, by stepping through the code and analysing the changes to variables within the rules themselves.

As well as being able to debug the rules, when reaching a point in the rules such as where the value of a parameter is being obtained, the developer can step into this line which will bring the developer back to the C# source code to obtain parameter values so this can be debugged also.

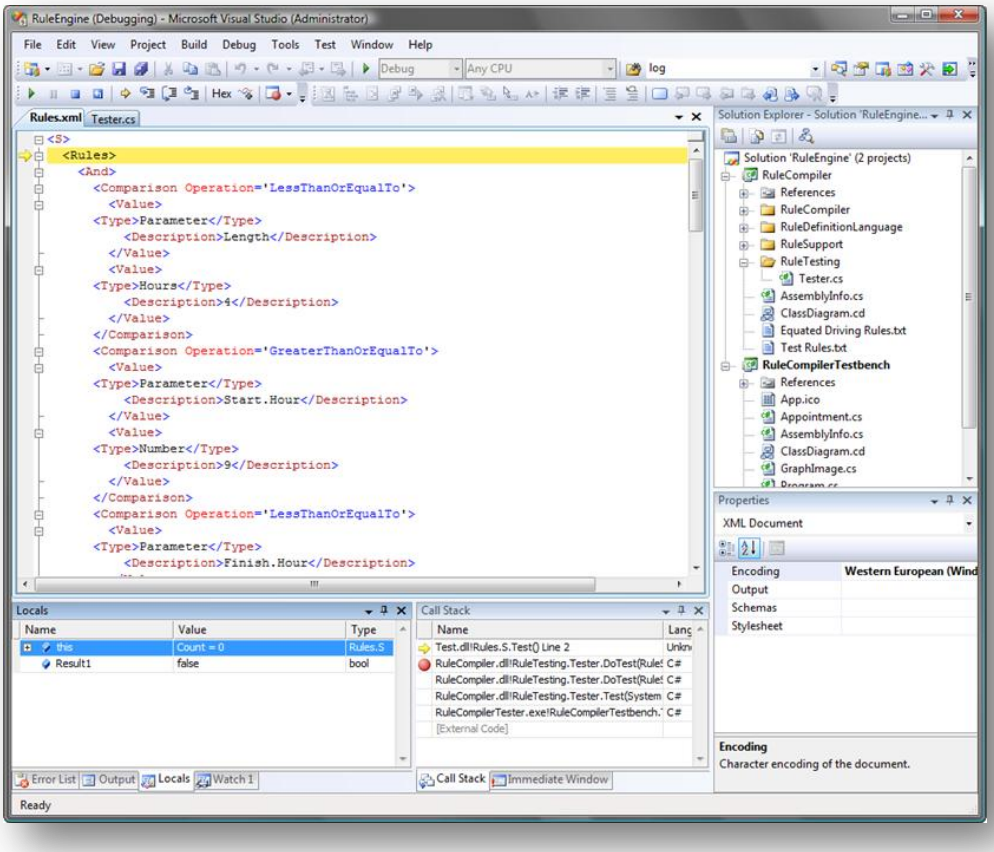


Figure 8-3 - Screen shot illustrating the "step-through" debugging of the XML rules

Figure 8-4 shows the effect of stepping into the rules in order to obtain a property's value, such as the Length of a work or rest period. There are a few interesting things that can be seen from this figure; firstly the Locals window

in the bottom left shows the variables used by both the compiled Rules and the rule engine; secondly the Call Stack window, at the bottom in the middle, shows the current call stack and this shows the previous calling method to the GetProperty method is the compiled rule, of unknown language.

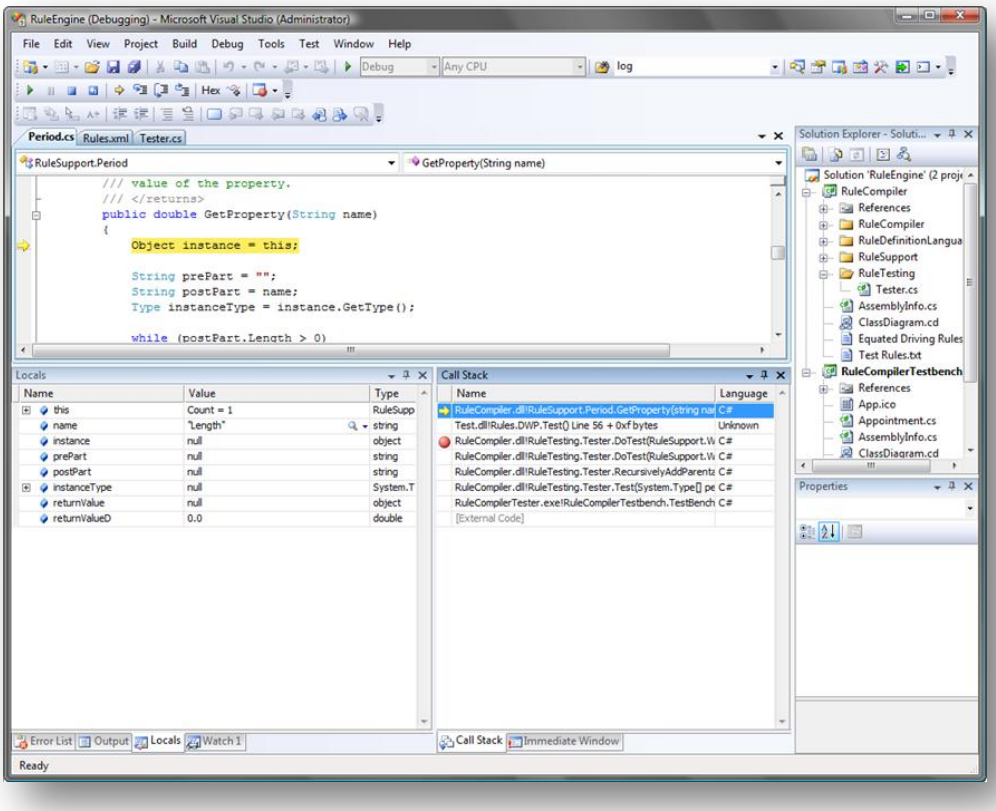


Figure 8-4 - Screen shot illustrating the Locals and Call Stack windows containing the debugged rules variables

With the addition of the debugging support, the process of removing faults with the rules becomes as simple as developing the rules themselves and results in an all round better solution for rule development.

One of the drawbacks to this approach is that when including debugging information into the compiled rules, the resulting dynamic link library (DLL) becomes bloated and the performance execution time is affected. One of the ways to prevent this affecting the rule engine was to specify that debugging

information is only included when the rule engine itself is compiled to include debug information. This means that two version of the engine can be used, one for diagnosing problems with described rules and the other for regular rule execution.

8.2.2 Comparison of Debuggable and Normal Rule Compilation

Tests have been undertaken to produce results that allow for the comparison of the computational performance and memory usage of the rule engine using both regular compiled rules and those with debugging information included. With these tests it is possible to see why it is important to use different versions for both testing and live implementations of the rule engine.

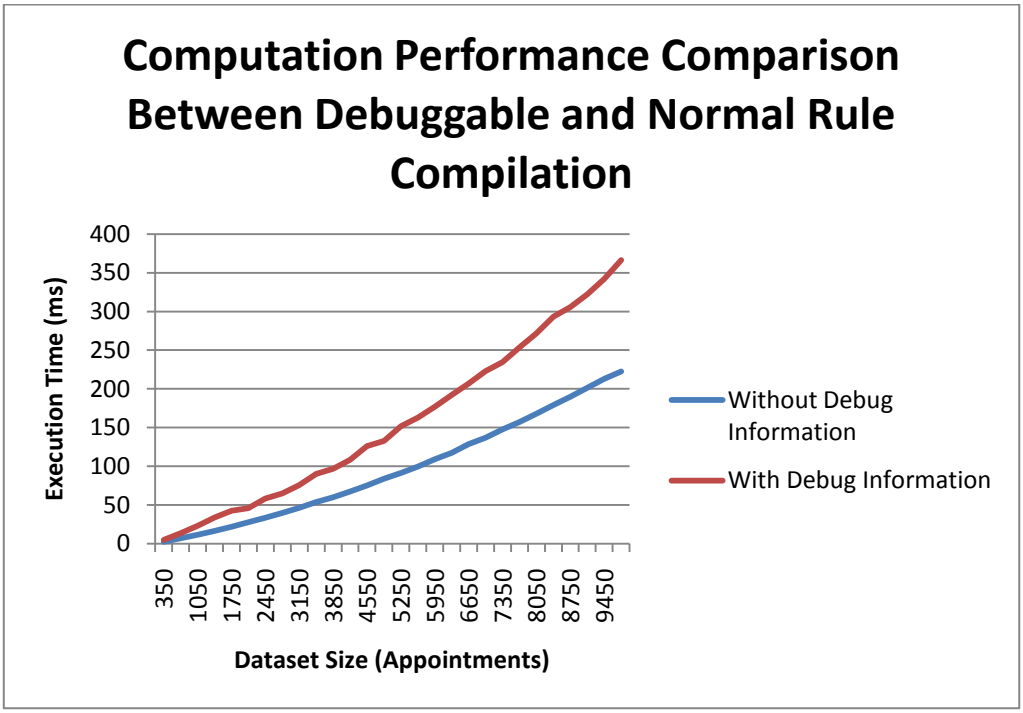


Figure 8-5 - Chart illustrating the computational performance comparison between rules compiled with and without debugging information

Figure 8-5 shows the results of the tests undertaken to show the difference between the computational performance of the rule testing process with debugging information included into the compiled rules and without. It is clear

to see from the results that the performance improvement obtained when not including the debug information is substantial, especially when testing large dataset sizes. This is an important find as it will help to identify the need for two versions of the runtime, one for the rule developer, to help with diagnosing faults in the rule descriptions, and the other for use in execution of the rule engine within a live rule testing application.

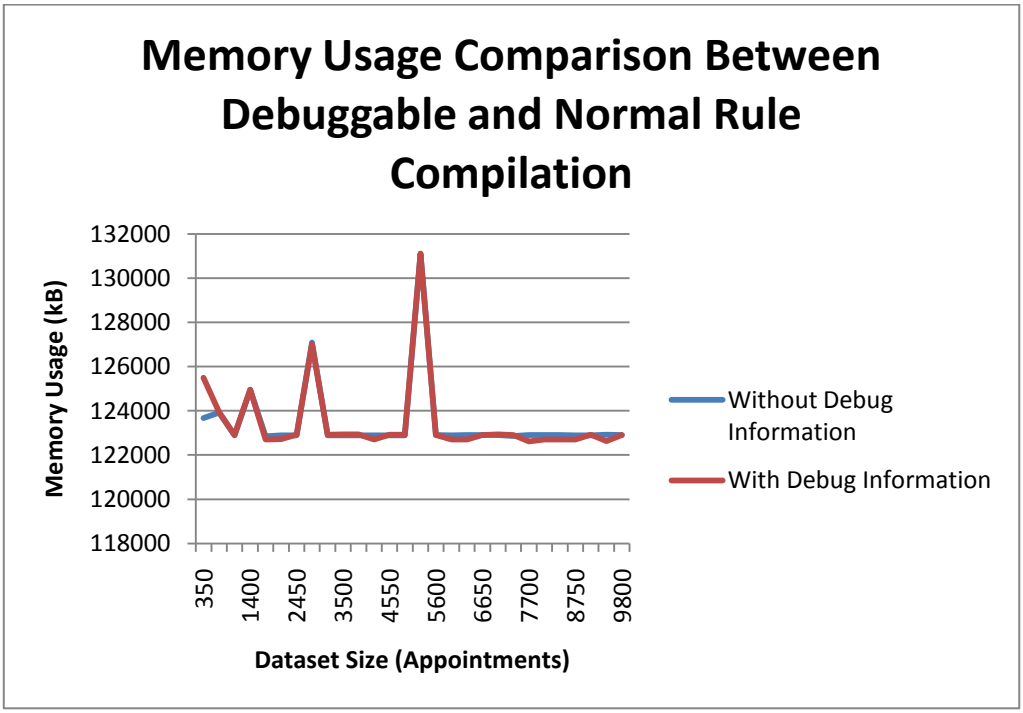


Figure 8-6 - Chart illustrating the memory usage comparison between rules compiled with and without debugging information

Interestingly, the memory usage comparison, show in Figure 8-6, shows that the quantity of memory required when executing the two approaches is nearly identical, with only mild differences that can be attributed to fluctuations in the garbage collectors cleanup routine. A strange similarity occurs when the dataset size reaches around 5000 elements, when a spike occurs in the memory consumption, simultaneously for both approaches. After some investigation into the cause of this sudden memory increase, it was found not to occur within the rule engine and therefore it is highly likely that, again, the common language runtimes garbage collector found it unsuitable to collect memory at that time.

8.3 Additional Testing Process

Once the new teaching rules were defined correctly, it was possible to undertake a testing process on those rules on a range of hardware architectures, in order to gain a comparison of the performance execution between the driving rules and the teaching rules.

Only the optimised testing process, without debugging information, was used to obtain results for comparison as it has already been proven that this approach produces a greater performance. Also, as modifications were made to the rule engine, the optimised driving rules were retested in conjunction with the new teaching rules to provide fair comparison between the two.

8.3.1 Test Plan

The teaching test data has been designed to test the boundaries of the teaching rules, described in Section 8.1.

Figure 8-7 shows a graphical representation of a week of teaching that will adhere to the new teaching rules and can be used to generate reoccurring sets of data spaced a week apart. The generation of larger datasets works in a similar way to the previous testing approach, described in detail in Section 6.1.1.2.

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
09:00 - 10:00	1	5			13		
10:00 - 11:00							
11:00 - 12:00	2	6			14		
12:00 - 13:00							
13:00 - 14:00							
14:00 - 15:00	3		7	11			
15:00 - 16:00							
16:00 - 17:00	4		8	12			
17:00 - 18:00							
18:00 - 19:00							
19:00 - 20:00			9				
20:00 - 21:00							
21:00 - 22:00			10				
22:00 - 23:00							

Figure 8-7 – A single week of the sample teaching dataset represented graphically

The test data for the driving rules are unchanged from those illustrated with Figure 6-1 and Figure 6-2. However, rather than generating datasets incrementing with 250 elements of data at a time, this test plan will use dataset sizes of 350 elements of data in order to work with multiples that can also support the teaching test data. This is because there are 14 elements of data to the teaching dataset and 10 elements of data to the driving set, and 350 is a multiple of both 14 and 10.

In order to support the testing of the new rules and data, additional functionality was added to the original test bench, described in Section 6.2, to include the ability to specify which type of rules to test for, driving or teaching. The top left of Figure 8-8 shows the ability to choose between Driving and Teaching rules within the testing process.

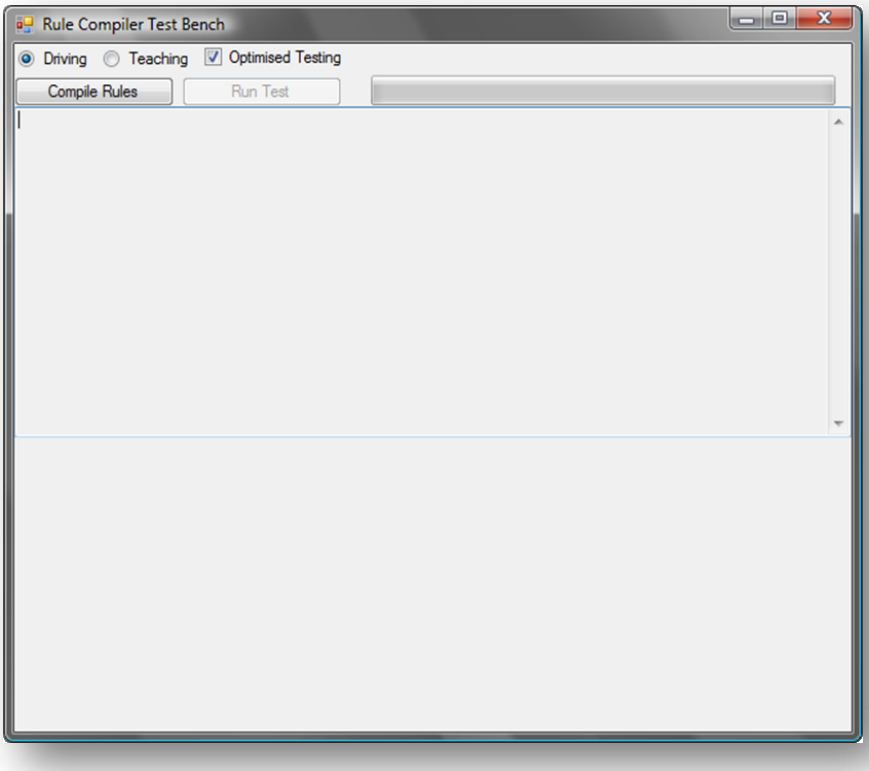


Figure 8-8 - Screen shot illustrating the modification to the test bench to include the testing of Teaching rules

8.4 Comparison of Driving Rules and Teaching Rules

The next step in the additional testing process was to compare the computation performance between the existing driving rules and the new teaching rules and their respective datasets.

With the inclusion of the Shift breaks, introduced within the teaching rules, that ruleset had 6 types of rule in comparison to the driving ruleset which had 4 types of rule. It was expected that this additional complexity would contribute toward an increase in computation in order that the additionally rules can be tested against their dataset.

Figure 8-9 shows the computational performance results of testing both the Driving and Teaching rules. The results are not quite as expected. Initially, the

expectation was that the additional 2 rules into the ruleset would cause a natural increase in computation time, relative to the number of rules; so that if it took the driving rules 16 seconds to test a set of data, the same quantity of data on the teaching rules would take 24 seconds.

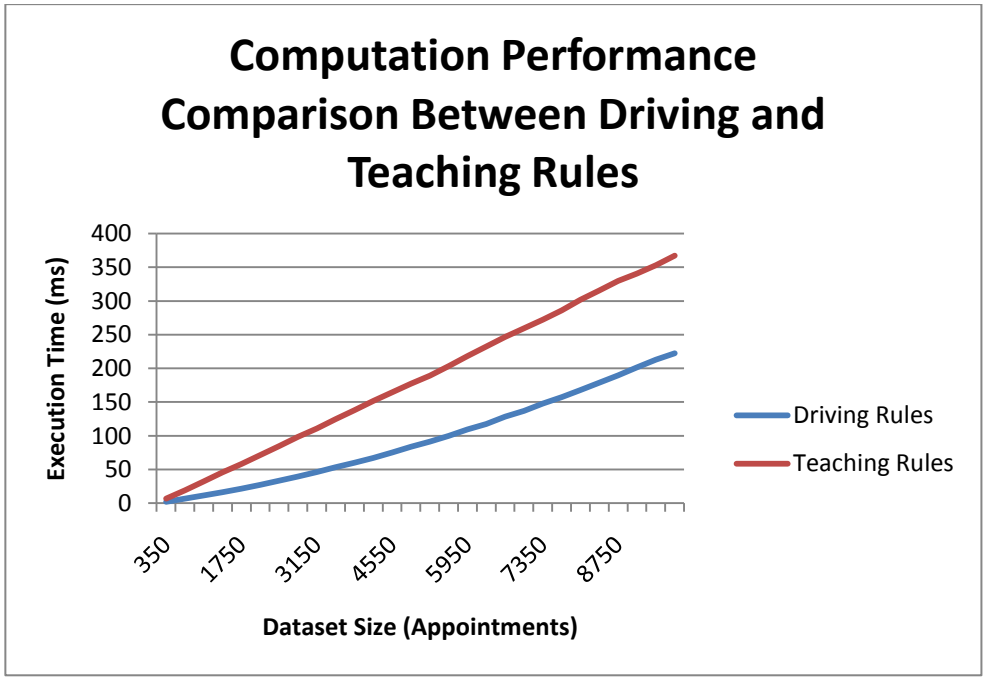


Figure 8-9 - Chart illustrating the computation performance comparison between driving and teaching rules

It is clear, from Figure 8-10 that this is not the case and that there isn't a direct correlation between ruleset size and computation performance. After some investigation, it was found that that the complexity of a given rule plays a significant role in the computation performance of the rulset. Also, the inter-relationship between rules influences computational performance. For example, the teaching ruleset contains a Shift which is contained within a Daily Work Period which is contained within a Weekly Work Period; this differs significantly as this ruleset has 3 levels of rules whereas the driving ruleset has only 2 levels of rules, reducing the work plan constructions workload substantially.

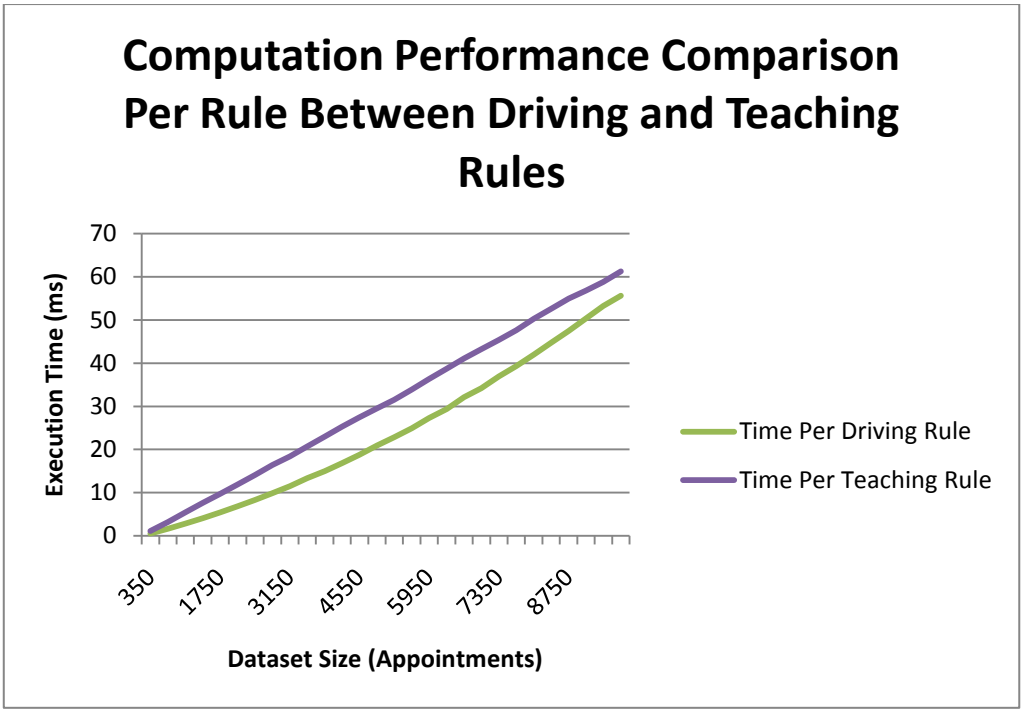


Figure 8-10 - Chart illustrating the per rule computation performance comparison between driving and teaching rules

Another interesting factor that influences the computational performance of the rule engine, also found in trying to understand the difference in performance between driving and teaching rules, is the use of various properties within the rule itself. For example, if the rule requires the testing of rest periods whose length must be calculated dynamically based on the size of the surrounding work periods, the additional time required for this calculation, when dealing with many tests each second, has a significant impact in reducing the rule engine's performance.

The results of the memory usage comparison between driving and teaching rules, given in Figure 8-11, are as expected from the various differences between the two rulesets as described above. No doubt, the addition of another layer in the rule hierarchy will have had an impact in the work plan produced by the rule engine and explains the increase in the amount of memory required to store that work plan.

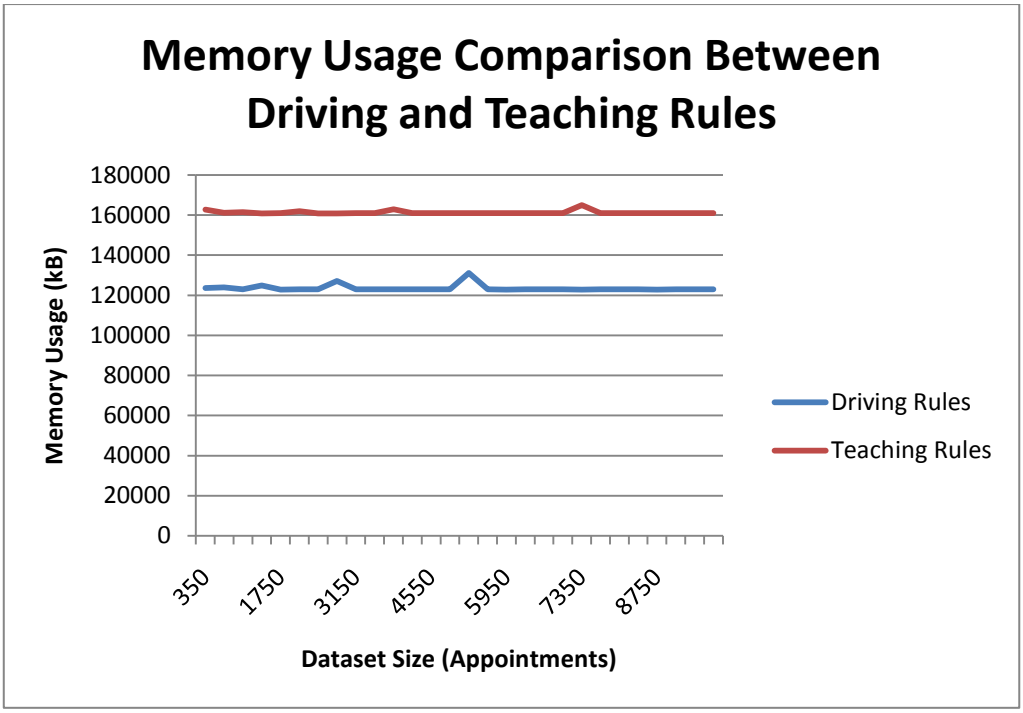


Figure 8-11 - Chart illustrating the memory usage comparison between driving and teaching rules

8.5 Comparison of Common Language Infrastructures

So far all of the development and testing work has been undertaken on a Microsoft Windows based operating systems using the Microsoft .NET CLI (Common Language Infrastructure) as the platform for executing the rule testing engine. There are various reasons for choosing this technology, as discussed in Section 3.3, but there are alternative implementations of the CLI on top of which software written using the .NET technology can run.

The most supported alternative implementation of the CLI is the Mono project and is greatly support by Novell, with many of the new developments for the OpenSUSE Linux implementation being written using the C# language and running on Mono. Unlike Microsoft's .NET CLI, the Mono CLI has been designed to be cross platform, to run on operating systems such as Linux, Mac OSX and Microsoft Windows. It can also run across multiple architectures such as x86, x86-64, ARM, s390 and the PowerPC.

One of the other benefits of Mono is that it is binary compatible with compiled .NET applications, i.e. it is able to run the compiled executables, because it is implemented to the same standard (ISO/IEC23271, 2006) as .NET.

All of the following tests use Mono 2.4.2.1 and are executed on the same hardware and operating system. As an additional measure of success, the rule engine has been tested against both .NET and Mono to provide a comparison between the computational performance of the two implementations of the CLI, to see if this has an effect on the performance of the rule testing engine.

Figure 8-12 shows the comparison between the computational performance of both the driving and teaching rules when run on .NET and on Mono. The Mono computational performance is far worse than .NET, although in some ways this is to be expected as Mono has been designed as a cross platform solution in comparison to .NET which has been finely tuned for performance under Microsoft Windows.

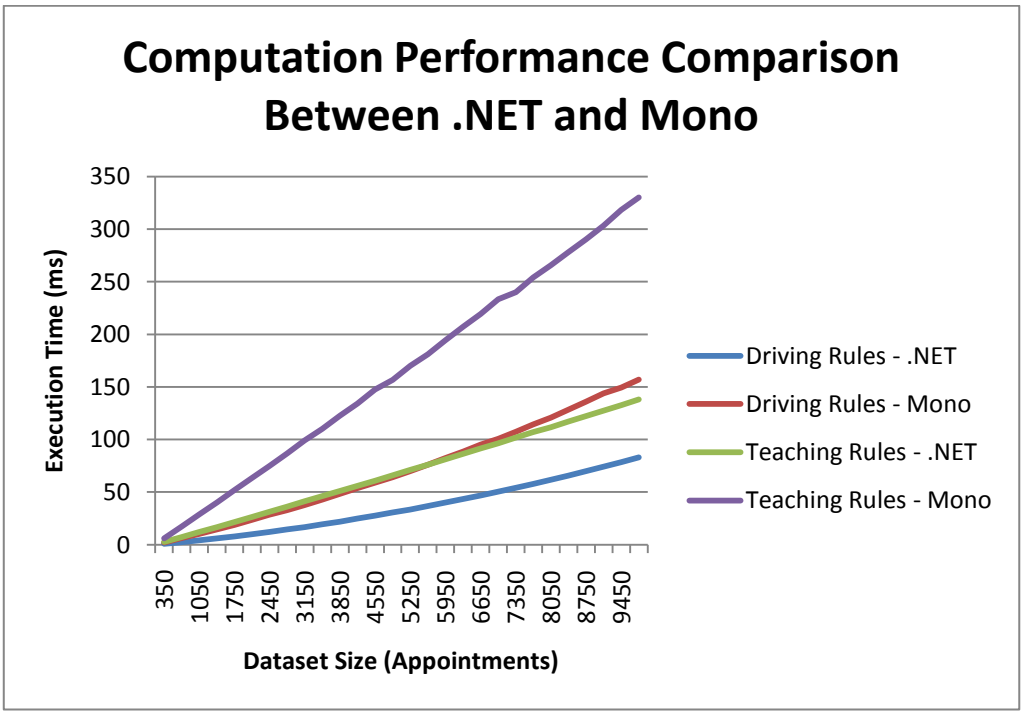


Figure 8-12 - Chart illustrating the computation performance comparison between .NET and Mono

The most interesting results in this comparison can be found in the memory usage. Figure 8-13 shows the memory usage comparison between .NET executing the rules and Mono. The .NET execution shows a smooth controlled usage of memory with only minor flutters during execution. The Mono execution shows a more sporadic memory usage across the rule testing process with large variations in the levels of memory being used. The reasons for this will be in the implementation of the two CLI's garbage collection algorithms, which is not standardised in the standard.

8.6 Comparison Across Operating Systems

As previously mentioned, the Mono implementation of the CLI standard has been designed to run across multiple operating systems. This is something that separates it from the .NET Framework, which, whilst faster and more efficient at managing memory, is unable to work on different platforms.

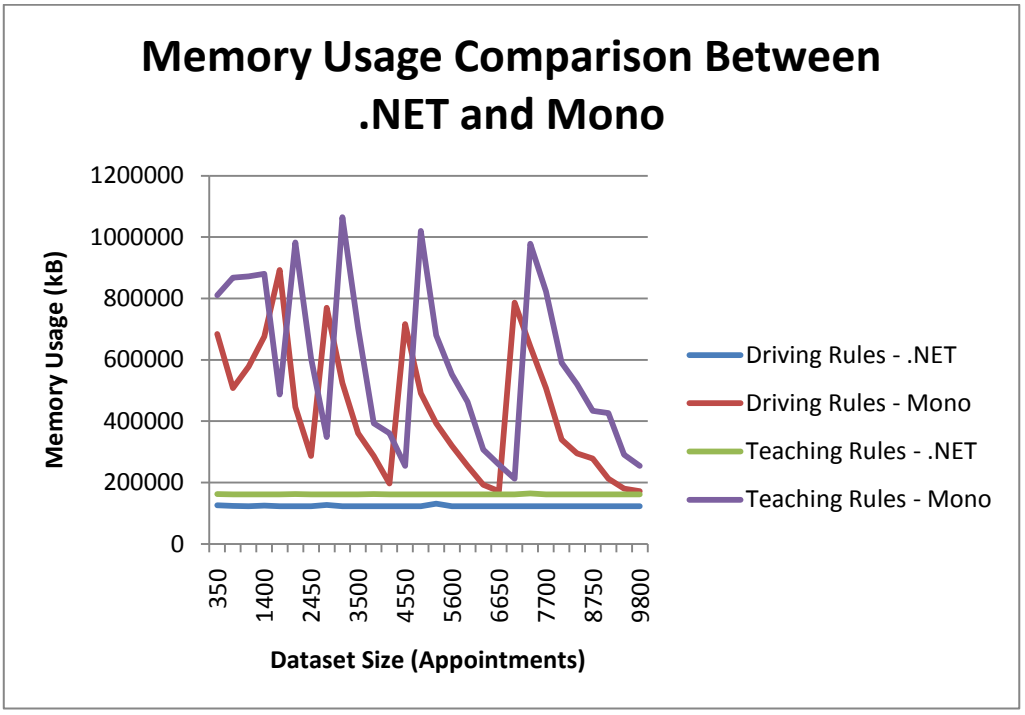


Figure 8-13 - Chart illustrating the memory usage comparison between .NET and Mono

In order to test the rule engine on alternate platforms, a slight alteration was required to the StopWatch class used to take measurements of the execution time as the Microsoft Windows based implementation depends upon the use of the High Performance Timer using native Windows API function calls. Obtaining the same level of accuracy on Unix based operating systems such as the Mac OS or Linux requires an equivalent function such as the gettimeofday function.

In order to obtain a comparison across multiple operating systems, a Mac Mini was used. Unfortunately, the only way of testing the solution using the Mac OS is to use Mac hardware so it was important, in order to use the same hardware for an equal comparison, that both Linux and Windows tests were also undertaken on this hardware. It is possible to boot a Mac into a Linux based Live CD in order to obtain test results. To run Windows Vista on a Mac requires the installation of the Windows Vista operating systems onto a separate partition and, with the support of Mac's Boot Camp facility, Windows can be dual booted.

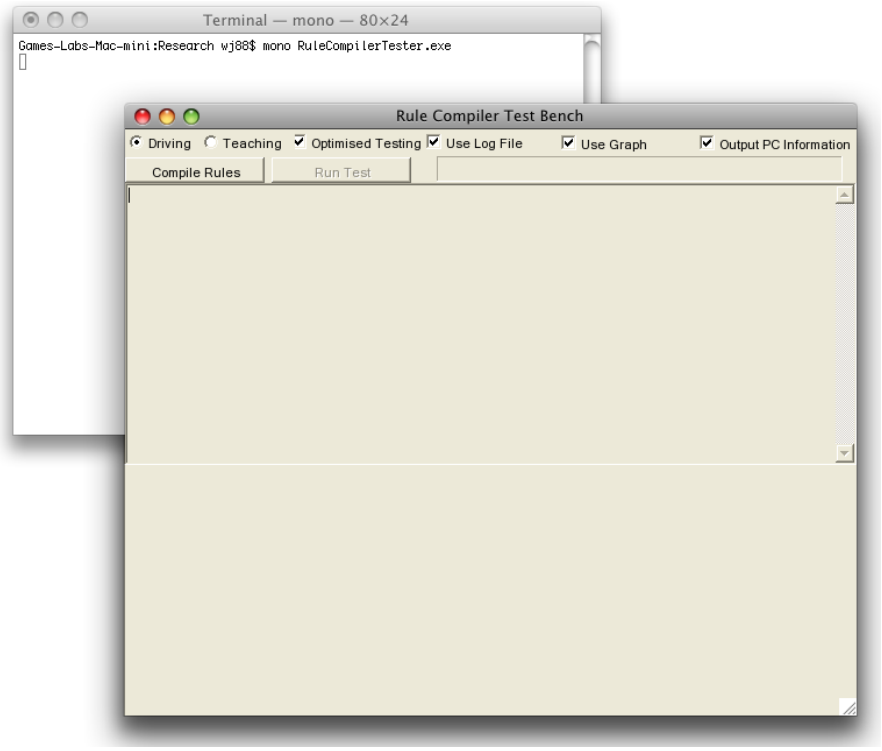


Figure 8-14 – Screen capture illustrating the test bench appearance when executed on the Mac OSX operating system

The following tests were undertaken using three different operating systems, Apple Mac OSX 10.5.7, Microsoft Windows Vista Business SP2 and openSUSE 11.1. All tests use the same version of Mono, version 2.4.2.1, and the same underlying hardware, a Mac Mini. Figure 8-14 shows the test bench software running on the Mac OS using Mono.

After running the test bench on the three platforms, some interesting results were found (Figure 8-15). Even though the physical hardware has been designed for the OSX operating system, the computational performance results showed this to be the worst performing operating system of the three. This is a strange result as you would expect, considering the hardware and OS have been designed to work together and the other two OS's are designed to work across various hardware architectures, which you would expect to see a performance drop as a result. The best performing operating system of the three is Windows Vista with OpenSUSE coming a close second.

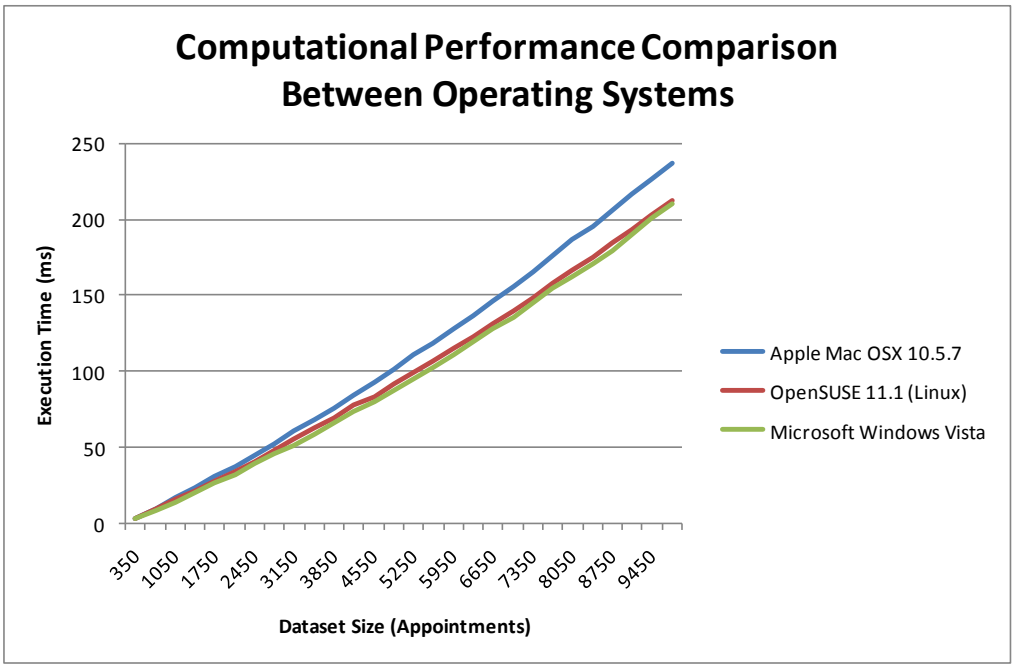


Figure 8-15 - Chart illustrating the computation performance comparison between operating systems

In terms of memory usage, Figure 8-16 shows an interesting outcome. The memory usage on the Windows platform is performing the same way as we saw in the .NET and Mono comparison, it is spiking at regular intervals. The memory usage on the other two operating systems shows a higher level of consistency than that of the Windows platform.

The reasons for the strange results from the memory tests are not entirely clear. The approaches taken to garbage collect are the same across Mono builds for the various operating systems so the reasons for the fluctuations on Windows and not the other two are not obvious. After a limited amount of investigation as to the cause, no feasible explanation was found, however, these tests have started to move outside the domain of the initial work and may be something worth coming back to.

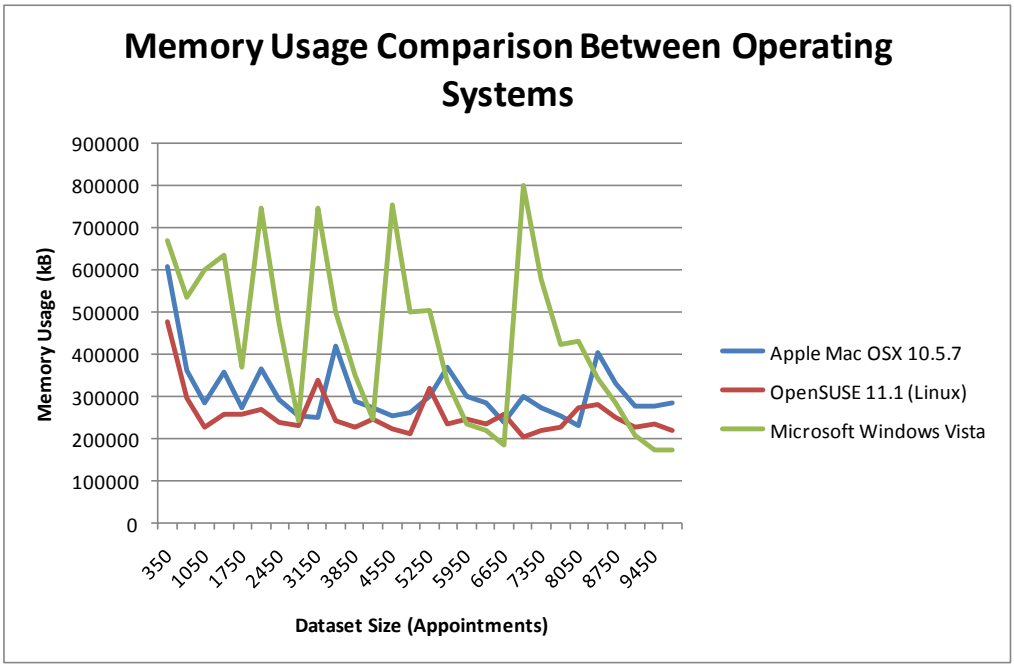


Figure 8-16 - Chart illustrating the memory usage comparison between operating systems

8.7 Summary

8 FURTHER RESULTS

In this Chapter of the thesis, an expanded set of tests have been undertaken to test the rule testing engine against an alternate set of rules. Expanding on this further, a range of tests were conducted to determine the systems performance on multiple runtimes and across a range of operating systems.

The next chapter of the thesis will draw some conclusions on the work completed and highlight how the work completed proposes a feasible solution to the problems outlined in Section 1.4.

9 CONCLUSION

As a result of undertaking the work reported in this thesis, a new method of rule testing for scheduling applications has been proposed. This provides a facility to improve the computational performance of a scheduling system requiring rule testing.

This chapter of the thesis draws conclusions on the process undertaken through the project and aims to justify the contributions that are being claimed as a result of this research.

9.1 Process

The process undertaken for this research followed, relatively closely, the RUP, which is normally found as a process for managing the flow of software engineering projects as opposed to a programme of study for a PhD. One of the justifications for using this approach is that this PhD is within the Engineering field and, as such, an engineering approach was required.

9.2 Testing

A lengthy testing process took place following the development of the solution to meet the identified requirements. This process initially focused on a single ruleset and computation performance and memory usage comparisons were made between various PC architectures. The outcomes of these tests tended to show an improved performance between dual core CPUs over single core CPUs however these results were unsurprising. An important outcome from these tests was that the developed system functioned correctly and demonstrated the concept of a rule testing engine that could be incorporated into scheduling applications.

Continuing on from the initial testing work, an additional set of tests were undertaken to demonstrate that this approach was not specifically designed for a single set of rules, but could be adapted to be used within an alternative application domain. The driving rules work was compared to a new set of teaching rules and data to analyse the impact that changing the rules would have on the computation performance and memory usage of the rule testing engine. Some interesting results were highlighted in that not just the quantity but the complexity of rules plays a large part in the performance of the rule testing engine.

Following on from the addition of a new ruleset, the computational performance and memory usage tests were undertaken on an alternate common language infrastructure to see the effects this may have on the overall system performance. It was clear from the results that execution of the system on Microsoft's .NET Framework differed substantially from that of the slower and more memory erratic Mono CLI.

Finally tests were undertaken to identify the differences across operating systems in terms of computational performance and memory usage. An interesting outcome was found in the result of these tests, in that the Mac OSX performed worst, even though the tests were running on Mac hardware.

In order to support the development and problem diagnosis of rules during their construction, a debugging method was included within the additional testing section. This played an important role in developing the new teaching rules as without this support, problems during the development process would have been very complex to diagnose. The ability to debug, pausing the execution, stepping throughout the code and monitoring variable values provided a key solution to problem diagnosis.

With respect to the testing work, there were some outcomes that were expected and naturally support the original contributions claimed by this research. These include the demonstration of the solutions computational performance and memory usage by comparison against varying PC architectures. The additional

testing work demonstrated the ability to use the same rule engine for testing a ruleset from an alternate application domain.

As well as these natural outcomes, some additional outcomes were discovered which do not necessarily contribute directly to the claims made by this research, but nevertheless may inform future work. These are that whilst the use of systems such as Mono, open source and free, are useful for providing a cross platform solution to executing software developed with .NET, the computational performance and memory usage comparisons clearly show that the .NET Framework is much more efficient at managing the execution of a process and Mono is not an effective substitute on the Windows platform.

The benefits of Mono were shown in that it is possible to execute applications compiled using .NET's C# without change. Also, the ability, with Mono, to execute the program across multiple operating systems to compare the differences in performance provides a justification for its performance to be less than that of the .NET Framework.

9.3 Justification of Contributions

The research reported in this thesis began with a detailed literature survey in the technical areas surrounding the proposed project, as reported in Chapter 2. Inherent in this process was the derivation of the target original contributions, as set out in section 2.7.

Having undertaken a plan of work aimed at meeting the Objectives outline in Section 1.2, this section sets out to review the target original contributions, and to provide a short summary of the work completed, aimed at addressing each of them.

This research will bridge the gap between rule testing and scheduling systems providing a solution where previously an optimum solution hasn't existed.

- From the literature review, a variety of research areas have been identified (Generic scheduling, transport scheduling, rule-based systems, GA's) that contribute part way toward the temporal rule testing goal, but do not succeed fully.

- The design of a rule testing language is presented in Sections 4.2.2 and 5.1. This provides a means of describing rules for a rule testing system which can function in a scheduling environment.

- The rule testing engine presented in Section 5.3 is specific to the field of scheduling systems as the metadata descriptions for each rule provide a means of supporting the work plan creation process.

It will describe a potential rule testing definition language that has been designed around the need for describing rules which relate to temporal problem domains such as scheduling.

- A rule definition language has been defined in Section 4.2.2 which can be used to serialise and deserialise rule testing ready rules (as seen in Section 4.3).

- The rule definition metadata described in Section 4.2.3 provides a means of describing rules that can be organised into a form of work plan.

- The lessons learnt in developing the rule definition language for describing temporal rules can be transferred to other rule description languages, such as RuleML(Boley, Tabet, & Wagner, Design Rationale of RuleML: A Markup Language for Semantic Web Rules, 2001), with the addition of some metadata to the rule definitions themselves.

It will demonstrate an approach to rule testing by compiling rules to optimise the speed for continuous testing scenarios such as scheduling.

- A method for compiling rules is set out in Section 5.1 to optimise the overall speed of rule testing. Whilst a discussion was presented in Section 3.3 to explain the difficulties in comparing compiled and interpreted rules, results in Chapter 7 show the computational performance comparisons of compiled rules in various scenarios.

- In addition, as a result of improving the computational performance of the rule testing itself, additional bottlenecks have been identified, addressed and tested and are reported in Chapter 8.

It will provide an example approach to modularising the rule testing engine in a way in which would allow it to fully integrate with existing systems.

- A clearly defined interface has been described in Section 4.2.1 in order to define a link between the rule testing engine and potential host systems. The standard software development interface provided has been demonstrated within the test bench in Chapter 6.

It will define a method of benchmarking the resulting solution in order that further improvements to future developments have a means of comparison.

- Chapter 6 has defined a specific test programme against which future developments within this field can test against in order to draw comparison to determine improvements to the overall rule testing process.

- Chapter 8 defines a second test case that can be used as well as approaches to testing future work across multiple software platforms and operating systems to further support future test case comparisons.

9.4 Impact

Whilst it has been recognised within the literature survey that there are various existing solutions which can integrate rules into a scheduling process, such as constraints based approaches and meta-heuristic methods, the limitation is in the need for an expert to design a scheduling algorithm which takes into account a set of rules. If the rules were to change, the expert would need to re-design the scheduling algorithm, at a substantial cost.

The approach proposed within this thesis provides a rule testing solution for scheduling applications, that can take a set of rules, described in an abstract, human readable language, and compile them into executable code, for greater performance than existing general purpose interpreted rule testing engines can provide.

The impact of such a solution would not only support existing transport scheduling systems, for which this proposed solution should be able to integrate due to its modular design, but also other scheduling areas where the testing of temporal rules is required. As discussed in Section 7.2.1, the value to an organisation who is currently undertaking the rule testing process manually is very high and, when looked at across an entire industry sector, or indeed multiple industry sectors, the potential impact is enormous.

9.5 Future Work

As a result of undertaking the research reported in this thesis a number of areas have been identified for future work. These are now presented in summary form.

9.5.1 Improved Algorithm for Work Plan construction

Currently, the solution proposed in this thesis uses a fairly simple algorithm for the generation of the work plan from the dataset provided to the rule testing process. Whilst performance increases have been identified in this research, additional work could take place in this area to further improve the performance of this bottleneck within the rule testing process.

This type of research could consider focusing more on mathematical methods for the improvement of performance as well as considerations for alternate performance gains in language specific feature areas, such as the reduction in the number method calls made in the rule testing process, to reduce the overhead of continually adding and removing stack based data.

As this method is called the most, in conjunction with the compiled rules, performance increased in this particular area could improve the overall system performance.

9.5.2 Graphical Interface for Rule Generation

It has become clear that there is a need beyond the domain of the research reported in this thesis for a tool that will allow the creation of rules using a high level intuitive graphical interface. For the generation of the rules used in the testing phase of this research, the rules were constructed programmatically and were serialised from their object states into XML.

A graphical tool would significantly improve the usability of the system for those who choose to integrate it within their scheduling system or for those rule providers who need to provide a representation of their rules to scheduling applications within their domain.

9.5.3 Multi-threaded Approach

The current state of processor manufacturer development indicates a potential for more parallelisation of the rule testing process by taking better advantage of the multi core and hyper threading technologies included within modern processors. Intel's I7 processor has 4 cores with each core enabled with hyper threading providing effectively 8 processing units that can be consumed by the operating system.

The computational performance benefits that can be found by further improving the rule testing process with the specific inclusion of parallelised work plan construction or rule testing could be significant and certainly an area for consideration in the future.

9.5.4 Hardware Acceleration

There is the potential, as in the fields of computer graphics, networking, physics and other computational expensive tasks, to off-load processing of the rule testing onto a dedicated piece of hardware. This would free up the main CPU (Central Processing Unit) for carrying out other standard tasks required by the Operating System and the user whilst the rule testing process is underway.

This would require a considerable amount of additional investigation in order to identify which aspects of the scheduling and rule testing processes should be off-loaded onto a dedicated processor and which should remain part of the job for the main CPU. A number of other research fields are also considering this method for performance increases, especially with the development of technology such as the Compute Unified Device Architecture (CUDA).

9.5.5 Validation Checking

The proposed solution does not undertake any validation checking of the rules to determine whether the initial set of rules are actually valid. There are two elements to the validation checking problem that need consideration:

1. Are the XML representation of the rules valid, i.e. do they meet with the XML standard?
2. Do any of the rules contradict any of the other rules?

Further work could be undertaken to address these two issues, the second of which, it is expected, would be a complex topic to address.

9.5.6 Integration with Existing Scheduling Systems

An important area of further work which is less of a research problem and more of an application problem is the further testing and demonstration of the ability to integrate the proposed solution within a variety of existing scheduling systems. This is the natural next step for the outcome of this research, both in terms of further proving the concept but also, more importantly, in an effort toward the commercialisation of the proposed solution.

One such example of this is the potential integration of the proposed solution into the GAS system, described in Section 2.2.1.

9.6 Summary

In summary this chapter has summarised the work completed, aimed at addressing the target original contributions. Additionally, Section 9.5 has identified a number of potential areas for future work.

10 REFERENCES

Abramson, D., & Adela, J. (1991). A Parallel Genetic Algorithm for Solving the School Timetabling Problem. *IJCAI workshop on Parallel Processing in AI.*

Alba, E., & Troya, J. (2002). Improving Flexibility and Efficiency by Adding Parallelism to Genetic Algorithms. *Statistics & Computing* , 91-114.

Ambler, S. (2004). *The Object Primer: Agile Model-Driven Development with UML 2.0* (3rd Edition ed.). Cambridge University Press.

Ammons, G., Bodik, R., & Larus, J. (2002). Mining Specifications. *Proceedings of the 29th ACM Symposium on Principles of Programming Languages.*

Baranovsky, A. (n.d.). *paxScript.NET*. Retrieved 01 2010, from http://www.paxscript.net

Bartak, R. (2003). Visopt ShopFloor: On The Edge Of Planning And Scheduling. *International Conference on Automated Planning and Scheduling.* Toronto.

Beck, J. C., & Fox, M. S. (1998). A Generic Framework for constraint-directed search and scheduling. *AI Magazine* .

Bittner, K. &. (2002). *Use Case Modeling.* Addison Wesley Professional.

Bittner, K. (2006). *Driving Iterative Development With Use Cases.* IBM.

Boley, H. (2001). The Rule Markup Language: RDF-XML Data Model, XML Schema Hierarchy, and XSL Transformations. *14th International Conference of Applications of Prolog (INAP).*

Boley, H., Paschke, A., Tabet, S., & Grosof, B. (2010). *Schema Specification of RuleML 1.0*. Retrieved July 2010, from http://ruleml.org/1.0/

Boley, H., Tabet, S., & Wagner, G. (2001). Design Rationale of RuleML: A Markup Language for Semantic Web Rules. *Proceedings of the International Semantic Web Working Symposium.*

Bollow, N. (n.d.). *DotGNU Project*. Retrieved 01 2010, from http://www.gnu.org/projects/dotgnu/

Bray, T., Paoli, J., Sperberg-McQueen, C., & Maler, E. (2000). *eXtensible Markup Language (XML) 1.0 (Second Edition).* W3C Recommendation.

Burke, E., & Petrovic, S. (2002). Recent Research Directions in Automated Timetabling. *European Journal of Operational Research , 140* (2), 266-280.

Cesta, A., Oddi, A., & Susi, A. (2000). Fast Production and Flexible Maintenance of Schedule for Space Applications. *World Automation Congress.*

Chen, L. C., Wetherall, J., & Doncheva, R. (2005). Performance optimisation of rule-based testing in scheduling within a distributed processing environment (A). *Proceedings of the ESPRC PREP 2005.* Lanaster, UK.

Chen, L. C., Wetherall, J., & Doncheva, R. (2005). Performance optimisation of rule-based testing in scheduling within a distributed processing environment (B). *Proceedings of the Joint DCABES and ICPACE Meeting.*

Dail, H. J. (2002). *A modular framework for adaptive scheduling in Grid Application Development Environments.* San Diago: University of California.

de Icaza, M., & Jepson, B. (2002). Mono. *Dr. Dobb's Journal of Software Tools* .

Dietrich, J. (2003). *The Mandarax Manual.* Massey University, New Zealand: Institute of Information Sciences & Technology.

Dietrich, J., Hiller, J., & Schenke, B. (2007). Take - A Rule Compiler for Derivation Rules. *Advances in Rule Interchange and Applications* , 134-148.

Dumbill, E. B. (2004). *Mono (Developer's Notebook).* O'Reilly Media.

Fischetti, M., Lodi, A., Martello, S., & Toth, P. (2001). A Polyhedral Approach to Simplified Crew Scheduling and Vehicle Scheduling Problems. *Management Science* , 833-850.

Fores, S., & Proll, L. (1998). Driver scheduling by integer linear programming-the TRACS II approach. *Proceedings CESA'98 Computational Engineering in Systems Applications Symposium on Industrial and Manufacturing Systems, LÚnion des Chercheurs et Ingenieurs Scientifiques (UCIS).*

Forrest, S. (1996). Genetic Algorithms. *ACM Computing Surveys (CSUR)* , 77-70.

Freling, R., Huisman, D., & Wagelmans, A. (2000). *Applying an Integrated Approach to Vehicle and Crew Scheduling in Practice.* Erasmus Research Institute of Management, Erasmus University Rotterdam.

Freling, R., Huisman, D., & Wagelmans, A. (2003). Models and Algorithms for Integration of Vehicle and Crew Scheduling. *Journal of Scheduling* , 63-85.

Geer, D. (2005). Chip Makers Turn to Multicore Processors. *IEEE Computer Society* .

Goncalves, J., Mendes, J., & Resende, M. (2008). A Genetic Algorithm for the Resource Constrained Multi-Project Scheduling Problem. *European Journal of Operational Research , 189* (3), 1171-1190.

Green, J. (2001). Buyer's Guide: Job Scheduling Software. *Windows 2000 Magazine* .

Hayes-Roth, F. (1985). Rule-Based Systems. . *Communications of the ACM* , 921-932.

Hericko, M., Juric, M. B., Rozman, I., Beloglavec, S., & Zivkovic, A. (2003). Object serialization analysis and comparison in Java and .NET. *ASM SIGPLAN Notices*, (pp. 44-54).

ISO/IEC23270. (2006). *C# Language Specification.* International Organisation for Standardization (ISO).

ISO/IEC23271. (2006). *Common Language Infrastructure (CLI) Partitions I to VI.* International Organisation for Standardization (ISO).

JSR-199. (n.d.). *JavaTM Compiler API.* Retrieved from Java Community Process: http://jcp.org/en/jsr/detail?id=199

JSR-94. (n.d.). *JavaTM Rule Engine API.* Retrieved 2010, from Java Community Process: http://jcp.org/en/jsr/detail?id=94

Kerningham, B., & Van Wyk, C. (1998). Timing Trials, or the Trials of Timing: Experiments with Scripting and User-Interface Languages. *Software - Practice and Experience,* (pp. 819-843).

Krintz, C., Grove, D., Lieber, D., Sarkar, V., & Calder, B. (2001). Reducing the Overhead of Dynamic Compilation. *Software - Practice and Experience*, (pp. 717-738).

Kwan, R., Kwan, A., & Wren, A. (2001). Evolutionary Driver Scheduling with Relief Chains. *Evolutionary Computation , 9* (4), 445 - 460.

Lee, J. K., & Sohn, M. M. (2003). The Extensible Rule Markup Language. *Communications of the ACM* , 59-64.

Liebowitz, J. (1997). Intelligent scheduling with GUESS (Generically used expert scheduling system): development and testing result. *International Journal of Knowledge Engineering and Neural Networks (Expert Systems)* .

Lu, C., Stankovic, J. A., Tao, G., & Son, S. H. (2001). Feedback Control Real-Time Scheduling: Framework, Modelling, and Algorithms. *Journal of Real-Time Systems* .

Mak, B., & Blanning, R. (2003). A logic-based approach to rule induction in expert systems. *International Journal of Knowledge Engineering and Neural Networks* .

Malachy, C., & Crawford, I. (2007). Scheduling Trains on a Network of Busy Complex Stations. *Transportation Research Part B: Methodological , 41* (2), 159-178.

Martijn, M., van der Heijden, M., & van Harten, A. (2007). Comparison of Agent-Based Scheduling to Look-Ahead Heuristics for Real-Time Transportation Problems. *European Journal of Operational Research , 181* (1), 59-75.

Newhall, T. (1999). *Performance Measurement of Interpreted, Just-In-Time Compiled and Dynamically Compiled Executions: PhD Dissertation.* Madison: University of Wisconsin.

Pan, J. (2005). Requirements for a Semantic Web Rule Language. *W3C Workshop on Rule Languages for Interoperability.* Washington D.C., USA.

Pavel, O., Ivan, S., & Jan, R. (1996). Multilevel Distributed Genetic Algorithms. *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications.*

Peters, E., de Matta, R., & Boe, W. (2007). Short-Term Work Scheduling with Job Assignment Flexibility for a Multi-Fleet Transport System. *European Journal of Operational Research , 180* (1), 82-98.

Pinto., I. G., Monterio, V. S., & Rosa, A. C. (1999). Distributed Genetic Algorithms Using DLL. *IEEE International Conference on Systems, Man, and Cybernetics.* Tokyo, Japan.

Proth, J.-M. (2007). Scheduling: New Trends in Industrial Environment. *Annual Reviews in Control , 31* (1), 157-166.

RuleML. (n.d.). Retrieved 2010, from RuleML: http://ruleml.org/

Rumbaugh, J., Jacobson, I., & Booch, G. (2004). *The Unified Modeling Language Reference Manual.* Addison-Wesley.

Shadbolt, N., Berners-Lee, T., & Hall, W. (2006). The Semantic Web Revisited. *IEEE Intelligent Systems* , 93-101.

Sommerville, I. (2006). *Software Engineering 8.* Addison Wesley.

Stutz, D., Neward, T., & Shilling, G. (2003). *Shared Source CLI Essentials.*

Subramanian, L., Katz, R. H., & Franklin, J. (2000). Aggregate Scheduling: Enhancing Throughput in Collective Tasking Systems. *Fourth USENIX Symposium on Operating Systems Design and Implementation (OSDI).* San Diego.

Thompson, H., Tobin, R., & Connolly, D. (2005). *Validator for XML Schema*, XSV 3.1-1 of 2007/12/11 16:20:05. Retrieved July 2010, from http://www.w3.org/2001/03/webdata/xsv

Tongchim, S., & Chongstitvatana, P. (2002). Parallel Genetic Algorithm with Parameter Adaption. *Information Processing Letter* , pp. 47-54.

Transport, D. o. (1998). *Drivers' Hours and Tachograph Rules.* UK.

Wagner, G. (2002). *How to Design a General Rule Markup Language. Technologies for the Semantic Web Workshop Proceedings.*

Weerdt, M. (1999). *Towards Algorithms For Scheduling Of Transport Processes.* TRAIL Research School, Delft University of Technology.

Wetherall, J. (2002). *Kings Ferry Coaches, Intermediate Progress Report.* School of Engineering, University of Greenwich.

Whaley, J., Martin, M., & Lam, M. (2002). Automatic Extraction of Object Oriented Component Interfaces. *Proceedings of the ACM SIGSOFT 2002.* International Symposium on Software Testing and Analysis.

Wren, A., Fores, S., Kwan, A., Kwan, R., Parker, M., & Proll, L. (2002). *A Flexible System for Scheduling Drivers.* University of Leeds Research Report Series. University of Leeds.

11 APPENDICES

11.1 Object Serialisation Example

An example of object serialisation using the C# programming languages is as follows. This simple example uses a class called Value which has two public variables, Description and Type. Figure 11-1 provides an example of how this class is defined.

```
public class MyClass
{
    public string Description;
    public string Value;
}
```

Figure 11-1 - XML Serialisation Example A

The serialisation process is made very simple in the sense that there are a set of classes that exist to support the process of serialisation and deserialisation. Figure 11-2 provides an example of the serialisation process using the .NET Framework approach to XML object serialisation. The variable `outputStream` being passed in as the first parameter of the `Serialize` method of the `XMLSerializer` object can be any type of stream such as a File stream or even a Network stream.

```
// Create a new instance of MyClass and set some values
// to the public variables.
MyClass myClass = new MyClass();
myClass.Description = "Hello World";
myClass.Value = "Anything";

// Create the XmlSerializer object which will be used to
// convert the object into a stream of XML.
XmlSerializer newSerializer = new XmlSerializer(typeof(MyClass));

// Perform the serialization of the array to the stream.
newSerializer.Serialize(outputStream, myClass);
```

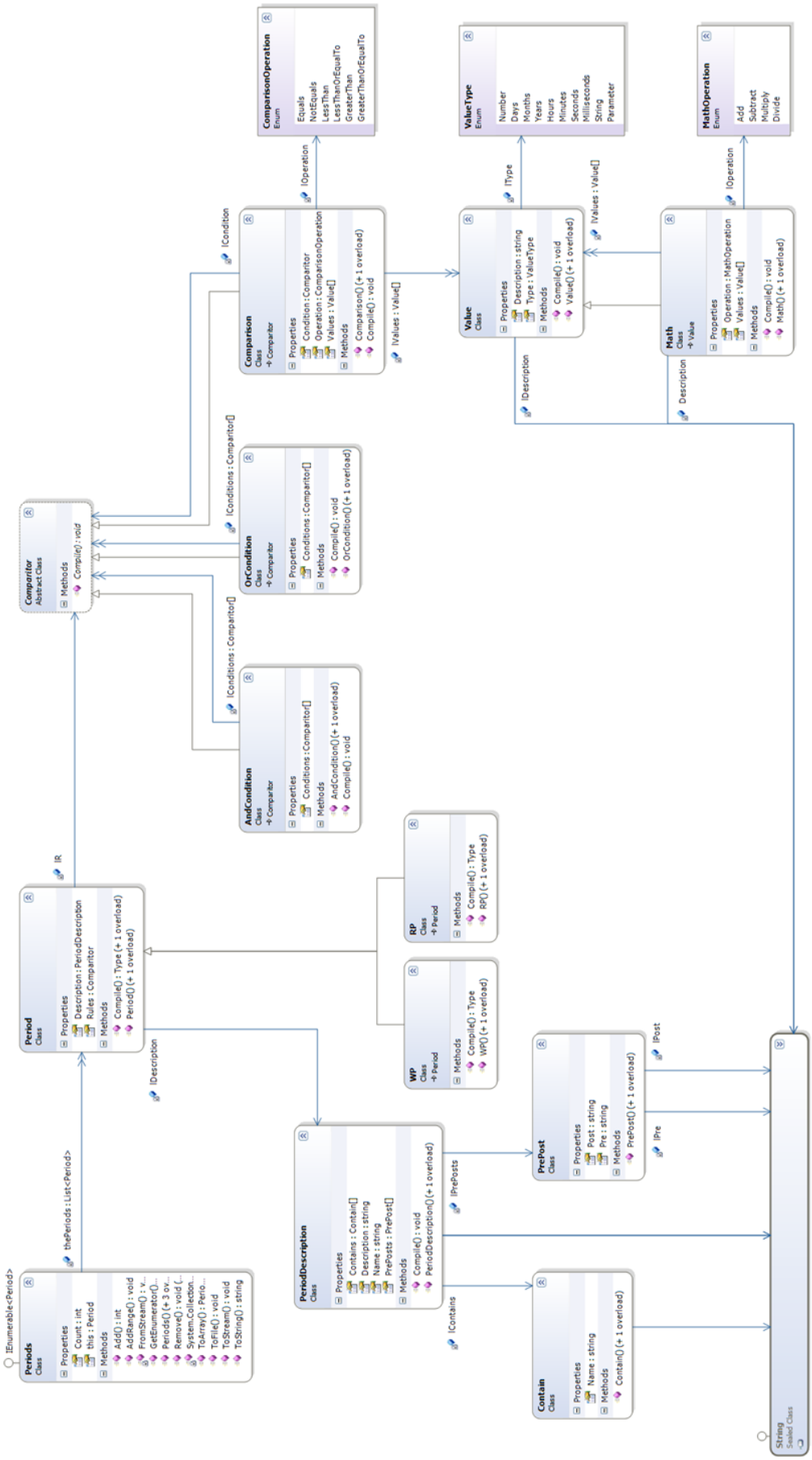
Figure 11-2 - XML Serialisation Example B

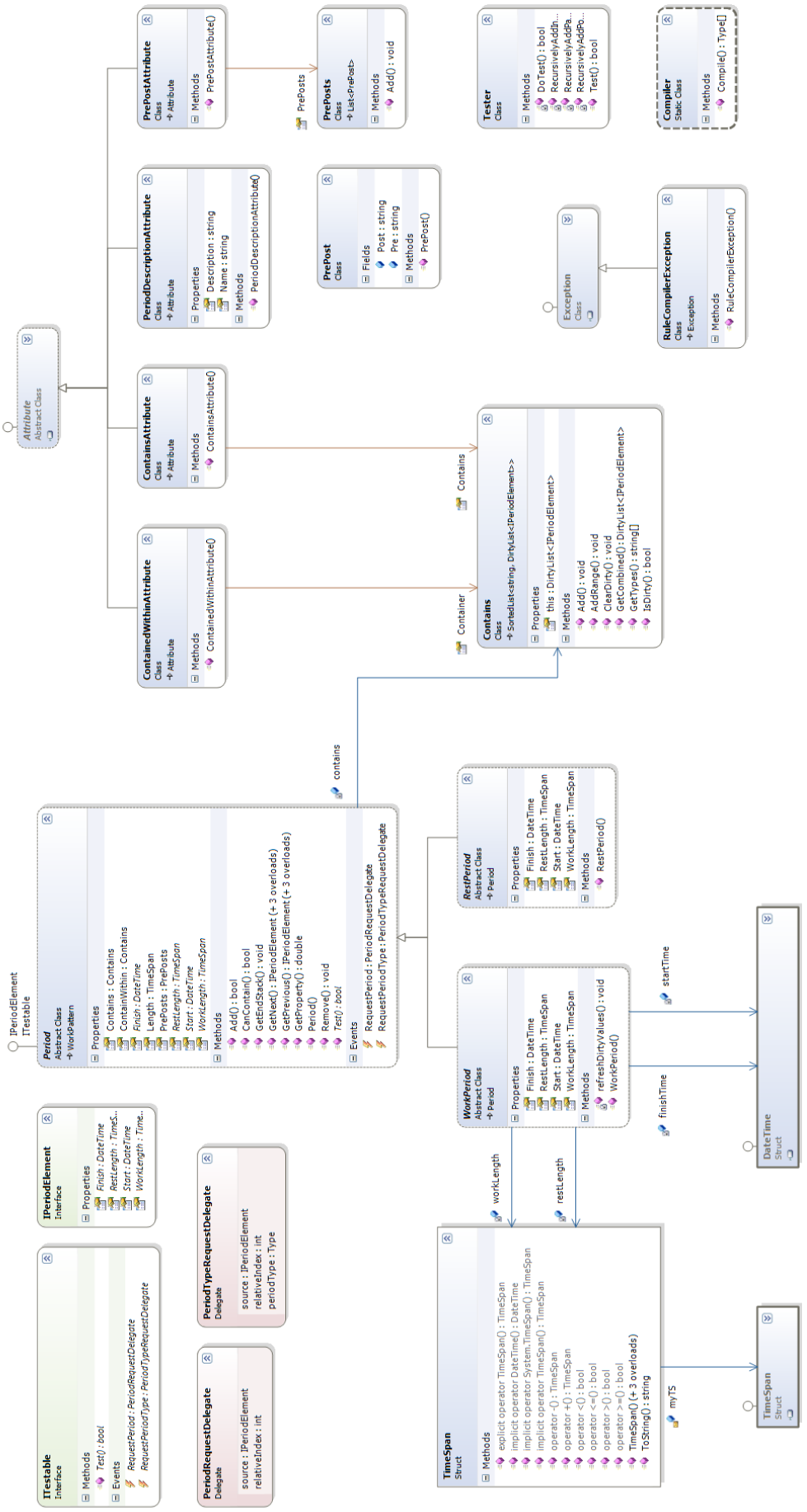
Now that serialisation of the object has taken place the XML representation exists at the other end of the `outputStream`. This means that it would reside in a file if it were a file stream or even on another machine if it were a network stream. Figure 11-3 shows the resulting XML representation of the object as produced by the `XmlSerializer` object in .NET.

```
<?xml version="1.0"?>
<MyClass>
  <Description>Hello World</Description>
  <Value>Anything</Value>
</MyClass>
```

Figure 11-3 - XML Serialisation Example C

11.2 Class Diagrams


ComparisonOperation
Enum
Equals
NotEquals
LessThan
LessThanOrEqualTo
GreaterThan
GreaterThanOrEqualTo
ValueType
Enum
Number
Days
Months
Years
Hours
Minutes
Seconds
Milliseconds
String
Parameter
MathOperation
Enum
Add
Subtract
Multiply
Divide
IOperation
IType
IValues : Value[]
IDescription
Description
ICondition
IConditions : Comparitor[]
IR
IPost
IPre
IPrePosts
IContains
IEnumerable<Period>
thePeriods : List<Period>
Comparitor
Abstract Class
Methods
Compile() : void
Comparison
Class
+ Comparitor
Properties
Condition : Comparitor
Operation : ComparisonOperation
Values : Value[]
Methods
Comparison() (+ 1 overload)
Compile() : void
Value
Class
Properties
Description : string
Type : ValueType
Methods
Compile() : void
Value() (+ 1 overload)
Math
Class
+ Value
Properties
Operation : MathOperation
Values : Value[]
Methods
Compile() : void
Math() (+ 1 overload)
OrCondition
Class
+ Comparitor
Properties
Conditions : Comparitor[]
Methods
Compile() : void
OrCondition() (+ 1 overload)
AndCondition
Class
+ Comparitor
Properties
Conditions : Comparitor[]
Methods
AndCondition() (+ 1 overload)
Compile() : void
Period
Class
Properties
Description : PeriodDescription
Rules : Comparitor
Methods
Compile() : Type (+ 1 overload)
Period() (+ 1 overload)
RP
Class
+ Period
Methods
Compile() : Type
RP() (+ 1 overload)
WP
Class
+ Period
Methods
Compile() : Type
WP() (+ 1 overload)
IDescription
PeriodDescription
Class
Properties
Contains : Contain[]
Description : string
Name : string
PrePosts : PrePost[]
Methods
Compile() : void
PeriodDescription() (+ 1 overload)
PrePost
Class
Properties
Post : string
Pre : string
Methods
PrePost() (+ 1 overload)
Contain
Class
Properties
Name : string
Methods
Contain() (+ 1 overload)
Periods
Class
Properties
Count : int
this : Period
Methods
Add() : int
AddRange() : void
FromStream() : v...
GetEnumerator()...
Periods() (+ 3 ov...
Remove() : void (...
System.Collection...
ToArray() : Pero...
ToFile() : void
ToStream() : void
ToString() : string
String
Sealed Class


Attribute
Abstract Class
PrePostAttribute
Class
+ Attribute
Methods
PrePostAttribute()
PrePosts
PrePosts
Class
+ List<PrePost>
Methods
Add() : void
PeriodDescriptionAttribute
Class
+ Attribute
Properties
Description : string
Name : string
Methods
PeriodDescriptionAttribute()
PrePost
Class
Fields
Post : string
Pre : string
Methods
PrePost()
Tester
Class
Methods
DoTest() : bool
RecursivelyAddPr...
RecursivelyAddPa...
RecursivelyAddPo...
Test() : bool
Compiler
Static Class
Methods
Compile() : Type[]
ContainsAttribute
Class
+ Attribute
Methods
ContainsAttribute()
Exception
Class
RuleCompilerException
Class
+ Exception
Methods
RuleCompilerException()
ContainedWithinAttribute
Class
+ Attribute
Methods
ContainedWithinAttribute()
Contains
Container
Contains
Class
+ SortedList<string, DirtyList<IPeriodElement>>
Properties
this : DirtyList<IPeriodElement>
Methods
Add() : void
AddRange() : void
ClearDirty() : void
GetCombined() : DirtyList<IPeriodElement>
GetTypes() : string[]
IsDirty() : bool
IPeriodElement
ITestable
Period
Abstract Class
+ WorkPattern
Properties
Contains : Contains
ContainWithin : Contains
Finish : DateTime
Length : TimeSpan
PrePosts : PrePosts
RestLength : TimeSpan
Start : DateTime
WorkLength : TimeSpan
Methods
Add() : bool
CanContain() : bool
GetEndStack() : void
GetNext() : IPeriodElement (+ 3 overloads)
GetPrevious() : IPeriodElement (+ 3 overloads)
GetProperty() : double
Period()
Remove() : void
Test() : bool
Events
RequestPeriod : PeriodRequestDelegate
RequestPeriodType : PeriodTypeRequestDelegate
contains
RestPeriod
Abstract Class
+ Period
Properties
Finish : DateTime
RestLength : TimeSpan
Start : DateTime
WorkLength : TimeSpan
Methods
RestPeriod()
WorkPeriod
Abstract Class
+ Period
Properties
Finish : DateTime
RestLength : TimeSpan
Start : DateTime
WorkLength : TimeSpan
Methods
refreshDirtyValues() : void
WorkPeriod()
startTime
finishTime
workLength
restLength
DateTime
Struct
IPeriodElement
Interface
Properties
Finish : DateTime
RestLength : TimeS...
Start : DateTime
WorkLength : Time...
ITestable
Interface
Methods
Test() : bool
Events
RequestPeriod : PeriodRequestDelegate
RequestPeriodType : PeriodTypeRequestDelegate
PeriodTypeRequestDelegate
Delegate
source : IPeriodElement
relativeIndex : int
periodType : Type
PeriodRequestDelegate
Delegate
source : IPeriodElement
relativeIndex : int
TimeSpan
Struct
Methods
explicit operator TimeSpan() : TimeSpan
implicit operator DateTime() : DateTime
implicit operator System.TimeSpan() : TimeSpan
implicit operator TimeSpan() : TimeSpan
operator -() : TimeSpan
operator +() : TimeSpan
operator <() : bool
operator <=() : bool
operator >() : bool
operator >=() : bool
TimeSpan() (+ 3 overloads)
ToString() : string
myTS
TimeSpan
Struct

11.3 Code Listing

11.3.1 RuleCompiler Namespace

11.3.1.1 Compiler Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use this library for providing dynamically sizable collections.
using System.Collections.Generic;

// Use these libraries for enabling Reflection of the classes into
// executable code.
using System.Reflection;
using System.Reflection.Emit;

// Use all of the types defined in the RuleDefinitionLanguage to do the
// majority of the compilation process.
using RuleDefinitionLanguage;

/// <summary>
/// The RuleCompiler namespace contains all of the functionality
/// required to support the compilation of the rules in the rule
/// definition language to executable code.
/// </summary>
namespace RuleCompiler
{

        /// <summary>
        /// Compiler Class.
        /// This class is used to compile rules into executable code.
        /// </summary>
        public static class Compiler
        {

                /// <summary>
                /// The Compile method is used to take in the Rule Description
                /// Language and return a Type object ready for execution.
                /// </summary>
                /// <param name="periods">
                /// The periods parameters defines the collection of period
                /// objects from which the new Types are to be created.
                /// </param>
                /// <returns>
                /// Returns a Type array from which new instances can be created
                /// and instantiated.
                /// </returns>
                public static Type[] Compile(Periods periods, Type[]
datasetTypes)
                {

                        // Create a collection to store type mapping information
                        // which is used to allow us to see the relationship
                        // between the various data types.
                        SortedList<string, TypeMap> typeMapping = new
SortedList<string, TypeMap>();

                        // Find out what type of data we are looking at so that
                        // it can be related to the periods types.
                        for(int a = 0; a < datasetTypes.Length; a++)
                        {
                                String typeName = datasetTypes[a].Name;
                                if(!typeMapping.ContainsKey(typeName))
                                {
                                        typeMapping.Add(typeName, new
TypeMap(typeName));
                                }
                        }

                        // Create a mapping between the periods and the dataset
                        // so that we can build up a relationship list between
                        // entries.
```

```
                        foreach(Period period in periods)
                        {

                                // Add the object names which are contained
                                // within the period object.
                                TypeMap typeMap = new
TypeMap(period.Description.Name);

                                foreach(Contain contain in
period.Description.Contains)
                                        typeMap.Contains.Add(contain.Name);

                                typeMapping.Add(period.Description.Name,
typeMap);

                        }

                        // Create a cross-relationship to allow an object to
                        // know what objects it is contained within.
                        foreach(TypeMap typeMap in typeMapping.Values)
                        {
                                for(int c = 0; c < typeMap.Contains.Count; c++)
                                {

                                        // Check to ensure that we have a typeMap
                                        // for the type that is being
                                        // cross-referenced. If not then it is
                                        // necessary to throw an exception as we
                                        // are missing necessary information.
                                        String containName = (String)
typeMap.Contains[c];
                                        if(!typeMapping.ContainsKey(containName))
                                                throw new
RuleCompilerException("Unknown Type: " + containName);
                                        else
                                                ((TypeMap)
typeMapping[containName]).ContainWithin.Add(typeMap.Name);

                                }
                        }

                        // Perform a check to ensure that all of the prepost
                        // entries have type names which are listed in our
                        // type map. If not then we must throw an exception.
                        foreach(Period period in periods)
                        {
                                foreach(PrePost prePost in
period.Description.PrePosts)
                                {
                                        if((prePost.Pre != "") &&
(!typeMapping.ContainsKey(prePost.Pre)))
                                                throw new
RuleCompilerException("Unknown Type: " + prePost.Pre + "\nDefined within rule:
" + period.Description.Name + " as post.");
                                        if((prePost.Post != "") &&
(!typeMapping.ContainsKey(prePost.Post)))
                                                throw new
RuleCompilerException("Unknown Type: " + prePost.Post + "\nDefined within rule:
" + period.Description.Name + " as post.");

                                        ((TypeMap)
typeMapping[period.Description.Name]).PrePost.Add(new PrePost(prePost.Pre,
prePost.Post));
                                }
                        }

                        // Perform a check to ensure that each of the give
                        // dataset types can be contained by at least one
                        // of the periods, otherwise we would wont be able
                        // to test the dataset type later.
                        foreach(Type datasetType in datasetTypes)
                        {
                                String typeName = datasetType.Name;


        if(((TypeMap)typeMapping[typeName]).ContainWithin.Count == 0)
```

```
                                        throw new
RuleCompilerException("Uncontained Type: " + typeName + "\nThe names type is
not contained by a rule statement and therefore cannot be tested.");
                        }

                        // Declare the necessary assembly builder variables for
                        // use in this method.
                        AssemblyName assemblyName = null;
                        AssemblyBuilder builder = null;
                        ModuleBuilder module = null;

                        // Intialise the new Assembly and declare it is run only
                        // for the time beging. Later in may be worth look at
                        // ways of caching the compiled assembly to prevent
                        // recompilation but for the time being, we will
                        // recompile.
                        assemblyName = new AssemblyName();
                        assemblyName.Name = "TestAssembly";
                        builder =
AppDomain.CurrentDomain.DefineDynamicAssembly(assemblyName,
AssemblyBuilderAccess.RunAndSave);

                        // Define the module within the Assembly that will
                        // contain the new types.
                        module = builder.DefineDynamicModule("TestModule",
"Test.dll", DebugSupport.IsDebuggable);

                        // Define the object that will deal with debug support
                        // for us.
                        DebugSupport debugSupport = new DebugSupport(builder,
module);

                        // Define an array to store the new types, one type per
                        // period, then, using a loop, get each period to
                        // compile itself.
                        Type[] theTypes = new Type[periods.Count];
                        for(int i = 0; i < periods.Count; i++)
                                theTypes[i] = periods[i].Compile(module,
typeMapping[periods[i].Description.Name], debugSupport);

                        // Dispose the debugSupport object, ensuring the source
                        // file gets closed properly.
                        debugSupport.Dispose();

                        // Save the compiled rules into a dynamic link library
                        builder.Save("Test.dll");

                        // Save an XML representation of the rules along with
                        // the DLL to allow for debugging later.
                        return theTypes;
                }

        }

}
```

11.3.1.2 DebugSupport Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use this library for providing dynamically sizable collections.
using System.Collections.Generic;

// Use these libraries for access to the classes required for emitting
// debug information.
using System.Diagnostics;
using System.Diagnostics.SymbolStore;

// Use this library for access to the file relate functionality.
using System.IO;
```

```
// Use these libraries for enabling Reflection of the classes into
// executable code.
using System.Reflection.Emit;

/// <summary>
/// The RuleCompiler namespace contains all of the functionality
/// required to support the compilation of the rules in the rule
/// definition language to executable code.
/// </summary>
namespace RuleCompiler
{

    /// <summary>
    /// The DebuggingCodeMarker struct is used by the
    /// DebuggingSupport class to identify the line and
    /// character position in a snapshot of the emitted
    /// file.
    /// </summary>
    public struct DebugCodeMarker
    {
        /// <summary>
        /// The Line parameters stores the current line
        /// in the source file to which this marker refers.
        /// </summary>
        public int Line;

        /// <summary>
        /// The CharPos parameter stores the current column
        /// in the source file to which this marker refers.
        /// </summary>
        public int CharPos;

        /// <summary>
        /// A simple constructor used to provide the line and
        /// charpos on construction.
        /// </summary>
        public DebugCodeMarker(int line, int charPos)
        {
            Line = line;
            CharPos = charPos;
        }
    }

    /// <summary>
    /// The DebuggingSupport file contains everything required
    /// to include the ability to include debug information
    /// with the compiled rules.
    /// </summary>
    public class DebugSupport : IDisposable
    {
        /// <summary>
        /// The debuggable variable is used locally to store our
        /// debuggability.
        /// </summary>
        private bool debuggable;

        /// <summary>
        /// The sourceFile variable is used locally to store the
        /// location of the sourceFile.
        /// </summary>
        private string sourceFile;

        /// <summary>
        /// The sourceWriter variable is used locally to store the
        /// stream writer for churning source code out to the file.
        /// </summary>
        private StreamWriter sourceWriter;

        /// <summary>
        /// The symbolDocument variable is used locally when writing
        /// source code marker points into the compiled rules assembly.
        /// </summary>
        private ISymbolDocumentWriter symbolDocument;

        /// <summary>
```

```
        /// The charPos variable is used locally to track the charPos
        /// of the current character in the source file.
        /// </summary>
        private int charPos = 0;

        /// <summary>
        /// The lineCount variable is used locally to track the number
        /// of lines stored in the source file.
        /// </summary>
        private int lineCount = 1;

        /// <summary>
        /// The tabCount variable is used locally to track the number of
        /// tabs to indent newly written source code to the source file
        /// with.
        /// </summary>
        private int tabCount = 0;

        /// <summary>
        /// The varCounts variable is used to track the number of
        /// occurances of variables with similar types in order that each
        /// can be assigned a unique number.
        /// </summary>
        SortedList<string, int> varCounts;

        /// <summary>
        /// The marker stack is used to store start markers for the
        /// beginning of a code segment within the source code. Calling
        /// StartMarker will add a new entry to the top of the stack and
        /// calling EndMarker will pop an entry off, using it to complete
        /// the beginning and end of a segment of code.
        /// </summary>
        Stack<DebugCodeMarker> markers;

        /// <summary>
        /// The TabCount property returns the current number of tabs that
        /// the source is to be indented with.
        /// </summary>
        public int TabCount
        {
            get
            {
                return tabCount;
            }
        }

        /// <summary>
        /// The Debuggable property describes when the compiliation process
        /// is currently emitting debug information.
        /// </summary>
        public bool Debuggable
        {
            get
            {
                return debuggable;
            }
        }

        /// <summary>
        /// The static IsDebuggable property determines firstly when the
        /// RuleCompiler library is built to include debug symbals and
        /// secondly whether a debugger is attached. The result is a
        /// combination of the two to determine whether or not to debug.
        /// </summary>
        public static bool IsDebuggable
        {
            get
            {
#if DEBUG
                return (Debugger.IsAttached);
#else
                return false;
#endif
            }
        }
```

```
        /// <summary>
        /// The StartMarker method starts a new segment marker point within
        /// the source code at the current line and character position,
        /// adding it to the marker stack.
        /// </summary>
        public void StartMarker()
        {
            // We only carry out this functionality if the new code is
            // debuggable.
            if (!debuggable) return;

            // Crete a new start marker using out current line and character
            // position.
            DebugCodeMarker startMarker = new DebugCodeMarker(lineCount,
charPos);

            // Push this new marker onto our stack.
            markers.Push(startMarker);
        }

        /// <summary>
        /// The EndMarker method pops the last added entry off of the marker
        /// start and uses it, along with the current line and character
        /// position, to determine the section of code within the source code
        /// that is used as the mark sequence point in the currently emitting
        /// IL Generator.
        /// </summary>
        /// <param name="ilGenerator">
        /// The ilGenerator parameter is used to emit a MarkSequencePoint,
        /// adding debug information to the code.
        /// </param>
        public void EndMarker(ILGenerator ilGenerator)
        {
            // We only carry out this functionality if the new code is
            // debuggable.
            if (!debuggable) return;

            // Pop the last added marker form the stack, this is our start
            // point within the source code.
            DebugCodeMarker startMarker = markers.Pop();

            // Emit a MarkSequencePoint within the code to tie the segment of
            // source code to the emitted il code.
            ilGenerator.MarkSequencePoint(symbolDocument, startMarker.Line,
startMarker.CharPos, lineCount, charPos);

            // By emitting a Nop instruction we are ensuring that out programs
            // execution with stop at the marked sequenced point, even if no
            // real functionality is taken place.
            ilGenerator.Emit(OpCodes.Nop);
        }

        /// <summary>
        /// The SetVariableName method is used to assign each variable to
        /// be viewed in the variable list with a unique name.
        /// </summary>
        /// <param name="builder">
        /// The builder parameter represents the LocalBuilder to which we
        /// are assigning a variable name.
        /// </param>
        /// <param name="name">
        /// The name parameter provides a string which describes the
        /// name to which a unique number will be attached to
        /// form the variable name.
        /// </param>
        public void SetVariableName(LocalBuilder builder, string name)
        {
            // We only carry out this functionality if the new code is
            // debuggable.
            if (!debuggable) return;

            // If the name exists in the sorted list, get the value the
            // is currently being held.
            int num = 0;
            if (varCounts.ContainsKey(name))
                num = varCounts[name];
```

```
            // Increment the unique number and save it back to the array
            // for potential use later.
            num++;
            varCounts[name] = num;

            // Set the debug data so that the variable name is the name
            // provided with the unique number appended to the end.
            builder.SetLocalSymInfo(name + num.ToString());
        }

        /// <summary>
        /// The DebuggingSupport constructor takes the AssemblyBuilder and
        /// ModuleBuilder and sets them up to incldue all the necessary
        /// configuration to support debugging, if debugging should be
        /// enabled.
        /// </summary>
        /// <param name="builder">
        /// The builder parameter provides the AssemblyBuilder object to be
        /// configured for debugging.
        /// </param>
        /// <param name="module">
        /// The module parameter provides the ModuleBuilder object to be
        /// configured for debugging.
        /// </param>
        public DebugSupport(AssemblyBuilder builder, ModuleBuilder module)
        {
            // Store in our local variable whether or not we should produce
            // any debug information.
            debuggable = IsDebuggable;

            // We only carry out the rest of this functionality if the new
            // code is debuggable.
            if (!debuggable)
                return;

            // Create ourselves the stack for the markers.
            markers = new Stack<DebugCodeMarker>();

            // Create ourselves the array of variable numbers so we don't
            // duplicate within the code.
            varCounts = new SortedList<string, int>();

            // Determine the sourceFile filename for use by the filestream.
            sourceFile = System.IO.Path.Combine(Environment.CurrentDirectory,
"Rules.xml");

            // Open our source file output stream.
            sourceWriter = new StreamWriter(sourceFile);

            // If we are debuggable, add the debuggable attribute to our
            // new assembly.
            builder.SetCustomAttribute(new
CustomAttributeBuilder(typeof(DebuggableAttribute).GetConstructor(new Type[] {
typeof(DebuggableAttribute.DebuggingModes) }), new object[] {
DebuggableAttribute.DebuggingModes.DisableOptimizations |
DebuggableAttribute.DebuggingModes.Default }));

            // If we are debuggable, we need to define a symbol document
writer.
            symbolDocument = module.DefineDocument(sourceFile, Guid.Empty,
Guid.Empty, Guid.Empty);
        }

        /// <summary>
        /// The WriteSource method is used to write a line of source code
        /// to the source file. It does not emit any mark points to the
        /// debug information, just emits text to the source file.
        /// </summary>
        /// <param name="source">
        /// The source parameter provides the source code to write to the
        /// file.
        /// </param>
        public void WriteSource(string source)
        {
            // We only carry out this functionality if the new code is
```

```
            // debuggable.
            if (!debuggable) return;

            // Count the number of lines within the text be output.
            lineCount += source.Split('\n').Length - 1;

            // If the source includes end brackets we'll reduce the
            // indention.
            int deTabs = source.Split(new string[] { @"</" },
StringSplitOptions.None).Length + source.Split(new string[] { @"/>" },
StringSplitOptions.None).Length - 2;
            tabCount -= deTabs;

            // Prepend the tabs onto the start of the source code.
            source = (new string('\t', tabCount)) + source;

            // Does this source code have a new line, if so, we need
            // to determine the new charPos based on the last line
            // within the source, other we just increment the current
            // charPos.
            if (source.LastIndexOf('\n') > 0)
                charPos = source.Length - source.LastIndexOf('\n');
            else
                charPos += source.Length;

            // Write the new line of source to the sourceWriter.
            sourceWriter.Write(source);

            // If the source includes start brackets we'll increase the
            // indention.
            int tabs = (source.Split(new string[] { @"<" },
StringSplitOptions.None).Length - 1) - (source.Split(new string[] { @"</" },
StringSplitOptions.None).Length - 1);
            tabCount += tabs;
        }

        /// <summary>
        /// The WriteSourceLine method is used to write a line of source
        /// code to the source file, appending a newline to the end. It
        /// does not emit any mark points to the debug information, just
        /// emits text to the source file.
        /// </summary>
        /// <param name="source">
        /// The source parameter provides the source code to write to the
        /// file.
        /// </param>
        public void WriteSourceLine(string source)
        {
            // We only carry out this functionality if the new code is
            // debuggable.
            if (!debuggable) return;

            // Using the WriteSource method to output the given source
            // code, appending a newline character to it.
            WriteSource(source + "\r\n");
        }


        /// <summary>
        /// The WriteSource method is used to write a line of source code
        /// to the source file. This method will also emit debug information
        /// to surround the source.
        /// </summary>
        /// <param name="source">
        /// The source parameter provides the source code to write to the
        /// file.
        /// </param>
        /// <param name="ilGenerator">
        /// The ilGenerator parameter provides the ILGenerator instance to
        /// mark with the given piece of source.
        /// </param>
        public void WriteSourceAndMark(string source, ILGenerator ilGenerator)
        {
            // We only carry out this functionality if the new code is
            // debuggable.
            if (!debuggable) return;
```

```
            // First we set a StartMarker, them we WriteSource, then
            // finally an EndMarker.
            StartMarker();
            WriteSource(source);
            EndMarker(ilGenerator);
        }

        /// <summary>
        /// The WriteSourceLine method is used to write a line of source
        /// code to the source file, appending a newline to the end. This
        /// method will also emit debug information to surround the source.
        /// </summary>
        /// <param name="source">
        /// The source parameter provides the source code to write to the
        /// file.
        /// </param>
        /// <param name="ilGenerator">
        /// The ilGenerator parameter provides the ILGenerator instance to
        /// mark with the given piece of source.
        /// </param>
        public void WriteSourceLineAndMark(string source, ILGenerator
ilGenerator)
        {
            // We only carry out this functionality if the new code is
            // debuggable.
            if (!debuggable) return;

            // First we set a StartMarker, them we WriteSourceLine, then
            // finally an EndMarker.
            StartMarker();
            WriteSourceLine(source);
            EndMarker(ilGenerator);
        }

        /// <summary>
        /// This dispose method will clean up any open file streams and
        /// clear any arrays ready for garbage collection.
        /// </summary>
        public void Dispose()
        {
            // Only clear up the markers if the object has been created.
            if (markers != null)
            {
                markers.Clear();
                markers = null;
            }

            // Only clear up the sourceWriter if the object has been
            // created.
            if (sourceWriter != null)
            {
                sourceWriter.Close();
                sourceWriter.Dispose();
                sourceWriter = null;
            }
        }

    }
}
```

11.3.1.3 RuleCompilerException Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

namespace RuleCompiler
{

    /// <summary>
    /// The RuleCompilerException class is used to describe a
```

```
    /// throwable exception that is generation which there are
    /// problems found during the compilation process.
    /// </summary>
    public class RuleCompilerException : Exception
    {
        public RuleCompilerException(String Message)
            : base(Message)
        {
        }
    }

}
```

11.3.1.4 TypeMap Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use the these libraries for providing dynamically sizable collections.
using System.Collections.Generic;

// Use the RuleDefinitionLanguage library, PrePost, as a means of storage.
using RuleDefinitionLanguage;

/// <summary>
/// The RuleCompiler namespace contains all of the functionality
/// required to support the compilation of the rules in the rule
/// definition language to executable code.
/// </summary>
namespace RuleCompiler
{

    /// <summary>
    /// The TypeMap class provides a means of storing
    /// a complete list of the PrePost's, type names contained
    /// within or that can contain Types. This is used after the
    /// main compilation to do a check to ensure all loosely
    /// defined types, which are referenced, exist, once compiled.
    /// </summary>
    public class TypeMap
    {

        public String Name;
        public List<PrePost> PrePost = new List<PrePost>();
        public List<string> Contains = new List<string>();
        public List<string> ContainWithin = new List<string>();

        public TypeMap(String name)
        {
            Name = name;
        }

    }

}
```

11.3.2 RuleDefinitionLanguage Namespace

11.3.2.1 AndCondition Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use these libraries for enabling Reflection of the classes into
// executable code.
using System.Reflection.Emit;

/// <summary>
/// RuleDefinitionLanguage Namespace.
/// This namespace defines core types used to help define a
/// set of rules. These types are not used for execution but for
/// description of the rules.
/// </summary>
namespace RuleDefinitionLanguage
{

    /// <summary>
    /// AndCondition Class. Used to an And condition which
    /// devides into different parts.
    /// </summary>
    [Serializable()]
    public class AndCondition : Comparitor
    {

        private Comparitor[] lConditions;

        /// <summary>
        /// The Conditions member contains the various Comparitor which are
        /// to be And-ed together to determine a positive or negative
        /// result.
        /// </summary>
        /// <exception cref="System.ArgumentNullException"></exception>
        public Comparitor[] Conditions
        {
            get
            {
                return lConditions;
            }
            set
            {
                if (value == null)
                    throw new ArgumentNullException();
                else
                    lConditions = value;
            }
        }

        /// <summary>
        /// The default constructor for this class allocates and empty
        /// array of conditions for the AndCondition which, if tested, would
        /// result in a True statement.
        /// </summary>
        public AndCondition()
        {
            Conditions = new Comparitor[0];
        }

        /// <summary>
        /// This constructor allows the allocation of the conditions to be
        /// set on initialisation of the class.
        /// </summary>
        /// <param name="conditions">
        /// The Conditions member contains the various Comparitor which are
        /// to be And-ed together to determine a positive or negative
        /// result.
        /// </param>
        public AndCondition(Comparitor[] conditions)
```

```
        {
            Conditions = conditions;
        }

        /// <summary>
        /// The Compile method for the And comparision will emit the
        /// appropriate IL to combine to boolean conditions.
        /// </summary>
        /// <param name="retVal">
        /// The retVal parameter is the local variable passed in by the
        /// caller to be used to store the result of this And boolean
        /// comparison.
        /// </param>
        /// <param name="ilGenerator">
        /// The ilGenerator parameter is used to perform the emitting of
        /// the necessary And comparison code.
        /// </param>
        /// <param name="debugSupport">
        /// The debugSupport is used to allow the compilation process
        /// emit debug information.
        /// </param>
        public override void Compile(LocalBuilder retVal, ILGenerator
ilGenerator, RuleCompiler.DebugSupport debugSupport)
        {
            debugSupport.WriteSourceLineAndMark("<And>", ilGenerator);

            LocalBuilder[] retVals = new LocalBuilder[Conditions.Length];

            for (int i = 0; i < Conditions.Length; i++)
            {
                retVals[i] = ilGenerator.DeclareLocal(typeof(bool));
                debugSupport.SetVariableName(retVals[i], "And");
                ilGenerator.BeginScope();
                Conditions[i].Compile(retVals[i], ilGenerator, debugSupport);
                ilGenerator.EndScope();
            }

            Label endLabel = ilGenerator.DefineLabel();
            ilGenerator.Emit(OpCodes.Ldc_I4_0);
            ilGenerator.Emit(OpCodes.Stloc_S, retVal);

            for (int ii = 1; ii < retVals.Length; ii++)
            {
                ilGenerator.Emit(OpCodes.Ldloc_S, retVals[ii - 1]);
                ilGenerator.Emit(OpCodes.Ldloc_S, retVals[ii]);
                Label nextLabel = ilGenerator.DefineLabel();
                ilGenerator.Emit(OpCodes.Beq_S, nextLabel);
                ilGenerator.Emit(OpCodes.Br_S, endLabel);
                ilGenerator.MarkLabel(nextLabel);
            }

            ilGenerator.Emit(OpCodes.Ldc_I4_1);
            ilGenerator.Emit(OpCodes.Stloc_S, retVal);
            ilGenerator.MarkLabel(endLabel);

            debugSupport.WriteSourceLineAndMark("</And>", ilGenerator);
        }

    }

}
```

11.3.2.2 Comparison Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use these libraries for enabling Reflection of the classes into
// executable code.
using System.Reflection.Emit;
```

```
/// <summary>
/// RuleDefinitionLanguage Namespace.
/// This namespace defines core types used to help define a
/// set of rules. These types are not used for execution but for
/// description of the rules.
/// </summary>
namespace RuleDefinitionLanguage
{

    /// <summary>
    /// The ComparisonOperation enum describes the type of
    /// comparison to make on two true or false values.
    /// </summary>
    public enum ComparisonOperation
    {
        /// <summary>
        /// Are the two values equal?
        /// </summary>
        Equals,

        /// <summary>
        /// Are the two values not equal?
        /// </summary>
        NotEquals,

        /// <summary>
        /// Is the first value less than the second? Can only be
        ///  applied to values of a numeric nature.
        /// </summary>
        LessThan,

        /// <summary>
        /// Is the first value less than or equal to the second?
        /// Can only be applied to values of a numeric nature.
        /// </summary>
        LessThanOrEqualTo,

        /// <summary>
        /// Is the first value greater than the second? Can only
        /// be applied to values of a numeric nature.
        /// </summary>
        GreaterThan,

        /// <summary>
        /// Is the first value greater or equal to the second?
        /// Can only be applied to values of a numeric nature.
        /// </summary>
        GreaterThanOrEqualTo
    };

    /// <summary>
    /// Comparison Class. Used to compare two values.
    /// </summary>
    [Serializable()]
    public class Comparison : Comparitor
    {
        private ComparisonOperation lOperation;
        private Value[] lValues;
        private Comparitor lCondition;

        /// <summary>
        /// Operation describe the type of comparison to make.
        /// </summary>
        public ComparisonOperation Operation
        {
            get
            {
                return lOperation;
            }
            set
            {
                lOperation = value;
            }
        }

        /// <summary>
```

```
        /// Values contain the different values to make comparisons with.
        /// The different values must be of the same ValueType to have a
        /// chance at returning a True value.
        /// </summary>
        /// <exception cref="System.ArgumentNullException"></exception>
        public Value[] Values
        {
            get
            {
                return lValues;
            }
            set
            {
                if (value == null)
                    throw new ArgumentNullException();
                else
                    lValues = value;
            }
        }

        /// <summary>
        /// Condition describes any additional condition which must be
        /// satisfied in order to return a True value, such as requiring
        /// compensation. This member can be null.
        /// </summary>
        public Comparitor Condition
        {
            get
            {
                return lCondition;
            }
            set
            {
                lCondition = value;
            }
        }

        /// <summary>
        /// The default constructor allocates initial values. The default
        /// comparison is set to Equals with no Values or Conditions which
        /// will return True unless changed.
        /// </summary>
        public Comparison()
        {
            Operation = ComparisonOperation.Equals;
            Values = new Value[0];
            //Condition = new Comparitor();
        }

        /// <summary>
        /// This constructor allows the member variables to be initialised
        /// at construction.
        /// </summary>
        /// <param name="operation">
        /// operation describe the type of comparison to make.
        /// </param>
        /// <param name="values">
        /// values contain the different values to make comparisons with.
        /// The different values must be of the same ValueType to have a
        /// chance at returning a True value.
        /// </param>
        /// <param name="condition">
        /// condition describes any additional condition which must be
        /// satisfied in order to return a True value, such as requiring
        /// compensation. This parameter can be null.
        /// </param>
        public Comparison(ComparisonOperation operation, Value[] values,
Comparitor condition)
        {
            Operation = operation;
            Values = values;
            Condition = condition;
        }

        /// <summary>
        /// The Compile method of the Comparison class is used to Emit the
```

```
        /// IL for comparing two values to each other and return the result.
        /// </summary>
        /// <param name="retVal">
        /// The retVal parameter is passed in by the caller to be used by
        /// the Emitted code to set the return value to.
        /// </param>
        /// <param name="ilGenerator">
        /// The ilGenerator parameter is the object used to Emit the code
        /// and would have been used previously by the caller.
        /// </param>
        /// <param name="debugSupport">
        /// The debugSupport is used to allow the compilation process
        /// emit debug information.
        /// </param>
        public override void Compile(LocalBuilder retVal, ILGenerator
ilGenerator, RuleCompiler.DebugSupport debugSupport)
        {
            debugSupport.WriteSourceLineAndMark("<Comparison Operation='" +
Operation.ToString() + "'>", ilGenerator);

            LocalBuilder[] retVals = new LocalBuilder[Values.Length];

            for (int i = 0; i < Values.Length; i++)
            {
                ilGenerator.BeginScope();
                Values[i].Compile(ilGenerator, ref retVals[i], debugSupport);
                ilGenerator.EndScope();
            }

            Label endLabel = ilGenerator.DefineLabel();
            ilGenerator.Emit(OpCodes.Ldc_I4_0);
            ilGenerator.Emit(OpCodes.Stloc_S, retVal);
            ilGenerator.Emit(OpCodes.Ldloc, retVals[0]);
            for (int ii = 1; ii < retVals.Length; ii++)
            {

                ilGenerator.Emit(OpCodes.Ldloc, retVals[ii]);
                Label nextLabel;

                switch (Operation)
                {

                    case ComparisonOperation.Equals:

                        nextLabel = ilGenerator.DefineLabel();
                        ilGenerator.Emit(OpCodes.Beq_S, nextLabel);
                        ilGenerator.Emit(OpCodes.Br_S, endLabel);
                        ilGenerator.MarkLabel(nextLabel);
                        break;

                    case ComparisonOperation.NotEquals:

                        ilGenerator.Emit(OpCodes.Beq_S, endLabel);
                        break;

                    case ComparisonOperation.LessThan:

                        ilGenerator.Emit(OpCodes.Bge_S, endLabel);
                        break;

                    case ComparisonOperation.LessThanOrEqualTo:

                        ilGenerator.Emit(OpCodes.Bgt_S, endLabel);
                        break;

                    case ComparisonOperation.GreaterThan:

                        ilGenerator.Emit(OpCodes.Ble_S, endLabel);
                        break;

                    case ComparisonOperation.GreaterThanOrEqualTo:

                        ilGenerator.Emit(OpCodes.Blt_S, endLabel);
                        break;

                }
```

```
                ilGenerator.Emit(OpCodes.Ldc_I4_1);
                ilGenerator.Emit(OpCodes.Stloc_S, retVal);
                ilGenerator.MarkLabel(endLabel);

            }

            debugSupport.WriteSourceLineAndMark("</Comparison>", ilGenerator);
        }

    }

}
```

11.3.2.3 Comparitor Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use this library to provide a means of serialising the objects into
// XML for file storage and network transportation.
using System.Xml.Serialization;

// Use these libraries for enabling Reflection of the classes into
// executable code.
using System.Reflection.Emit;

/// <summary>
/// RuleDefinitionLanguage Namespace.
/// This namespace defines core types used to help define a
/// set of rules. These types are not used for execution but for
/// description of the rules.
/// </summary>
namespace RuleDefinitionLanguage
{

    /// <summary>
    /// Comparitor Class. Used to represent an operation which will
    /// result in either a true or false answer.
    /// </summary>
    [Serializable(), XmlInclude(typeof(AndCondition)),
XmlInclude(typeof(OrCondition)), XmlInclude(typeof(Comparison))]
    public abstract class Comparitor
    {
        /// <summary>
        /// This method, when consumed by a parent class, is used for
        /// the parent to emit its IL.
        /// </summary>
        /// <param name="retVal">
        /// The retVal is the boolean value where the operation will
        /// store the result of the comparison.
        /// </param>
        /// <param name="ilGenerator">
        /// The ilGenerator is to be used for emitting the IL to.
        /// </param>
        /// <param name="debugSupport">
        /// The debugSupport is used to allow the compilation process
        /// emit debug information.
        /// </param>
        public abstract void Compile(LocalBuilder retVal, ILGenerator
ilGenerator, RuleCompiler.DebugSupport debugSupport);
    }

}
```

11.3.2.4 Contain Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

/// <summary>
/// RuleDefinitionLanguage Namespace.
/// This namespace defines core types used to help define a
/// set of rules. These types are not used for execution but for
/// description of the rules.
/// </summary>
namespace RuleDefinitionLanguage
{

    /// <summary>
    /// Content Class. Used to define the type of contents that
    /// a period may have, imply multiple X's make up a Y.
    /// </summary>
    [Serializable()]
    public class Contain
    {

        private String lName;

        /// <summary>
        /// The Name describes the name of a period which can have multiple
        /// instances contained with the period.
        /// </summary>
        /// <exception cref="System.ArgumentNullException"></exception>
        public String Name
        {
            get
            {
                return lName;
            }
            set
            {
                if (value == null)
                    throw new ArgumentNullException();
                else
                    lName = value;
            }
        }

        /// <summary>
        /// The default constructor for this class simply assigns default
        /// values to the member variables.
        /// </summary>
        public Contain()
        {
            Name = "";
        }

        /// <summary>
        /// This constructor for this class has parameters to assign the
        /// start values of the member variables.
        /// </summary>
        /// <param name="name">
        /// The Name describes the name of a period which can have multiple
        /// instances contained with the period.
        /// </param>
        public Contain(String name)
        {
            Name = name;
        }
    }

}
```

11.3.2.5 Math Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use this library to provide a means of serialising the objects into
// XML for file storage and network transportation.
using System.Xml.Serialization;

// Use these libraries for enabling Reflection of the classes into
// executable code.
using System.Reflection.Emit;

/// <summary>
/// RuleDefinitionLanguage Namespace.
/// This namespace defines core types used to help define a
/// set of rules. These types are not used for execution but for
/// description of the rules.
/// </summary>
namespace RuleDefinitionLanguage
{

    /// <summary>
    /// The MathOperation enum describe a type of mathematical operation
    /// which can be used with the Math class.
    /// </summary>
    public enum MathOperation
    {
        /// <summary>
        /// The operation of adding two numbers together.
        /// </summary>
        Add,

        /// <summary>
        /// The operation of subtracting two numbers.
        /// </summary>
        Subtract,

        /// <summary>
        /// The operation of multiplying two numbers together.
        /// </summary>
        Multiply,

        /// <summary>
        /// The operation of dividing two numbers.
        /// </summary>
        Divide
    };

    /// <summary>
    /// Math Class. Used to perform math operations on a set
    /// of values.
    /// </summary>
    [Serializable()]
    public class Math : Value
    {
        private MathOperation lOperation;
        private Value[] lValues;

        /// <summary>
        /// Operation describes the type of math function to be carried out
        /// on the given Values.
        /// </summary>
        public MathOperation Operation
        {
            get
            {
                return lOperation;
            }
            set
            {
                lOperation = value;
            }
        }
```

```
        /// <summary>
        /// The Values are the given values to have the mathematical
        /// functions performed on.
        /// </summary>
        /// <exception cref="System.ArgumentNullException"></exception>
        public Value[] Values
        {
            get
            {
                return lValues;
            }
            set
            {
                if (value == null)
                    throw new ArgumentNullException();
                else
                    lValues = value;
            }
        }

        /// <summary>
        /// The description variable overriden from the base class to hide
        /// it as it is no longer required at this level.
        /// </summary>
#pragma warning disable 0169
        [XmlIgnore()]
        private new String Description;
#pragma warning restore 0169

        /// <summary>
        /// The default constructor intializing the Math object to an Add
        /// operation with no Values, which should return a zero values as
        /// a result.
        /// </summary>
        public Math()
        {
            Operation = MathOperation.Add;
            Values = new Value[0];
        }

        /// <summary>
        /// This constructor allows for the members to be initialised to
        /// the given values passed in as paramters.
        /// </summary>
        /// <param name="operation">
        /// operation describes the type of math function to be carried out
        /// on the given Values.
        /// </param>
        /// <param name="values">
        /// The values are the given values to have the mathematical
        /// functions performed on.
        /// </param>
        public Math(MathOperation operation, Value[] values)
        {
            Operation = operation;
            Values = values;
        }

        /// <summary>
        /// The Compile method of the Math class is used to Emit the
        /// IL for performing operations on various values.
        /// </summary>
        /// <param name="ilGenerator">
        /// The ilGenerator parameter is the object used to Emit the code
        /// and would have been used previously by the caller.
        /// </param>
        /// <param name="newLocal">
        /// The newLocal parameter is used to refer to the variable
        /// that stores the result of the Math operation;
        /// </param>
        /// <param name="debugSupport">
        /// The debugSupport is used to allow the compilation process
        /// emit debug information.
        /// </param>
        public override void Compile(ILGenerator ilGenerator, ref LocalBuilder
newLocal, RuleCompiler.DebugSupport debugSupport)
```

```
        {
            debugSupport.WriteSourceLineAndMark("<Math>", ilGenerator);
            newLocal = ilGenerator.DeclareLocal(typeof(double));
            debugSupport.SetVariableName(newLocal, "Math");

            ilGenerator.BeginScope();

            // Start by generating all of the local variables which will
            // be used.
            LocalBuilder[] valueLocals = new LocalBuilder[Values.Length];
            for (int i = 0; i < valueLocals.Length; i++)
            {
                Values[i].Compile(ilGenerator, ref valueLocals[i],
debugSupport);
            }

            // If you are adding numbers, you can start at 0.0,
            // but if you are subtracting, of course it wont work!
            debugSupport.WriteSourceLineAndMark("<Operation type='" +
Operation.ToString() + "' />", ilGenerator);
            ilGenerator.Emit(OpCodes.Ldloc_S, valueLocals[0]);

            for (int ii = 1; ii < valueLocals.Length; ii++)
            {
                ilGenerator.Emit(OpCodes.Ldloc_S, valueLocals[ii]);

                switch (Operation)
                {
                    case MathOperation.Add:
                        ilGenerator.Emit(OpCodes.Add);
                        break;
                    case MathOperation.Subtract:
                        ilGenerator.Emit(OpCodes.Sub);
                        break;
                    case MathOperation.Multiply:
                        ilGenerator.Emit(OpCodes.Mul);
                        break;
                    case MathOperation.Divide:
                        ilGenerator.Emit(OpCodes.Div);
                        break;
                }
            }

            // Store the result of the math operation in our new local
            // variable.
            ilGenerator.Emit(OpCodes.Stloc S, newLocal);

            ilGenerator.EndScope();

            debugSupport.WriteSourceLineAndMark("</Math>", ilGenerator);
        }

    }

}
```

11.3.2.6 OrCondition Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use these libraries for enabling Reflection of the classes into
// executable code.
using System.Reflection.Emit;

/// <summary>
/// RuleDefinitionLanguage Namespace.
/// This namespace defines core types used to help define a
/// set of rules. These types are not used for execution but for
/// description of the rules.
/// </summary>
```

```
namespace RuleDefinitionLanguage
{

    /// <summary>
    /// OrCondition Class. Used to an Or condition which
    /// devides into different parts.
    /// </summary>
    [Serializable()]
    public class OrCondition : Comparitor
    {
        private Comparitor[] lConditions;

        /// <summary>
        /// The Conditions member contains the various Comparitor which are
        /// to be Or-ed together to determine a positive or negative
        /// result.
        /// </summary>
        /// <exception cref="System.ArgumentNullException"></exception>
        public Comparitor[] Conditions
        {
            get
            {
                return lConditions;
            }
            set
            {
                if (value == null)
                    throw new ArgumentNullException();
                else
                    lConditions = value;
            }
        }

        /// <summary>
        /// The default constructor for this class allocates and empty
        /// array of conditions for the AndCondition which, if tested, would
        /// result in a True statement.
        /// </summary>
        public OrCondition()
        {
            Conditions = new Comparitor[0];
        }

        /// <summary>
        /// This constructor allows the allocation of the conditions to be
        /// set on initialisation of the class.
        /// </summary>
        /// <param name="conditions">
        /// The Conditions member contains the various Comparitor which are
        /// to be Or-ed together to determine a positive or negative
        /// result.
        /// </param>
        public OrCondition(Comparitor[] conditions)
        {
            Conditions = conditions;
        }

        /// <summary>
        /// The Compile method for the Or comparision will emit the
        /// appropriate IL to combine to boolean conditions.
        /// </summary>
        /// <param name="retVal">
        /// The retVal parameter is the local variable passed in by the
        /// caller to be used to store the result of this Or boolean
        /// comparison.
        /// </param>
        /// <param name="ilGenerator">
        /// The ilGenerator parameter is used to perform the emitting of
        /// the necessary Or comparison code.
        /// </param>
        /// <param name="debugSupport">
        /// The debugSupport is used to allow the compilation process
        /// emit debug information.
        /// </param>
        public override void Compile(LocalBuilder retVal, ILGenerator
ilGenerator, RuleCompiler.DebugSupport debugSupport)
```

```
        {
            debugSupport.WriteSourceLineAndMark("<Or>", ilGenerator);

            LocalBuilder[] retVals = new LocalBuilder[Conditions.Length];

            for (int i = 0; i < Conditions.Length; i++)
            {
                retVals[i] = ilGenerator.DeclareLocal(typeof(bool));
                ilGenerator.BeginScope();
                Conditions[i].Compile(retVals[i], ilGenerator, debugSupport);
                ilGenerator.EndScope();
            }

            Label endLabel = ilGenerator.DefineLabel();
            ilGenerator.Emit(OpCodes.Ldc_I4_0);
            ilGenerator.Emit(OpCodes.Stloc_S, retVal);
            Label trueLabel = ilGenerator.DefineLabel();

            for (int ii = 0; ii < retVals.Length; ii++)
            {
                ilGenerator.Emit(OpCodes.Ldc_I4_1);
                ilGenerator.Emit(OpCodes.Ldloc, retVals[ii]);
                ilGenerator.Emit(OpCodes.Beq_S, trueLabel);
            }
            ilGenerator.Emit(OpCodes.Br_S, endLabel);

            ilGenerator.MarkLabel(trueLabel);
            ilGenerator.Emit(OpCodes.Ldc_I4_1);
            ilGenerator.Emit(OpCodes.Stloc_S, retVal);

            ilGenerator.MarkLabel(endLabel);

            debugSupport.WriteSourceLineAndMark("</Or>", ilGenerator);
        }

    }

}
```

11.3.2.7 Period Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use this library to provide a means of serialising the objects into
// XML for file storage and network transportation.
using System.Xml.Serialization;

// Use these libraries for enabling Reflection of the classes into
// executable code.
using System.Reflection;
using System.Reflection.Emit;

// Use the RuleCompiler namespace to gain access to the TypeMap class.
using RuleCompiler;

/// <summary>
/// RuleDefinitionLanguage Namespace.
/// This namespace defines core types used to help define a
/// set of rules. These types are not used for execution but for
/// description of the rules.
/// </summary>
namespace RuleDefinitionLanguage
{

    /// <summary>
    /// Period Class. Used to define a period.
    /// </summary>
    [Serializable(), XmlInclude(typeof(WP)), XmlInclude(typeof(RP)),
XmlInclude(typeof(AndCondition)), XmlInclude(typeof(Comparison)),
XmlInclude(typeof(Comparitor)), XmlInclude(typeof(Contain)),
```

```
XmlInclude(typeof(Math)), XmlInclude(typeof(OrCondition)),
XmlInclude(typeof(PeriodDescription)), XmlInclude(typeof(PrePost)),
XmlInclude(typeof(Value))]
    public class Period
    {

        private PeriodDescription lDescription;
        private Comparitor lRules;

        /// <summary>
        /// Description is an instance of PeriodDescription used to describe
        /// general information about the period such as its name and other
        /// personalised pieces of information.
        /// </summary>
        /// <exception cref="System.ArgumentNullException"></exception>
        public PeriodDescription Description
        {
            get
            {
                return lDescription;
            }
            set
            {
                if (value == null)
                    throw new ArgumentNullException();
                else
                    lDescription = value;
            }
        }

        /// <summary>
        /// Rules is an instance of a Comparitor which can be any of the
        /// inherited classes of type Comparitor used to define a set of
        /// rules which must evaluate to true in order to meet their
        /// conditions.
        /// </summary>
        /// <exception cref="System.ArgumentNullException"></exception>
        public Comparitor Rules
        {
            get
            {
                return lRules;
            }
            set
            {
                if (value == null)
                    throw new ArgumentNullException();
                else
                    lRules = value;
            }
        }

        /// <summary>
        /// Period Constructor is used for defining an instance of the
        /// Period class without specifying any of the details up front.
        /// </summary>
        public Period()
        {
            Description = new PeriodDescription();
            //Rules = new Comparitor();
        }

        /// <summary>
        /// Period Constructor is used for defining an instance of the
        /// Period class including the details which make this period
        /// instance unique from the others.
        /// </summary>
        /// <param name="description">
        /// description is an instance of PeriodDescription used to describe
        /// general information about the period such as its name and other
        /// personalised pieces of information.
        /// </param>
        /// <param name="rules">
        /// rules is an instance of a Comparitor which can be any of the
        /// inherited classes of type Comparitor used to define a set of
        /// rules which must evaluate to true in order to meet their
```

```
        /// conditions.
        /// </param>
        public Period(PeriodDescription description, Comparitor rules)
        {
            Description = description;
            Rules = rules;
        }

        /// <summary>
        /// The Compile method is a general implementation of the compile
        /// method used within more specific implementing classes such as
        /// WP and RP. They would typically call this base class method
        /// in order to compile the rules.
        /// </summary>
        /// <param name="moduleBuilder">
        /// The moduleBuilder parameter is the module into which the new
        /// type is created and can be then consumed.
        /// </param>
        /// <param name="typeMap">
        /// The typeMap variable provides a means of describing how the
        /// variables types fit together.
        /// </param>
        /// <param name="baseType">
        /// The baseType parameter is the Type of object which will be the
        /// base class for the new Type. Each base Type such as WP or RP
        /// has slightly different implementations to support that
        /// particular type of rule.
        /// </param>
        /// <param name="debugSupport">
        /// The debugSupport is used to allow the compilation process
        /// emit debug information.
        /// </param>
        /// <returns>
        /// The return of the method is a TypeBuilder object describing
        /// the new Type. Before the new type can be used, it must be
        /// creating using the TypeBuilder's CreateType method.
        /// </returns>
        public Type Compile(ModuleBuilder moduleBuilder, TypeMap typeMap, Type
baseType, DebugSupport debugSupport)
        {

            // Define the variables used within this method.
            TypeBuilder newType = null;
            ILGenerator ilGenerator = null;
            ConstructorBuilder constructor = null;
            MethodBuilder testMethod = null;

            // Define the new class for this particular period type basing
            // it on the base class provided.
            newType = moduleBuilder.DefineType("Rules." + Description.Name,
TypeAttributes.Class | TypeAttributes.Public, baseType);

            // Define the constructor for the new object and emit the code
            // to get the constructor to call the base classes constructor
            // which should initialise everything properly. Also get the
            // new types Description object to inject its intial setup
            // values inserted into the constructor.
            constructor = newType.DefineConstructor(MethodAttributes.Public,
CallingConventions.Standard, new Type[0]);
            ilGenerator = constructor.GetILGenerator();
            debugSupport.WriteSourceLineAndMark("<" + Description.Name + ">",
ilGenerator);
            ilGenerator.Emit(OpCodes.Ldarg 0);
            ilGenerator.Emit(OpCodes.Call, baseType.GetConstructor(new Type[] {
}));
            Description.Compile(baseType, newType, typeMap, ilGenerator);
            ilGenerator.Emit(OpCodes.Ret);

            // Now define the Test method which returns a boolean value,
            // true meaning that the test was successful, false it is was
            // not.
            testMethod = newType.DefineMethod("Test", MethodAttributes.Public |
MethodAttributes.Virtual, CallingConventions.Standard, typeof(bool), new
Type[0]);

            // In order to generate the necessary IL code for this method,
```

```
            // we get the rules themselves to inject their sections of the
            // code into. Each section of the rule will normally inject
            // its code into seperate scopes to keep the different sections
            // seperated.
            ilGenerator = testMethod.GetILGenerator();
            debugSupport.WriteSourceLineAndMark("<Rules>", ilGenerator);
            LocalBuilder retVal = ilGenerator.DeclareLocal(typeof(bool));
            debugSupport.SetVariableName(retVal, "Result");
            ilGenerator.BeginScope();
            Rules.Compile(retVal, ilGenerator, debugSupport);
            ilGenerator.EndScope();

            debugSupport.WriteSourceLineAndMark("</Rules>", ilGenerator);
            ilGenerator.Emit(OpCodes.Ldloc S, retVal);

            debugSupport.WriteSourceLineAndMark("</" + Description.Name + ">",
ilGenerator);
            ilGenerator.Emit(OpCodes.Ret);

            // Return the newly created type to the calling object.
            try
            {
                return newType.CreateType();
            }
            catch (Exception e)
            {
                throw new RuleCompiler.RuleCompilerException("CreateType
Failed: " + e.ToString());
            }
        }

        /// <summary>
        /// As this is the base class for an 'normal' period type, it
        /// defines a typical Compile method which any overriding class
        /// would need to implement.
        /// </summary>
        /// <param name="moduleBuilder">
        /// The moduleBuilder parameter defines the module into which the
        /// new type will be placed.
        /// </param>
        /// <param name="typeMap">
        /// The typeMap variable provides a means of describing how the
        /// variables types fit together.
        /// </param>
        /// <param name="debugSupport">
        /// The debugSupport is used to allow the compilation process
        /// emit debug information.
        /// </param>
        /// <returns>
        /// The return value of this compile method is the TypeBuilder for
        /// the newly created type.
        /// </returns>
        public virtual Type Compile(ModuleBuilder moduleBuilder, TypeMap
typeMap, DebugSupport debugSupport)
        {
            return Compile(moduleBuilder, typeMap, typeof(Object),
debugSupport);
        }

    }

}
```

11.3.2.8 PeriodDescription Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use the these libraries for providing dynamically sizable collections.
using System.Collections.Generic;
```

```
// Use these libraries for enabling Reflection of the classes into
// executable code.
using System.Reflection;
using System.Reflection.Emit;

// Use the RuleCompiler namespace to gain access to the TypeMap class.
using RuleCompiler;

/// <summary>
/// RuleDefinitionLanguage Namespace.
/// This namespace defines core types used to help define a
/// set of rules. These types are not used for execution but for
/// description of the rules.
/// </summary>
namespace RuleDefinitionLanguage
{

    /// <summary>
    /// PeriodDescription Class. Used to define details about a type
    /// of period.
    /// </summary>
    [Serializable()]
    public class PeriodDescription
    {

        private String lName;
        private String lDescription;
        private PrePost[] lPrePosts;
        private Contain[] lContains;

        /// <summary>
        /// The Name of the PeriodDescription instance.
        /// </summary>
        /// <remarks>
        /// This property should be unique.
        /// </remarks>
        /// <exception cref="System.ArgumentNullException"></exception>
        public String Name
        {
            get
            {
                return lName;
            }
            set
            {
                if (value == null)
                    throw new ArgumentNullException();
                else
                    lName = value;
            }
        }

        /// <summary>
        /// The Description of the PeriodDescription instance.
        /// </summary>
        /// <exception cref="System.ArgumentNullException"></exception>
        public String Description
        {
            get
            {
                return lDescription;
            }
            set
            {
                if (value == null)
                    throw new ArgumentNullException();
                else
                    lDescription = value;
            }
        }

        /// <summary>
        /// The PrePosts array define a set of periods which can come before
        /// and after this type of period.
        /// </summary>
        /// <remarks>
```

```
        /// Value cannot be null.
        /// </remarks>
        /// <exception cref="System.ArgumentNullException"></exception>
        public PrePost[] PrePosts
        {
            get
            {
                return lPrePosts;
            }
            set
            {
                if (value == null)
                    throw new ArgumentNullException();
                else
                    lPrePosts = value;
            }
        }

        /// <summary>
        /// The Contains array describe the periods which are contained
        /// within this type of period.
        /// </summary>
        /// <remarks>
        /// Value cannot be null.
        /// </remarks>
        /// <exception cref="System.ArgumentNullException"></exception>
        public Contain[] Contains
        {
            get
            {
                return lContains;
            }
            set
            {
                if (value == null)
                    throw new ArgumentNullException();
                else
                    lContains = value;
            }
        }

        /// <summary>
        /// This constructor is used to create a new PeriodDescription
        /// instance without specifying any unique characteristics.
        /// </summary>
        public PeriodDescription()
        {
            Name = "";
            Description = "";
            PrePosts = new PrePost[0];
            Contains = new Contain[0];
        }

        /// <summary>
        /// This constructor is used to create a new PeriodDescription
        /// instance including specifying any unique characteristics.
        /// </summary>
        /// <param name="name">
        /// The name of the PeriodDescription instance. This should be
        /// unique.
        /// </param>
        /// <param name="description">
        /// The description of the PeriodDescription instance.
        /// </param>
        /// <param name="prePosts">
        /// The prePosts array define a set of periods which can come before
        /// and after this type of period. Value cannot be null.
        /// </param>
        /// <param name="contains">
        /// The contains array describe the periods which are contained
        /// within this type of period. Value cannot be null.
        /// </param>
        public PeriodDescription(String name, String description, PrePost[]
prePosts, Contain[] contains)
        {
            Name = name;
```

```
            Description = description;
            PrePosts = prePosts;
            Contains = contains;
        }

        /// <summary>
        /// The Compile method of the PeriodDescription class is used to
        /// inject into the constructor of the new Type the entries to be
        /// placed in the Contains collection.
        /// </summary>
        /// <param name="baseType">
        /// The baseType property provides the base class of the new Type
        /// so that the Contains collection can be obtained.
        /// </param>
        /// <param name="newType">
        /// The newType property describes the type we are currently
        /// building and for which this method is designed to compile for.
        /// </param>
        /// <param name="typeMap">
        /// The typeMap property describes how the various types all
        /// interact with each other.
        /// </param>
        /// <param name="ilGenerator">
        /// The ilGenerator property provides the ILGenerator for the
        /// constructor so that the necessary code can be injected into it.
        /// </param>
        public void Compile(Type baseType, TypeBuilder newType, TypeMap
typeMap, ILGenerator ilGenerator)
        {
            // The customAttribute local variable is used to store the
            // CustomAttributes that will be defined for this rule type.
            CustomAttributeBuilder customAttribute;

            // The values local variable is used to store an array of
            // string to be held by the customAttribute once compiled.
            List<string> values;

            // First of all construct array of the items this type can
            // contain, then construct a ContainsAttribute for this type.
            values = new List<string>();
            foreach (string contain in typeMap.Contains)
                values.Add(contain);
            customAttribute = new
CustomAttributeBuilder(typeof(RuleSupport.ContainsAttribute).GetConstructor(new
Type[] { typeof(string[]) }), new object[] { values.ToArray() });
            newType.SetCustomAttribute(customAttribute);

            // Now construct an array of the types that can contain this
            // type, then construct a ContainedWithinAttribute for this
            // type.
            values = new List<string>();
            foreach (string containWithin in typeMap.ContainWithin)
                values.Add(containWithin);
            customAttribute = new
CustomAttributeBuilder(typeof(RuleSupport.ContainedWithinAttribute).GetConstruc
tor(new Type[] { typeof(string[]) }), new object[] { values.ToArray() });
            newType.SetCustomAttribute(customAttribute);

            // Now construct a list of the items than can come before
            // and after this type, then construct a PrePostAttribute
            // for this type.
            values = new List<string>();
            foreach (PrePost prePost in typeMap.PrePost)
            {
                values.Add(prePost.Pre);
                values.Add(prePost.Post);
            }
            customAttribute = new
CustomAttributeBuilder(typeof(RuleSupport.PrePostAttribute).GetConstructor(new
Type[] { typeof(string[]) }), new object[] { values.ToArray() });
            newType.SetCustomAttribute(customAttribute);
        }

    }

}
```

11.3.2.9 Periods Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use the these libraries for providing dynamically sizable collections.
using System.Collections.Generic;

// Use this library for file and stream activity.
using System.IO;

// Use this library to provide a means of serialising the objects into
// XML for file storage and network transportation.
using System.Xml.Serialization;

/// <summary>
/// RuleDefinitionLanguage Namespace.
/// This namespace defines core types used to help define a
/// set of rules. These types are not used for execution but for
/// description of the rules.
/// </summary>
namespace RuleDefinitionLanguage
{
    /// <summary>
    /// The Periods class is used to encapsulate a set of Period objects
    /// into a single class. It has a number of supporting methods which
    /// can be used to perform actions like serialization and
    /// deserialization of rules to and from XML.
    /// </summary>
    [Serializable(), XmlInclude(typeof(Period)), XmlInclude(typeof(WP)),
XmlInclude(typeof(RP)), XmlInclude(typeof(AndCondition)),
XmlInclude(typeof(Comparison)), XmlInclude(typeof(Comparitor)),
XmlInclude(typeof(Contain)), XmlInclude(typeof(Math)),
XmlInclude(typeof(OrCondition)), XmlInclude(typeof(PeriodDescription)),
XmlInclude(typeof(PrePost)), XmlInclude(typeof(Value))]
    public class Periods : IEnumerable<Period>
    {

        private List<Period> thePeriods;

        /// <summary>
        /// This is the most simple constructor used to create a new
        /// instance of the Periods class without specifying any Period
        /// objects to add to it.
        /// </summary>
        public Periods()
        {
            thePeriods = new List<Period>();
        }

        /// <summary>
        /// This implementation of the constructor adds an array of
        /// Period objects to the collection and is used if the Period
        /// objects do not need to be loaded from a stream.
        /// </summary>
        /// <param name="periods">
        /// The periods parameter is the array of Period objects to be
        /// added to the collection.
        /// </param>
        public Periods(Period[] periods)
        {
            thePeriods = new List<Period>(periods);
        }

        /// <summary>
        /// This implementation of the constructor is used to load the
        /// Period objects from an Xml encoding from a stream. This allows
        /// rules to be loaded from a file, network connection or other
        /// type of Stream enabled means.
        /// </summary>
        /// <param name="inputStream">
        /// The Stream from with the Period objects are to be loaded.
        /// </param>
```

```
        public Periods(Stream inputStream)
        {
            FromStream(inputStream);
        }

        /// <summary>
        /// This implementation of the constructor is used to load the
        /// Period objects from an Xml encoding from a file.
        /// </summary>
        /// <param name="path">
        /// The path parameter describes the path of the file to load.
        /// </param>
        public Periods(String path)
        {
            // The stream to read from.
            Stream file = new FileStream(path, FileMode.Open, FileAccess.Read,
FileShare.Read);

            // Load the periods form the stream.
            FromStream(file);

            // Close the file.
            file.Close();
        }

        /// <summary>
        /// The FromStream method is a helper method to perform the
        /// function of loading the periods from a stream.
        /// </summary>
        /// <param name="inputStream">
        ///    The input stream to load the periods from.
        /// </param>
        private void FromStream(Stream inputStream)
        {
            // The array to deserialise into.
            Period[] theArray = new Period[0];

            // Create a new XmlSerializer object which is capable of
            // converting a stream of bytes into a Period array for use
            // in this object.
            XmlSerializer newSerializer = new
XmlSerializer(theArray.GetType());

            // Perform the Deserialization from the stream.
            theArray = (Period[])newSerializer.Deserialize(inputStream);

            // Add the array to this collection.
            thePeriods = new List<Period>(theArray);
        }

        /// <summary>
        /// The ToStream method allows the Periods object to convert the
        /// Period objets contained within to Xml and output to the given
        /// Stream. This allows the periods to be serialised to a file,
        /// network location or other Stream enabled location.
        /// </summary>
        /// <param name="outputStream">
        /// The outputStream parameter is the output stream to which the
        /// Period collection will be serialised to.
        /// </param>
        public void ToStream(Stream outputStream)
        {
            // The array to serialise.
            Period[] theArray = ToArray();

            // Create the XmlSerializer object which will be used to
            // convert the array of periods into a stream of bytes.
            XmlSerializer newSerializer = new
XmlSerializer(theArray.GetType());
            //XmlSerializer newSerializer = new XmlSerializer(typeof(Periods));

            // Perform the serialization of the array to the stream.
            newSerializer.Serialize(outputStream, theArray);
            //newSerializer.Serialize(outputStream, this);
        }
```

```
        /// <summary>
        /// The ToStream method allows the Periods object to convert the
        /// Period objets contained within Xml and output to the given path.
        /// </summary>
        /// <param name="path">
        /// The path parameter allows you to specifiy the filename to save
        /// the periods to.
        /// </param>
        public void ToFile(String path)
        {
            // Open a new file stream to write the data to.
            Stream file = new FileStream(path, FileMode.CreateNew,
FileAccess.Write, FileShare.Write);

            // Call the ToStream method to write the periods to the file.
            ToStream(file);

            // Close the file now we've finished with it.
            file.Close();
        }

        /// <summary>
        /// The ToString method returns an XML representation of the Rules
        /// object.
        /// </summary>
        /// <returns>
        /// A String value containing the XML representation of the object.
        /// </returns>
        public override String ToString()
        {
            MemoryStream memStream = new MemoryStream();
            StreamReader streamReader = new StreamReader(memStream);
            ToStream((Stream)memStream);
            memStream.Seek(0, SeekOrigin.Begin);
            return streamReader.ReadToEnd();
        }

        /// <summary>
        /// The Add method allows the addition of individual period
        /// objects to the collection.
        /// </summary>
        /// <param name="period">
        /// The period parameter is the period which will be added to the
        /// collection.
        /// </param>
        /// <returns>
        /// The return value is the index of the collection where the
        /// period has been added.
        /// </returns>
        public int Add(Period period)
        {
            thePeriods.Add(period);
            return thePeriods.Count - 1;
        }

        /// <summary>
        /// The AddRange method allows the addition of an array of
        /// period values to the collection of periods.
        /// </summary>
        /// <param name="periods">
        /// The periods parameters contains the periods to be added to
        /// the collection.
        /// </param>
        public void AddRange(Period[] periods)
        {
            thePeriods.AddRange(periods);
        }

        /// <summary>
        /// The Remove method is used to remove an individual Period object
        /// from the collection of periods.
        /// </summary>
        /// <param name="Index">
        /// The Index within the collection at which the object to be
        /// removed is placed.
        /// </param>
```

```
        public void Remove(int Index)
        {
            thePeriods.RemoveAt(Index);
        }

        /// <summary>
        /// The Remove method is used to remove an individual Period object
        /// from the collection of periods.
        /// </summary>
        /// <param name="period">
        /// The period parameter describes the object within the collection
        /// which is to be removed.
        /// </param>
        public void Remove(Period period)
        {
            thePeriods.Remove(period);
        }

        /// <summary>
        /// The Count property returns the number of entries that exist
        /// in the collection of Period objects as an integer value.
        /// </summary>
        public int Count
        {
            get
            {
                return thePeriods.Count;
            }
        }

        /// <summary>
        /// The indexer for the Periods object allows the getting or
        /// setting of a Period value at a given Index within the Period
        /// collection.
        /// </summary>
        public Period this[int Index]
        {
            get
            {
                return (Period)thePeriods[Index];
            }
            set
            {
                thePeriods[Index] = value;
            }
        }

        /// <summary>
        /// The ToArray method simply returns an array of Period objects
        /// so that the array can be used elsewhere where a collection is
        /// not necessary.
        /// </summary>
        /// <returns>
        /// The return value is the array of Period objects.
        /// </returns>
        public Period[] ToArray()
        {
            return thePeriods.ToArray();
        }

        /// <summary>
        /// The GetEnumerator method implements the interface required by
        /// IEnumerable to allow this collection to be iterated through
        /// with the Foreach operator.
        /// </summary>
        /// <returns>
        /// The IEnumarator interface.
        /// </returns>
        public IEnumerator<Period> GetEnumerator()
        {
            return thePeriods.GetEnumerator();
        }


        /// <summary>
        /// The GetEnumerator method implements the interface required by
```

```
        /// IEnumerable to allow this collection to be iterated through
        /// with the Foreach operator.
        /// </summary>
        /// <returns>
        /// The IEnumarator interface.
        /// </returns>
        System.Collections.IEnumerator
System.Collections.IEnumerable.GetEnumerator()
        {
            return thePeriods.GetEnumerator();
        }

    }

}
```

11.3.2.10 PrePost Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

/// <summary>
/// RuleDefinitionLanguage Namespace.
/// This namespace defines core types used to help define a
/// set of rules. These types are not used for execution but for
/// description of the rules.
/// </summary>
namespace RuleDefinitionLanguage
{

    /// <summary>
    /// PrePost Class. Used to define what comes before or
    /// after a period.
    /// </summary>
    [Serializable()]
    public class PrePost
    {

        private String lPre;
        private String lPost;

        /// <summary>
        /// Pre is the period that can come before the one in question.
        /// </summary>
        /// <exception cref="System.ArgumentNullException"></exception>
        public String Pre
        {
            get
            {
                return lPre;
            }
            set
            {
                if (value == null)
                    throw new ArgumentNullException();
                else
                    lPre = value;
            }
        }

        /// <summary>
        /// Post is the period that can come after the one in question.
        /// </summary>
        /// <exception cref="System.ArgumentNullException"></exception>
        public String Post
        {
            get
            {
                return lPost;
            }
            set
```

```
            {
                if (value == null)
                    throw new ArgumentNullException();
                else
                    lPost = value;
            }
        }

        /// <summary>
        /// This constructor will create a new instance of the class with
        /// blank entries for the Pre and Post members.
        /// </summary>
        public PrePost()
        {
            Pre = "";
            Post = "";
        }

        /// <summary>
        /// This constructor will create a new instance of the class with
        /// paramters to specify the members.
        /// </summary>
        /// <param name="pre">
        /// Pre is the period that can come before the one in question.
        /// </param>
        /// <param name="post">
        /// Post is the period that can come after the one in question.
        /// </param>
        public PrePost(String pre, String post)
        {
            Pre = pre;
            Post = post;
        }
    }

}
```

11.3.2.11 RP Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use these libraries for enabling Reflection of the classes into
// executable code.
using System.Reflection.Emit;

// Use the RuleCompiler namespace to gain access to the TypeMap class.
using RuleCompiler;

/// <summary>
/// RuleDefinitionLanguage Namespace.
/// This namespace defines core types used to help define a
/// set of rules. These types are not used for execution but for
/// description of the rules.
/// </summary>
namespace RuleDefinitionLanguage
{

    /// <summary>
    /// RP Class. Used to define a rest period.
    /// </summary>
    [Serializable()]
    public class RP : Period
    {
        /// <summary>
        /// This constructor is used for defining the rest period without
        /// specifying any unique characteristics.
        /// </summary>
        public RP()
            : base()
        {
```

```
        }

        /// <summary>
        /// This constructor is used for defining the rest period including
        /// parameters for specifying any unique characteristics.
        /// </summary>
        /// <param name="description">
        /// description is an instance of PeriodDescription used to describe
        /// general information about the period such as its name and other
        /// personalised pieces of information.
        /// </param>
        /// <param name="rules">
        /// rules is an instance of a Comparitor which can be any of the
        /// inherited classes of type Comparitor used to define a set of
        /// rules which must evaluate to true in order to meet their
        /// conditions.
        /// </param>
        public RP(PeriodDescription description, Comparitor rules)
            : base(description, rules)
        {
            // This constructor is here to pass the two parameters
            // in to the base class.
        }

        /// <summary>
        /// The Compile method of the RP class uses the Period base class
        /// Compile method to generate the test method of the new class.
        /// </summary>
        /// <param name="moduleBuilder">
        /// The input for this is the module into which the new type will
        /// be created in.
        /// </param>
        /// <param name="typeMap">
        /// The typeMap variable provides a means of describing how the
        /// variables types fit together.
        /// </param>
        /// <param name="debugSupport">
        /// The debugSupport is used to allow the compilation process
        /// emit debug information.
        /// </param>
        /// <returns>
        /// This method returns the TypeBuilder for the new object type
        /// which is created with the Test method.
        /// </returns>
        public override Type Compile(ModuleBuilder moduleBuilder, TypeMap
typeMap, RuleCompiler.DebugSupport debugSupport)
        {
            return base.Compile(moduleBuilder, typeMap,
typeof(RuleSupport.RestPeriod), debugSupport);
        }

    }

}
```

11.3.2.12 Value Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use this library to provide a means of serialising the objects into
// XML for file storage and network transportation.
using System.Xml.Serialization;

// Use these libraries for enabling Reflection of the classes into
// executable code.
using System.Reflection.Emit;

/// <summary>
/// RuleDefinitionLanguage Namespace.
/// This namespace defines core types used to help define a
```

```
/// set of rules. These types are not used for execution but for
/// description of the rules.
/// </summary>
namespace RuleDefinitionLanguage
{

    /// <summary>
    /// The ValueType enum is used to describe different types of
    /// Values which can be used by a Value object.
    /// </summary>
    public enum ValueType
    {
        /// <summary>
        /// A number which will be of a floating point type to allow
        /// for maximum compatibility as a general number.
        /// </summary>
        Number = 0,

        /// <summary>
        /// A quantity of days as a floating point to allow
        /// fractions of days.
        /// </summary>
        Days = 1,

        /// <summary>
        /// A quantity of months as a floating point to allow
        /// fractions of months.
        /// </summary>
        Months = 2,

        /// <summary>
        /// A quantity of years as a floating point to allow
        /// fractions of years.
        /// </summary>
        Years = 3,

        /// <summary>
        /// A quantity of hours as a floating point to allow
        /// fractions of hours.
        /// </summary>
        Hours = 4,

        /// <summary>
        /// A quantity of minutes as a floating point to allow
        /// fractions of minutes.
        /// </summary>
        Minutes = 5,

        /// <summary>
        /// A quantity of seconds as a floating point to allow
        /// fractions of seconds.
        /// </summary>
        Seconds = 6,

        /// <summary>
        /// A quantity of milliseconds as a floating point to allow
        /// fractions of days.
        /// </summary>
        Milliseconds = 7,

        /// <summary>
        /// A string value.
        /// </summary>
        String = 8,

        /// <summary>
        /// A property of the given period or a related period denoted
        /// by a seperation of decimal points within the description.
        /// </summary>
        Parameter = 9
    };

    /// <summary>
    /// Value Class. Used to define a specific value such as a
    /// constant or the name of a property of a type.
    /// </summary>
```

```
    [Serializable(), XmlInclude(typeof(MathOperation))]
    public class Value
    {
        private String lDescription;
        private ValueType lType;

        /// <summary>
        /// The Description member holds the actual value of this class
        /// as a String. Any conversion to other types as described by
        /// Type can be carried out later.
        /// </summary>
        /// <exception cref="System.ArgumentNullException"></exception>
        public String Description
        {
            get
            {
                return lDescription;
            }
            set
            {
                if (value == null)
                    throw new ArgumentNullException();
                else
                    lDescription = value;
            }
        }

        /// <summary>
        /// The Type is a ValueType enum variable used to describe the type
        /// of data held by Description.
        /// </summary>
        public ValueType Type
        {
            get
            {
                return lType;
            }
            set
            {
                lType = value;
            }
        }

        /// <summary>
        /// The default constructor of the value class assumes an empty
        /// String value and type for the member variables.
        /// </summary>
        public Value()
        {
            Description = "";
            Type = ValueType.String;
        }

        /// <summary>
        /// This constructor allocates the member values by the parameters
        /// passed in.
        /// </summary>
        /// <param name="description">
        /// The description parameter holds the actual value of this class
        /// as a String. Any conversion to other types as described by
        /// Type can be carried out later.
        /// </param>
        /// <param name="type">
        /// The type is a ValueType enum variable used to describe the type
        /// of data held by Description.
        /// </param>
        public Value(string description, ValueType type)
        {
            Description = description;
            Type = type;
        }

        /// <summary>
        /// The Compile method for the Value class will emit the
        /// appropriate IL to setup the variable ready for use.
        /// </summary>
```

```
        /// <param name="ilGenerator">
        /// The ilGenerator parameter is used to perform the emitting of
        /// the necessary And comparison code.
        /// </param>
        /// <param name="newLocal">
        /// The newLocal parameter is used to refer to the variable
        /// that stores the resulting parameter of this Value;
        /// </param>
        /// <param name="debugSupport">
        /// The debugSupport is used to allow the compilation process
        /// emit debug information.
        /// </param>
        public virtual void Compile(ILGenerator ilGenerator, ref LocalBuilder
newLocal, RuleCompiler.DebugSupport debugSupport)
        {
            debugSupport.WriteSourceLineAndMark("<Value>", ilGenerator);
            debugSupport.WriteSourceLineAndMark("<Type>" + Type.ToString() +
"</Type>\r\n" + (new string('\t', debugSupport.TabCount)) + "<Description>" +
Description + "</Description>", ilGenerator);

            LocalBuilder tmpLocal;
            switch (Type)
            {
                case ValueType.Number:
                    newLocal = ilGenerator.DeclareLocal(typeof(double));
                    debugSupport.SetVariableName(newLocal, "Value");
                    ilGenerator.Emit(OpCodes.Ldc_R8,
double.Parse(Description));
                    break;
                case ValueType.Days:
                    newLocal = ilGenerator.DeclareLocal(typeof(double));
                    debugSupport.SetVariableName(newLocal, "Value");
                    tmpLocal = ilGenerator.DeclareLocal(typeof(TimeSpan));
                    debugSupport.SetVariableName(tmpLocal, "Value");
                    ilGenerator.Emit(OpCodes.Ldc R8,
double.Parse(Description));
                    ilGenerator.Emit(OpCodes.Call,
typeof(TimeSpan).GetMethod("FromDays", new Type[1] { typeof(double) }));
                    ilGenerator.Emit(OpCodes.Stloc S, tmpLocal);
                    ilGenerator.Emit(OpCodes.Ldloca S, tmpLocal);
                    ilGenerator.Emit(OpCodes.Call,
typeof(TimeSpan).GetMethod("get_TotalHours", new Type[0]));
                    break;
                case ValueType.Months:
                    newLocal = ilGenerator.DeclareLocal(typeof(double));
                    debugSupport.SetVariableName(newLocal, "Value");
                    tmpLocal = ilGenerator.DeclareLocal(typeof(TimeSpan));
                    debugSupport.SetVariableName(tmpLocal, "Value");
                    ilGenerator.Emit(OpCodes.Ldc_R8,
double.Parse(Description));
                    ilGenerator.Emit(OpCodes.Call,
typeof(TimeSpan).GetMethod("FromMonths", new Type[1] { typeof(double) }));
                    ilGenerator.Emit(OpCodes.Stloc_S, tmpLocal);
                    ilGenerator.Emit(OpCodes.Ldloca_S, tmpLocal);
                    ilGenerator.Emit(OpCodes.Call,
typeof(TimeSpan).GetMethod("get TotalHours", new Type[0]));
                    break;
                case ValueType.Years:
                    newLocal = ilGenerator.DeclareLocal(typeof(double));
                    debugSupport.SetVariableName(newLocal, "Value");
                    tmpLocal = ilGenerator.DeclareLocal(typeof(TimeSpan));
                    debugSupport.SetVariableName(tmpLocal, "Value");
                    ilGenerator.Emit(OpCodes.Ldc R8,
double.Parse(Description));
                    ilGenerator.Emit(OpCodes.Call,
typeof(TimeSpan).GetMethod("FromYears", new Type[1] { typeof(double) }));
                    ilGenerator.Emit(OpCodes.Stloc S, tmpLocal);
                    ilGenerator.Emit(OpCodes.Ldloca S, tmpLocal);
                    ilGenerator.Emit(OpCodes.Call,
typeof(TimeSpan).GetMethod("get_TotalHours", new Type[0]));
                    break;
                case ValueType.Hours:
                    newLocal = ilGenerator.DeclareLocal(typeof(double));
                    debugSupport.SetVariableName(newLocal, "Value");
                    tmpLocal = ilGenerator.DeclareLocal(typeof(TimeSpan));
                    debugSupport.SetVariableName(tmpLocal, "Value");
```

```
                    ilGenerator.Emit(OpCodes.Ldc_R8,
double.Parse(Description));
                    ilGenerator.Emit(OpCodes.Call,
typeof(TimeSpan).GetMethod("FromHours", new Type[1] { typeof(double) }));
                    ilGenerator.Emit(OpCodes.Stloc_S, tmpLocal);
                    ilGenerator.Emit(OpCodes.Ldloca_S, tmpLocal);
                    ilGenerator.Emit(OpCodes.Call,
typeof(TimeSpan).GetMethod("get_TotalHours", new Type[0]));
                    break;
                case ValueType.Minutes:
                    newLocal = ilGenerator.DeclareLocal(typeof(double));
                    debugSupport.SetVariableName(newLocal, "Value");
                    tmpLocal = ilGenerator.DeclareLocal(typeof(TimeSpan));
                    debugSupport.SetVariableName(tmpLocal, "Value");
                    ilGenerator.Emit(OpCodes.Ldc_R8,
double.Parse(Description));
                    ilGenerator.Emit(OpCodes.Call,
typeof(TimeSpan).GetMethod("FromMinutes", new Type[1] { typeof(double) }));
                    ilGenerator.Emit(OpCodes.Stloc_S, tmpLocal);
                    ilGenerator.Emit(OpCodes.Ldloca_S, tmpLocal);
                    ilGenerator.Emit(OpCodes.Call,
typeof(TimeSpan).GetMethod("get_TotalHours", new Type[0]));
                    break;
                case ValueType.Seconds:
                    newLocal = ilGenerator.DeclareLocal(typeof(double));
                    debugSupport.SetVariableName(newLocal, "Value");
                    tmpLocal = ilGenerator.DeclareLocal(typeof(TimeSpan));
                    debugSupport.SetVariableName(tmpLocal, "Value");
                    ilGenerator.Emit(OpCodes.Ldc_R8,
double.Parse(Description));
                    ilGenerator.Emit(OpCodes.Call,
typeof(TimeSpan).GetMethod("FromSeconds", new Type[1] { typeof(double) }));
                    ilGenerator.Emit(OpCodes.Stloc_S, tmpLocal);
                    ilGenerator.Emit(OpCodes.Ldloca_S, tmpLocal);
                    ilGenerator.Emit(OpCodes.Call,
typeof(TimeSpan).GetMethod("get_TotalHours", new Type[0]));
                    break;
                case ValueType.Milliseconds:
                    newLocal = ilGenerator.DeclareLocal(typeof(double));
                    debugSupport.SetVariableName(newLocal, "Value");
                    tmpLocal = ilGenerator.DeclareLocal(typeof(TimeSpan));
                    debugSupport.SetVariableName(tmpLocal, "Value");
                    ilGenerator.Emit(OpCodes.Ldc_R8,
double.Parse(Description));
                    ilGenerator.Emit(OpCodes.Call,
typeof(TimeSpan).GetMethod("FromMilliseconds", new Type[1] { typeof(double)
}));
                    ilGenerator.Emit(OpCodes.Stloc_S, tmpLocal);
                    ilGenerator.Emit(OpCodes.Ldloca_S, tmpLocal);
                    ilGenerator.Emit(OpCodes.Call,
typeof(TimeSpan).GetMethod("get_TotalHours", new Type[0]));
                    break;
                case ValueType.String:
                    newLocal = ilGenerator.DeclareLocal(typeof(String));
                    debugSupport.SetVariableName(newLocal, "Value");
                    ilGenerator.Emit(OpCodes.Ldstr, Description);
                    break;
                case ValueType.Parameter:
                    newLocal = ilGenerator.DeclareLocal(typeof(double));
                    debugSupport.SetVariableName(newLocal, "Value");
                    ilGenerator.Emit(OpCodes.Ldarg_0);
                    ilGenerator.Emit(OpCodes.Ldstr, Description);
                    ilGenerator.Emit(OpCodes.Call,
typeof(RuleSupport.Period).GetMethod("GetProperty", new Type[] { typeof(String)
}));
                    break;
            }

            // Finally, save the value to return as a result of this call.
            ilGenerator.Emit(OpCodes.Stloc_S, newLocal);
            debugSupport.WriteSourceLineAndMark("</Value>", ilGenerator);
        }

    }

}
```

11.3.2.13 WP Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use these libraries for enabling Reflection of the classes into
// executable code.
using System.Reflection.Emit;

// Use the RuleCompiler namespace to gain access to the TypeMap class.
using RuleCompiler;

/// <summary>
/// RuleDefinitionLanguage Namespace.
/// This namespace defines core types used to help define a
/// set of rules. These types are not used for execution but for
/// description of the rules.
/// </summary>
namespace RuleDefinitionLanguage
{

    /// <summary>
    /// WP Class. Used to define a work period.
    /// </summary>
    [Serializable()]
    public class WP : Period
    {
        /// <summary>
        /// This constructor is used for defining the work period without
        /// specifying any unique characteristics.
        /// </summary>
        public WP()
            : base()
        {
        }

        /// <summary>
        /// This constructor is used for defining the work period including
        /// parameters for specifying any unique characteristics.
        /// </summary>
        /// <param name="description">
        /// description is an instance of PeriodDescription used to describe
        /// general information about the period such as its name and other
        /// personalised pieces of information.
        /// </param>
        /// <param name="rules">
        /// rules is an instance of a Comparitor which can be any of the
        /// inherited classes of type Comparitor used to define a set of
        /// rules which must evaluate to true in order to meet their
        /// conditions.
        /// </param>
        public WP(PeriodDescription description, Comparitor rules)
            : base(description, rules)
        {
            // This constructor is here to pass the two parameters
            // in to the base class.
        }

        /// <summary>
        /// The Compile method of the WP class uses the Period base class
        /// Compile method to generate the test method of the new class.
        /// </summary>
        /// <param name="moduleBuilder">
        /// The input for this is the module into which the new type will
        /// be created in.
        /// </param>
        /// <param name="typeMap">
        /// The typeMap variable provides a means of describing how the
        /// variables types fit together.
        /// </param>
        /// <param name="debugSupport">
        /// The debugSupport is used to allow the compilation process
        /// emit debug information.
```

```
        /// </param>
        /// <returns>
        /// This method returns the TypeBuilder for the new object type
        /// which is created with the Test method.
        /// </returns>
        public override Type Compile(ModuleBuilder moduleBuilder, TypeMap
typeMap, RuleCompiler.DebugSupport debugSupport)
        {
            return base.Compile(moduleBuilder, typeMap,
typeof(RuleSupport.WorkPeriod), debugSupport);
        }

    }

}
```

11.3.3 RuleSupport Namespace

11.3.3.1 Contains Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use the these libraries for providing dynamically sizable collections.
using System.Collections.Generic;

/// <summary>
/// RuleSupport Namespace.
/// This namespace provides a range of functionality to support both the
/// compilation and execution of rules.
/// </summary>
namespace RuleSupport
{

    /// <summary>
    /// The Contains class is designed to describe those
    /// types of period that are stored within a particular
    /// class, such as an Appointment stored within a Daily
    /// Work Period.
    /// </summary>
    [Serializable()]
    public class Contains : SortedList<String, DirtyList<IPeriodElement>>
    {

        /// <summary>
        /// The Add method provides the ability to Add a
        /// list of entries contained within the SortedList
        /// using the containName as the key and creating
        /// a new DirtyList as the value.
        /// </summary>
        /// <param name="containName">
        /// The containing name used to refer to the new
        /// DirtyList with.
        /// </param>
        public void Add(String containName)
        {
            base.Add(containName, new DirtyList<IPeriodElement>());
        }

        /// <summary>
        /// The AddRange method provides the ability to add
        /// a range of entries to the SortedList, each
        /// entry will use the name provided as the key and
        /// have a new DirtyList created as its value.
        /// </summary>
        /// <param name="containNames">
        /// The containNames is a string array describing the
        /// range of names to created with the SortedList.
        /// </param>
        public void AddRange(String[] containNames)
        {
            for (int i = 0; i < containNames.Length; i++)
                Add(containNames[i]);
        }

        /// <summary>
        /// The GetTypes method returns an array of strings
        /// representing the Keys used within the SortedList.
        /// </summary>
        /// <returns>
        /// An array of string representing the Key used
        /// within the SortedList.
        /// </returns>
        public String[] GetTypes()
        {
            List<string> arr = new List<string>(base.Keys.Count);
            foreach (string a in base.Keys)
```

```
                arr.Add(a);
            return arr.ToArray();
        }

        /// <summary>
        /// The IsDirty method bases its dirty value on the
        /// items stored within the various arrays that are
        /// stored within its sorted list.
        /// </summary>
        /// <returns>
        /// The return result is a boolean value describing
        /// the whether or not there are Dirty values within
        /// the arrays stored within the sorted list.
        /// </returns>
        public bool IsDirty()
        {
            for (int i = 0; i < base.Count; i++)
                if (this[i].IsDirty()) return true;
            return false;
        }

        /// <summary>
        /// The ClearDirty method runs through each array
        /// stored within its SortedList, clearly the dirty
        /// flag..
        /// </summary>
        public void ClearDirty()
        {
            for (int i = 0; i < base.Count; i++)
                this[i].ClearDirty();
        }

        /// <summary>
        /// This indexer provides an additional means of
        /// accessing members of the SortedList based upon
        /// the index of the Key, stored in alphabetical
        /// order.
        /// </summary>
        /// <param name="index">
        /// The index provided refers to the key position
        /// within the Keys collection.
        /// </param>
        /// <returns>
        /// The return value is the DirtyList object
        /// related to the Key found at the give index in
        /// the Keys collection.
        /// </returns>
        public DirtyList<IPeriodElement> this[int index]
        {
            get
            {
                return (DirtyList<IPeriodElement>)base[base.Keys[index]];
            }
        }

        /// <summary>
        /// The GetCombined method takes all of the entries stored
        /// within the the various internal arrays and produced a
        /// single larger list.
        /// </summary>
        /// <returns></returns>
        public DirtyList<IPeriodElement> GetCombined()
        {
            DirtyList<IPeriodElement> newCol = new DirtyList<IPeriodElement>();
            for (int i = 0; i < Count; i++)
                newCol.AddRange(this[i]);
            return newCol;
        }

    }

}
```

11.3.3.2 DirtyList Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use the these libraries for providing dynamically sizable collections.
using System.Collections.Generic;

/// <summary>
/// RuleSupport Namespace.
/// This namespace provides a range of functionality to support both the
/// compilation and execution of rules.
/// </summary>
namespace RuleSupport
{

    /// <summary>
    /// The DirtyList class is basically a slightly more
    /// advanced version of its base class List with
    /// some slight modifications.
    ///
    /// Firstly the class has the ability to determine
    /// whether or not it is dirty, ie, whether an item
    /// has been added or removed since the dirty flag was
    /// last cleared. This can help cut down on the over
    /// processing of data at a later stage.
    ///
    /// It also has a fairly advanced Sort method, which
    /// is able to generate a sorted list based upon
    /// a reflected member of the contained class set.
    /// </summary>
    [Serializable()]
    public class DirtyList<T> : List<T>
    {

        private bool dirty = true;
        public bool IsDirty() { return dirty; }
        public void ClearDirty() { dirty = false; }

        /// <summary>
        /// The Add method for the DirtyList is similar
        /// to the traditional List Add method except
        /// that it also sets the DirtyList's dirty flag.
        /// </summary>
        /// <param name="item">
        /// The item to add to the list.
        /// </param>
        public new void Add(T item)
        {
            dirty = true;
            base.Add(item);
        }

        /// <summary>
        /// The AddRange method for the DirtyList is similar
        /// to the traditional List AddRange method except
        /// that it also sets the DirtyList's dirty flag.
        /// </summary>
        /// <param name="collection">
        /// The collection of items to add to the list.
        /// </param>
        public new void AddRange(IEnumerable<T> collection)
        {
            dirty = true;
            base.AddRange(collection);
        }

        /// <summary>
        /// The Clear method for the DirtyList is similar
        /// to the traditional List Clear method except
        /// that it also sets the DirtyList's dirty flag.
        /// </summary>
        public new void Clear()
        {
            dirty = true;
```

```
            base.Clear();
        }

        /// <summary>
        /// The Insert method for the DirtyList is similar
        /// to the traditional List Clear method except
        /// that it also sets the DirtyList's dirty flag.
        /// </summary>
        /// <param name="index">
        /// The index parameter specifies the index location
        /// within the List to start inserting the item.
        /// </param>
        /// <param name="item">
        /// The item parameter specifies the item to be
        /// inserted into the List.
        /// </param>
        public new void Insert(int index, T item)
        {
            dirty = true;
            base.Insert(index, item);
        }

        /// <summary>
        /// The InsertRange method for the DirtyList is similar
        /// to the traditional List Clear method except
        /// that it also sets the DirtyList's dirty flag.
        /// </summary>
        /// <param name="index">
        /// The index parameter specifies the index location
        /// within the List to start inserting the range of
        /// values.
        /// </param>
        /// <param name="collection">
        /// The collection parameter specifies the items to be
        /// inserted into the List.
        /// </param>
        public new void InsertRange(int index, IEnumerable<T> collection)
        {
            dirty = true;
            base.InsertRange(index, collection);
        }

        /// <summary>
        /// The Remove method for the DirtyList is similar
        /// to the traditional List Clear method except
        /// that it also sets the DirtyList's dirty flag.
        /// </summary>
        /// <param name="item">
        /// The item parameter specifies the item to be
        /// removed from the List.
        /// </param>
        public new void Remove(T item)
        {
            dirty = true;
            base.Remove(item);
        }

        /// <summary>
        /// The RemoveAll method for the DirtyList is similar
        /// to the traditional List Clear method except
        /// that it also sets the DirtyList's dirty flag.
        /// </summary>
        /// <param name="match">
        /// The match parameter provides the Predicate to
        /// determine matching criteria for items to be
        /// removed.
        /// </param>
        /// <returns>
        /// The return values indicates the number of items
        /// removed from the List.
        /// </returns>
        public new int RemoveAll(Predicate<T> match)
        {
            dirty = true;
            return base.RemoveAll(match);
        }
```

```
        /// <summary>
        /// The RemoveAt method for the DirtyList is similar
        /// to the traditional List Clear method except
        /// that it also sets the DirtyList's dirty flag.
        /// </summary>
        /// <param name="index">
        /// The index parameter describes the index location
        /// of the item to remove from the List.
        /// </param>
        public new void RemoveAt(int index)
        {
            dirty = true;
            base.RemoveAt(index);
        }

        /// <summary>
        /// The RemoveRange method for the DirtyList is similar
        /// to the traditional List Clear method except
        /// that it also sets the DirtyList's dirty flag.
        /// </summary>
        /// <param name="index">
        /// The index parameter describes the index location of
        /// the first item to be removed from the List.
        /// </param>
        /// <param name="count">
        /// The count parameter describes the number of items
        /// from the given index to remove from the list.
        /// </param>
        public new void RemoveRange(int index, int count)
        {
            dirty = true;
            base.RemoveRange(index, count);
        }

        /// <summary>
        /// This sort method is designed to take a MemberInfo
        /// object which describes a particular member of the
        /// contained objects and then sorts the entries based
        /// upon that member.
        /// </summary>
        /// <param name="member">
        /// The member of the class to carry out a sort with.
        /// </param>
        /// <returns>
        /// A sorted list containing the array elements sorted by
        /// the given member.
        /// </returns>
        public SortedList<string, T> Sort(System.Reflection.MemberInfo member)
        {
            SortedList<string, T> newList = new SortedList<string,
T>(this.Count);
            for (int i = 0; i < this.Count; i++)
            {
                switch (member.MemberType)
                {
                    case System.Reflection.MemberTypes.Field:
                        System.Reflection.FieldInfo field =
(System.Reflection.FieldInfo)member;
                        newList.Add(GetString(field.GetValue(this[i])) + ":" +
i.ToString(), this[i]);
                        break;
                    case System.Reflection.MemberTypes.Property:
                        System.Reflection.PropertyInfo property =
(System.Reflection.PropertyInfo)member;
                        newList.Add(GetString(property.GetValue(this[i], null))
+ ":" + i.ToString(), this[i]);
                        break;
                    case System.Reflection.MemberTypes.Method:
                        System.Reflection.MethodInfo method =
(System.Reflection.MethodInfo)member;
                        newList.Add(GetString(method.Invoke(this[i], new
object[0] { })) + ":" + i.ToString(), this[i]);
                        break;
                }
            }
```

```
            return newList;
        }

        /// <summary>
        /// The GetString method is a helper method returns which
        /// returns a string representation of the object provided.
        /// It is mainly used to turn DateTime objects into a
        /// useful string.
        /// </summary>
        /// <param name="value">
        /// The value parameter provides the object which requires
        /// a string representation.
        /// </param>
        /// <returns>
        /// The return value is a string representation of value.
        /// </returns>
        private string GetString(object value)
        {
            if (value is DateTime)
                return ((DateTime)value).ToString("yyyyMMddHHmmss");
            else
                return value.ToString();
        }

    }

}
```

11.3.3.3 IPeriodElement Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

/// <summary>
/// RuleSupport Namespace.
/// This namespace provides a range of functionality to support both the
/// compilation and execution of rules.
/// </summary>
namespace RuleSupport
{

    /// <summary>
    /// A base interface which all others that are exposed from here
    /// tend to inherit from.
    /// </summary>
    public interface IPeriodElement
    {
        /// <summary>
        /// The Start property provides and IPeriodElement with
        /// a means of describing its Start date/time.
        /// </summary>
        DateTime Start { get; }

        /// <summary>
        /// The Finish property provides and IPeriodElement with
        /// a means of describing its Finish date/time.
        /// </summary>
        DateTime Finish { get; }

        /// <summary>
        /// The RestLength property provides and IPeriodElement with
        /// a means of describing its length of rest.
        /// </summary>
        TimeSpan RestLength { get; }

        /// <summary>
        /// The WorkLength property provides and IPeriodElement with
        /// a means of describing its length of work.
        /// </summary>
        TimeSpan WorkLength { get; }
    }
```

```
}
```

11.3.3.4 ITestable Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

/// <summary>
/// RuleSupport Namespace.
/// This namespace provides a range of functionality to support both the
/// compilation and execution of rules.
/// </summary>
namespace RuleSupport
{

    /// <summary>
    /// The PeriodRequestDelegate is designed to allow any type
    /// rule to request the information from other rules or data
    /// sitting relative to them within a work plan. This way if,
    /// for example, a daily rest period rule has a length which
    /// is dependant upon the previous daily work period length,
    /// the daily rest period can request access to the previous
    /// daily work period.
    /// </summary>
    /// <param name="source">
    /// The IPeriodElement who is requesting access to a relative
    /// item.
    /// </param>
    /// <param name="relativeIndex">
    /// The index of the item relative to the requesting
    /// IPeriodElement. For example, -1 would recieve the
    /// IPeriodElement that came before it and +1 would return the
    /// next IPeriodElement.
    /// </param>
    /// <returns>
    /// The IPeriodElement item that was located relative to the
    /// source IPeriodElement.
    /// </returns>
    public delegate IPeriodElement PeriodRequestDelegate(IPeriodElement source,
int relativeIndex);
    public delegate IPeriodElement PeriodTypeRequestDelegate(IPeriodElement
source, int relativeIndex, Type periodType);


    /// <summary>
    /// The ITestable interface defines a number of attributes that
    /// any object which has the ability to be tested must have of
    /// provide a facility for.
    /// </summary>
    public interface ITestable
    {
        event PeriodRequestDelegate RequestPeriod;
        event PeriodTypeRequestDelegate RequestPeriodType;

        /// <summary>
        /// The Test method returns a bolean value indicating the
        /// success of the test. All testable objects must have a
        /// test method.
        /// </summary>
        /// <returns>
        /// Returns a boolean value indicating the success of the
        /// test.
        /// </returns>
        bool Test();
    }

}
```

11.3.3.5 Period Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use the these libraries for providing dynamically sizable collections.
using System.Collections.Generic;

/// <summary>
/// RuleSupport Namespace.
/// This namespace provides a range of functionality to support both the
/// compilation and execution of rules.
/// </summary>
namespace RuleSupport
{

    /// <summary>
    /// The Period class provides a base class and relevant
    /// functionality for all types of periods. All periods
    /// have a start and finish time and a RestLength and
    /// WorkLength time spans.
    ///
    /// Additionally all period classes can have periods
    /// contained within them, be contained within another
    /// period and have other periods that can come before
    /// or after them.
    /// </summary>
    public abstract class Period : WorkPattern, IPeriodElement, ITestable
    {

        /// <summary>
        /// The ITestable interface states we must have a
        /// Start property which we do not yet want to
        /// define so it is specified as abstract.
        /// </summary>
        public abstract DateTime Start { get; }

        /// <summary>
        /// The ITestable interface states we must have a
        /// Finish property which we do not yet want to
        /// define so it is specified as abstract.
        /// </summary>
        public abstract DateTime Finish { get; }

        /// <summary>
        /// The ITestable interface states we must have a
        /// RestLength property which we do not yet want to
        /// define so it is specified as abstract.
        /// </summary>
        public abstract TimeSpan RestLength { get; }

        /// <summary>
        /// The ITestable interface states we must have a
        /// WorkLength property which we do not yet want to
        /// define so it is specified as abstract.
        /// </summary>
        public abstract TimeSpan WorkLength { get; }

        /// <summary>
        /// Length property describes the difference
        /// between the start and finish of the period.
        /// </summary>
        public TimeSpan Length { get { return new RuleSupport.TimeSpan(Finish -
Start); } }

        /// <summary>
        /// The ITestable interface states we must have a
        /// RequestPeriod event which we do not yet want to
        /// define so it is specified as abstract.
        /// </summary>
        public event PeriodRequestDelegate RequestPeriod;

        /// <summary>
```

```
        /// The ITestable interface states we must have a
        /// RequestPeriodType event which we do not yet want to
        /// define so it is specified as abstract.
        /// </summary>
        public event PeriodTypeRequestDelegate RequestPeriodType;

        /// <summary>
        /// The private contains variable stores a cahced copy
        /// of the loaded attribute details.
        /// </summary>
        private Contains contains = null;

        /// <summary>
        /// The Contains property provides a means accessing
        /// the collection of Types that can be contained within
        /// this type.
        /// </summary>
        public new Contains Contains
        {
            get
            {
                if (contains == null)
                    contains =
((ContainsAttribute)(this.GetType().GetCustomAttributes(typeof(ContainsAttribut
e), false)[0])).Contains;
                return contains;
            }
            set
            {
                contains = value;
            }
        }

        /// <summary>
        /// The ContainWithin property provides a means of
        /// accessing the Types that can contain this type.
        /// </summary>
        public Contains ContainWithin
        {
            get
            {
                return
((ContainedWithinAttribute)(this.GetType().GetCustomAttributes(typeof(Contained
WithinAttribute), false)[0])).Container;
            }
        }

        /// <summary>
        /// The PrePosts property provides a means of
        /// accessing the Types that can come before and
        /// after this type.
        /// </summary>
        public PrePosts PrePosts
        {
            get
            {
                return
((PrePostAttribute)(this.GetType().GetCustomAttributes(typeof(PrePostAttribute)
, false)[0])).PrePosts;
            }
        }

        /// <summary>
        /// The ITestable interface states we must have a
        /// Test method which we do not yet want to
        /// define so it is specified as abstract.
        /// </summary>
        public abstract bool Test();

        /// <summary>
        /// The constructor intialises the object.
        /// </summary>
        public Period()
        {
        }
```

```
        #region Overrides of the Add and Remove Methods

        /// <summary>
        /// This overrident Add method determines which of
        /// the storage containers a given element can be
        /// contained within and add's the object to that
        /// specific collection. If the given object does
        /// not belong to that collection, the method
        /// returns false.
        /// </summary>
        /// <param name="Period">
        /// The Period parameter provides the object to be
        /// added to a container.
        /// </param>
        /// <returns>
        /// The return value is a boolean describing
        /// whether the given object has been added
        /// to one of the containers.
        /// </returns>
        public new bool Add(IPeriodElement Period)
        {
            if (this.Contains.ContainsKey(Period.GetType().Name))
            {
                this.Contains[Period.GetType().Name].Add(Period);

                if (Period is RuleSupport.ITestable)
                    base.Add((ITestable)Period);

                return true;
            }
            else
                return false;
        }

        /// <summary>
        /// The Remove method identifies which of the
        /// containers a given object belows to and
        /// removes the object from that container.
        /// </summary>
        /// <param name="Period">
        /// The Period parameter provides the object to be
        /// removed from a container.
        /// </param>
        public new void Remove(IPeriodElement Period)
        {
            if (this.Contains.ContainsKey(Period.GetType().Name))
            {
                this.Contains[Period.GetType().Name].Remove(Period);

                if (base.Contains(Period))
                    base.Remove((ITestable)Period);
            }
        }

        /// <summary>
        /// The CanContain method provides a quick and simple
        /// lookup, to determine whether this period can contain
        /// a given element, based on the permitted containers
        /// originally define within this periods metadata.
        /// </summary>
        /// <param name="element"></param>
        /// <returns></returns>
        public bool CanContain(IPeriodElement element)
        {
            return this.Contains.ContainsKey(element.GetType().Name);
        }

        #endregion

        #region Relative Period Objects Access Methods

        /// <summary>
        /// The GetProperty method is a relatively complex method
        /// for obtaining a value for a named property item. This
        /// allows, for example, a period to obtain values from
        /// periods around them, periods they are contained within
```

```
        /// or periods they contain.
        /// </summary>
        /// <param name="name">
        /// The qualified name of the property whose value is to be
        /// returned. This would be in the form of, for example,
        /// PRE.Start or PRE(DWP.Finish).
        /// </param>
        /// <returns>
        /// The return value is a double representing the numberic
        /// value of the property.
        /// </returns>
        public double GetProperty(String name)
        {
            Object instance = this;

            String prePart = "";
            String postPart = name;
            Type instanceType = instance.GetType();

            while (postPart.Length > 0)
            {

                if (postPart.IndexOf(".") > 0)
                {

                    prePart = postPart.Substring(0, postPart.IndexOf("."));
                    postPart = postPart.Substring(postPart.IndexOf(".") + 1);

                    if (prePart.ToUpper() == "PRE")
                        instance = ((Period)instance).GetPrevious();
                    else if (prePart.ToUpper() == "POST")
                        instance = ((Period)instance).GetNext();
                    else if (prePart.ToUpper().StartsWith("PRE("))
                        instance =
((Period)instance).GetPrevious(int.Parse(prePart.ToUpper().Substring(0,
4).Substring(0, prePart.ToUpper().Substring(0, 4).Length - 1)));
                    else if (prePart.ToUpper().StartsWith("POST("))
                        instance =
((Period)instance).GetNext(int.Parse(prePart.ToUpper().Substring(0,
5).Substring(0, prePart.ToUpper().Substring(0, 5).Length - 1)));
                    else if (((Period)instance).Contains.ContainsKey(prePart))
                        instance = ((Period)instance).Contains[prePart];
                    else if (instanceType.GetProperty(prePart) != null)
                        instance =
instanceType.GetProperty(prePart).GetValue(instance, new object[] { });
                    else if (instanceType.GetField(prePart) != null)
                        instance =
instanceType.GetField(prePart).GetValue(instance);
                    else if (instanceType.GetMethod(prePart, new Type[] { }) !=
null)
                        instance =
instanceType.GetMethod(prePart).Invoke(instance, new Type[] { });

                    instanceType = instance.GetType();
                }
                else
                {
                    prePart = postPart;
                    postPart = "";
                }

            }

            Object returnValue = null;
            double returnValueD = (double)0;

            if (instanceType.GetProperty(prePart) != null)
                returnValue =
instanceType.GetProperty(prePart).GetValue(instance, new object[] { });

            if (instanceType.GetField(prePart) != null)
                returnValue =
instanceType.GetField(prePart).GetValue(instance);

            if (instanceType.GetMethod(prePart, new Type[] { }) != null)
                returnValue = instanceType.GetMethod(prePart).Invoke(instance,
```

```
new Type[] { });

            if (returnValue.GetType() == typeof(RuleSupport.TimeSpan))
                returnValueD =
(double)((RuleSupport.TimeSpan)returnValue).myTS.TotalHours;
            else
                double.TryParse(returnValue.ToString(),
System.Globalization.NumberStyles.Any,
System.Globalization.CultureInfo.CurrentCulture, out returnValueD);

            return returnValueD;
        }

        /// <summary>
        /// The GetPrevious method is a shortcut method to
        /// obtaining an object located before this one
        /// within a work pattern.
        /// </summary>
        /// <param name="relativeCount">
        /// The relativeCount parameter describes the number
        /// of place back in the work pattern to look to obtain
        /// the object location previous to this one.
        /// </param>
        /// <returns>
        /// The IPeriodElement located the give number of
        /// places before this in the work pattern.
        /// </returns>
        public IPeriodElement GetPrevious(int relativeCount)
        {
            return this.RequestPeriod(this, 0 - relativeCount);
        }

        /// <summary>
        /// The GetNext method is a shortcut method to
        /// obtaining an object located after this one
        /// within a work pattern.
        /// </summary>
        /// <param name="relativeCount">
        /// The relativeCount parameter describes the number
        /// of place forward in the work pattern to look to
        /// obtain the object location previous to this one.
        /// </param>
        /// <returns>
        /// The IPeriodElement located the give number of
        /// places after this in the work pattern.
        /// </returns>
        public IPeriodElement GetNext(int relativeCount)
        {
            return this.RequestPeriod(this, relativeCount);
        }

        /// <summary>
        /// The GetPrevious method is a shortcut method to
        /// obtaining the object located before this one
        /// within a work pattern.
        /// </summary>
        /// <returns>
        /// The IPeriodElement located the before this in
        /// the work pattern.
        /// </returns>
        public IPeriodElement GetPrevious()
        {
            return this.RequestPeriod(this, -1);
        }

        /// <summary>
        /// The GetNext method is a shortcut method to
        /// obtaining the object located after this one
        /// within a work pattern.
        /// </summary>
        /// <returns>
        /// The IPeriodElement located after this in the
        /// work pattern.
        /// </returns>
        public IPeriodElement GetNext()
        {
```

```
            return this.RequestPeriod(this, 1);
        }

        /// <summary>
        /// The GetPrevious method is a shortcut method to
        /// obtaining the object located before this one
        /// within a work pattern.
        /// </summary>
        /// <param name="type">
        /// The type parameter describes the type of object
        /// to obtain from the work pattern.
        /// </param>
        /// <param name="relativeCount">
        /// The relativeCount parameter describes the number
        /// of place back in the work pattern to look to obtain
        /// the object location previous to this one.
        /// </param>
        /// <returns>
        /// The IPeriodElement located the give number of
        /// places before this, of a given type in the work
        /// pattern.
        /// </returns>
        public IPeriodElement GetPrevious(Type type, int relativeCount)
        {
            return this.RequestPeriodType(this, 0 - relativeCount, type);
        }

        /// <summary>
        /// The GetNext method is a shortcut method to
        /// obtaining the object located after this one
        /// within a work pattern.
        /// </summary>
        /// <param name="type">
        /// The type parameter describes the type of object
        /// to obtain from the work pattern.
        /// </param>
        /// <param name="relativeCount">
        /// The relativeCount parameter describes the number
        /// of place forward in the work pattern to look to
        /// obtain the object location after to this one.
        /// </param>
        /// <returns>
        /// The IPeriodElement located the give number of
        /// places after this, of a given type in the work
        /// pattern.
        /// </returns>
        public IPeriodElement GetNext(Type type, int relativeCount)
        {
            return this.RequestPeriodType(this, relativeCount, type);
        }

        /// <summary>
        /// The GetPrevious method is a shortcut method to
        /// obtaining the object located before this one
        /// within a work pattern.
        /// </summary>
        /// <param name="type">
        /// The type parameter describes the type of object
        /// to obtain from the work pattern.
        /// </param>
        /// <returns>
        /// The IPeriodElement located the before this, of a
        /// given type in the work pattern.
        /// </returns>
        public IPeriodElement GetPrevious(Type type)
        {
            return this.RequestPeriodType(this, -1, type);
        }

        /// <summary>
        /// The GetNext method is a shortcut method to
        /// obtaining the object located before this one
        /// within a work pattern.
        /// </summary>
        /// <param name="type">
        /// The type parameter describes the type of object
```

```
        /// to obtain from the work pattern.
        /// </param>
        /// <returns>
        /// The IPeriodElement located the after this, of a
        /// given type in the work pattern.
        /// </returns>
        public IPeriodElement GetNext(Type type)
        {
            return this.RequestPeriodType(this, 1, type);
        }

        /// <summary>
        /// The GetEndStack method is an interative method used
        /// for looking down the work pattern hierachy and
        /// gaining access to the various layers.
        /// </summary>
        /// <param name="layers"></param>
        public void GetEndStack(Stack<RuleSupport.Period> layers)
        {
            // Add ourselves to the stack, for of all..
            layers.Push(this);

            // If we contain anything that resembles a Period
            // type of object, call its GetEndStack method.
            if ((this.Count > 0) && (this[this.Count - 1] is
RuleSupport.Period))
                ((RuleSupport.Period)this[this.Count - 1]).GetEndStack(layers);
        }

        #endregion

    }

}
```

11.3.3.6 PeriodDescriptionAttribute Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

namespace RuleSupport
{

    /// <summary>
    /// The PeriodDescriptionAttribute class provides a means of giving
    /// description to a period through an attribute, available later at
    /// runtime.
    /// </summary>
    public class PeriodDescriptionAttribute : Attribute
    {
        public string Name { get; private set; }
        public string Description { get; private set; }

        public PeriodDescriptionAttribute(string name, string description)
        {
            Name = name;
            Description = description;
        }
    }

    /// <summary>
    /// The PrePostAttribute provides a means of describing PrePost
    /// information through an attribute so that the periods pre and
    /// post information can be accessed at runtime without the need
    /// for class initialisation.
    /// </summary>
    public class PrePostAttribute : Attribute
    {
        public PrePosts PrePosts { get; private set; }

        public PrePostAttribute(params string[] preposts)
```

```
        {
            if (preposts.Length % 2 == 1)
                throw new InvalidOperationException("PrePosts should be
provided in pairs.");

            PrePosts = new PrePosts();
            for (int i = 0; i < preposts.Length / 2; i += 2)
                PrePosts.Add(preposts[i], preposts[i + 1]);
        }
    }

    /// <summary>
    /// The ContainsAttribute provides a means of describing the
    /// types that can be contained within this type through an
    /// attribute so that they can be accessed at runtime without
    /// the need for class initialisation.
    /// </summary>
    public class ContainsAttribute : Attribute
    {
        public Contains Contains { get; private set; }

        public ContainsAttribute(params string[] contains)
        {
            Contains = new Contains();
            foreach (string contain in contains)
                Contains.Add(contain);
        }
    }

    /// <summary>
    /// The ContainedWithinAttribute provides a means of
    /// describing the types that can contain the given type
    /// through an attribute so that they can be accessed at
    /// runtime without the need for class initialisation.
    /// </summary>
    public class ContainedWithinAttribute : Attribute
    {
        public Contains Container { get; private set; }

        public ContainedWithinAttribute(params string[] containers)
        {
            Container = new Contains();
            foreach (string container in containers)
                Container.Add(container);
        }
    }

}
```

11.3.3.7 PrePost Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use the these libraries for providing dynamically sizable collections.
using System.Collections.Generic;

/// <summary>
/// RuleSupport Namespace.
/// This namespace provides a range of functionality to support both the
/// compilation and execution of rules.
/// </summary>
namespace RuleSupport
{

    /// <summary>
    /// The PrePost class is designed to represent a description
    /// of those period types that come before and after a particular
    /// period.
    ///
    /// This is the run-time version of the PrePost class as opposed
```

```
    /// to the compile-time version which comes under the
    /// RuleDefinitionLanguage namespace.
    /// </summary>
    [Serializable()]
    public class PrePost
    {
        public String Pre;
        public String Post;

        public PrePost(String pre, String post)
        {
            Pre = pre;
            Post = post;
        }
    }

    /// <summary>
    /// The PrePosts class is a specialised form the the List class
    /// providing an additional function for adding a PrePost item
    /// by specifying the two constructor arguments of the PrePost.
    /// </summary>
    [Serializable()]
    public class PrePosts : List<PrePost>
    {
        /// <summary>
        /// The Add method provides the ability to add an PrePost
        /// entry based upon the string versions of the Pre and the
        /// Post, rather than an object instance of a PrePost.
        /// </summary>
        /// <param name="Pre">
        /// The string representation of the Pre entry.
        /// </param>
        /// <param name="Post">
        /// The string representation of the Post entry.
        /// </param>
        public void Add(String Pre, String Post)
        {
            base.Add(new PrePost(Pre, Post));
        }
    }

}
```

11.3.3.8 RestPeriod Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

/// <summary>
/// RuleSupport Namespace.
/// This namespace provides a range of functionality to support both the
/// compilation and execution of rules.
/// </summary>
namespace RuleSupport
{

    /// <summary>
    /// This is the base RestPeriod class used to describe
    /// a general rest period. It is inherited by classes
    /// generated by the rule compiler and is designed for
    /// the purpose of carrying out some of the basic
    /// functionality that is expected from a standard rest
    /// period.
    /// </summary>
    [Serializable()]
    public abstract class RestPeriod : Period
    {

        /// <summary>
        /// The constructor calls the base classes contructor
        /// to initialise any relevant variables.
```

```
        /// </summary>
        public RestPeriod()
            : base()
        {
        }

        /// <summary>
        /// The Start time of the rest period is determined
        /// by looking at the previous period within the
        /// work pattern to see when it finished, using this
        /// as the start time.
        /// </summary>
        public override DateTime Start
        {
            get
            {
                IPeriodElement previous = GetPrevious();
                if (previous == null)
                    return new DateTime();
                else
                    return previous.Finish;
            }
        }

        /// <summary>
        /// The Finish time of the rest period is determined
        /// by looking at the next period within the
        /// work pattern to see when it starts, using this
        /// as the finish time.
        /// </summary>
        public override DateTime Finish
        {
            get
            {
                IPeriodElement next = GetNext();
                if (next == null)
                    return new DateTime();
                else
                    return next.Start;
            }
        }

        /// <summary>
        /// As this is a rest period, the entire duration of
        /// the period is used as the rest length.
        /// </summary>
        public override TimeSpan RestLength
        {
            get
            {
                IPeriodElement previous = GetPrevious();
                IPeriodElement next = GetNext();

                if ((previous == null) || (next == null))
                    return new TimeSpan(long.MaxValue);
                else
                    return ((TimeSpan)(next.Start - previous.Finish));
            }
        }

        /// <summary>
        /// As this is a rest period, there is no work and
        /// as such the TimeSpan return contains no time.
        /// </summary>
        public override TimeSpan WorkLength
        {
            get
            {
                return new TimeSpan();
            }
        }

    }

}
```

11.3.3.9 TimeSpan Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

/// <summary>
/// RuleSupport Namespace.
/// This namespace provides a range of functionality to support both the
/// compilation and execution of rules.
/// </summary>
namespace RuleSupport
{
    /// <summary>
    /// This new definition of the TimeSpan class supports additional
    /// conversion between the original System.TimeSpan class and the
    /// System.DateTime class.
    /// </summary>
    [Serializable()]
    public struct TimeSpan
    {
        /// <summary>
        /// The internal System.TimeSpan object that contains the base
        /// functionality of a TimeSpan.
        /// </summary>
        internal System.TimeSpan myTS; // = new System.TimeSpan(0);

        /// <summary>
        /// The ToString method provides the TimeSpan object as a string
        /// representation.
        /// </summary>
        /// <returns>
        /// The string representation of the TimeSpan object.
        /// </returns>
        public override string ToString()
        {
            return myTS.ToString();
        }

        /// <summary>
        /// The implicit operator converts the TimeSpan into a DateTime
        /// data type.
        /// </summary>
        /// <param name="ts">
        /// The ts parameter is the TimeSpan object who will be converted
        /// into a DateTime object.
        /// </param>
        /// <returns>
        /// The return value is the DateTime version of the TimeSpan.
        /// </returns>
        public static implicit operator DateTime(TimeSpan ts)
        {
            return new DateTime(0, 0, ts.myTS.Days, ts.myTS.Hours,
ts.myTS.Minutes, ts.myTS.Seconds);
        }

        /// <summary>
        /// The implicit operator converts the TimeSpan into a
        /// System.TimeSpan data type.
        /// </summary>
        /// <param name="ts">
        /// The ts parameter is the TimeSpan object who will be converted
        /// into a System.TimeSpan object.
        /// </param>
        /// <returns>
        /// The return value is the System.TimeSpan version of the TimeSpan.
        /// </returns>
        public static implicit operator System.TimeSpan(TimeSpan ts)
        {
            return ts.myTS;
        }

        /// <summary>
        /// The implicit operator converts a DateTime type into a TimeSpan
```

```
        /// data type.
        /// </summary>
        /// <param name="ts">
        /// The dt parameter is the DateTime object who will be converted
        /// into a TimeSpan object.
        /// </param>
        /// <returns>
        /// The return value is the TimeSpan version of the DateTime.
        /// </returns>
        public static implicit operator TimeSpan(System.DateTime dt)
        {
            return new TimeSpan(new System.TimeSpan(dt.Day, dt.Hour, dt.Minute,
dt.Second));
        }

        /// <summary>
        /// The explicit operator converts the System.TimeSpan into a
        /// TimeSpan data type.
        /// </summary>
        /// <param name="ts">
        /// The ts parameter is the System.TimeSpan object who will be
        /// converted into a TimeSpan object.
        /// </param>
        /// <returns>
        /// The return value is the TimeSpan version of the
        /// System.TimeSpan.
        /// </returns>
        public static explicit operator TimeSpan(System.TimeSpan ts)
        {
            // code to convert from System.TimeSpan to TimeSpan
            return new TimeSpan(ts);
        }

        /// <summary>
        /// The operator + carries out the addition of two TimeSpan
        /// objects, returning the result.
        /// </summary>
        /// <param name="a">
        /// The left hand TimeSpan to add to.
        /// </param>
        /// <param name="b">
        /// The right hand TimeSpan to be added.
        /// </param>
        /// <returns>
        /// The result is the additional of the two TimeSpan objects.
        /// </returns>
        public static TimeSpan operator +(TimeSpan a, TimeSpan b)
        {
            return new TimeSpan(a.myTS + b.myTS);
        }

        /// <summary>
        /// The operator - carries out the substraction of two
        /// TimeSpan objects, returning the result.
        /// </summary>
        /// <param name="a">
        /// The left hand TimeSpan to substract from.
        /// </param>
        /// <param name="b">
        /// The right hand TimeSpan to be substracted.
        /// </param>
        /// <returns>
        /// The result is the substraction of the two TimeSpan
        /// objects.
        /// </returns>
        public static TimeSpan operator -(TimeSpan a, TimeSpan b)
        {
            return new TimeSpan(a.myTS - b.myTS);
        }

        /// <summary>
        /// The operator > compares two TimeSpan objects,
        /// returning the result.
        /// </summary>
        /// <param name="a">
        /// The left hand TimeSpan to compare.
```

```
        /// </param>
        /// <param name="b">
        /// The right hand TimeSpan to compare.
        /// </param>
        /// <returns>
        /// The result is the comparison of the of the two
        /// TimeSpan objects.
        /// </returns>
        public static bool operator >(TimeSpan a, TimeSpan b)
        {
            return a.myTS > b.myTS;
        }

        /// <summary>
        /// The operator ? compares two TimeSpan objects,
        /// returning the result.
        /// </summary>
        /// <param name="a">
        /// The left hand TimeSpan to compare.
        /// </param>
        /// <param name="b">
        /// The right hand TimeSpan to compare.
        /// </param>
        /// <returns>
        /// The result is the comparison of the of the two
        /// TimeSpan objects.
        /// </returns>
        public static bool operator <(TimeSpan a, TimeSpan b)
        {
            return a.myTS < b.myTS;
        }

        /// <summary>
        /// The operator >= compares two TimeSpan objects,
        /// returning the result.
        /// </summary>
        /// <param name="a">
        /// The left hand TimeSpan to compare.
        /// </param>
        /// <param name="b">
        /// The right hand TimeSpan to compare.
        /// </param>
        /// <returns>
        /// The result is the comparison of the of the two
        /// TimeSpan objects.
        /// </returns>
        public static bool operator >=(TimeSpan a, TimeSpan b)
        {
            return a.myTS >= b.myTS;
        }

        /// <summary>
        /// The operator ?= compares two TimeSpan objects,
        /// returning the result.
        /// </summary>
        /// <param name="a">
        /// The left hand TimeSpan to compare.
        /// </param>
        /// <param name="b">
        /// The right hand TimeSpan to compare.
        /// </param>
        /// <returns>
        /// The result is the comparison of the of the two
        /// TimeSpan objects.
        /// </returns>
        public static bool operator <=(TimeSpan a, TimeSpan b)
        {
            return a.myTS <= b.myTS;
        }

        /// <summary>
        /// This constructor constructs the object with the
        /// given number of ticks as the length of the
        /// TimeSpan.
        /// </summary>
        /// <param name="ticks">
```

```
        /// The number of ticks that represent this TimeSpan.
        /// </param>
        public TimeSpan(long ticks)
        {
            myTS = new System.TimeSpan(ticks);
        }

        /// <summary>
        /// This constructor constructs the object with the
        /// given number of hours, minutes and seconds as
        /// the length of the TimeSpan.
        /// </summary>
        /// <param name="Hours">
        /// The number of hours this TimeSpan represents.
        /// </param>
        /// <param name="Minutes">
        /// The number of minutes this TimeSpan represents.
        /// </param>
        /// <param name="Seconds">
        /// The number of seconds this TimeSpan represents.
        /// </param>
        public TimeSpan(int Hours, int Minutes, int Seconds)
        {
            myTS = new System.TimeSpan(Hours, Minutes, Seconds);
        }

        /// <summary>
        /// This constructor constructs the object using the
        /// System.TimeSpan to represent its length.
        /// </summary>
        /// <param name="ts">
        /// The System.TimeSpan to use to represent the length
        /// of this TimeSpan.
        /// </param>
        public TimeSpan(System.TimeSpan ts)
        {
            myTS = ts;
        }

        /// <summary>
        /// This constructor constructs the object using the
        /// System.DateTime to represent the length of the
        /// TimeSpan.
        /// </summary>
        /// <param name="dt">
        /// The System.DateTime to use to represent the length
        /// of this TimeSpan.
        /// </param>
        public TimeSpan(System.DateTime dt)
        {
            myTS = new System.TimeSpan(dt.Day, dt.Hour, dt.Minute, dt.Second);
        }

    }

}
```

11.3.3.10 Types Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use the these libraries for providing dynamically sizable collections.
using System.Collections.Generic;

/// <summary>
/// RuleSupport Namespace.
/// This namespace provides a range of functionality to support both the
/// compilation and execution of rules.
/// </summary>
namespace RuleSupport
```

```
{

    /// <summary>
    /// The Types class is used to help in the generation of the work
    /// pattern by providing a means of quickly and easily sorting through
    /// and finding Types which may be contained within or are the container
    /// for other types. Providing the logic for this in a seperate class
    /// helps to relieve complexity on the work pattern and tester classes.
    /// </summary>
    public class Types : SortedList<string, Type>, IEnumerable<Type>
    {

        /// <summary>
        /// The default constructor takes no arguments and simply
        /// calls the constructor for the base class.
        /// </summary>
        public Types()
            : base()
        {
        }

        /// <summary>
        /// This special constructor takes an array of Types to be
        /// added to the collection. This constructor also calls
        /// the default base class constructor.
        /// </summary>
        /// <param name="types">
        /// The array of Type objects to be added to the collection.
        /// </param>
        public Types(Type[] types)
            : base()
        {
            AddRange(types);
        }

        /// <summary>
        /// This Add method provides a means of adding a Type to the
        /// collection without specifying its key as the key is taken
        /// from the given Type objects name.
        /// </summary>
        /// <param name="type">
        /// The Type to be added to the collection.
        /// </param>
        public void Add(Type type)
        {
            base.Add(type.Name, type);
        }

        /// <summary>
        /// This AddRange method provides a means of adding multiple Type
        /// objects to the collection. Each one is added independantly to
        /// the collection, using its Type.Name property as the key for
        /// the entry in the collection.
        /// </summary>
        /// <param name="types">
        /// The array of Type objects to be added to the collection.
        /// </param>
        public void AddRange(Type[] types)
        {
            foreach (Type type in types)
                Add(type);
        }

        /// <summary>
        /// The Contains method returns a boolean value indicating whether
        /// the Type specified exists within the collection.
        /// </summary>
        /// <param name="type">
        /// The Type to check the existance of within the collection.
        /// </param>
        /// <returns>
        /// A boolean value indicating whether the Type exists in the
collection.
        /// </returns>
        public bool Contains(Type type)
        {
```

```
            return base.ContainsKey(type.Name);
        }

        /// <summary>
        /// The GetTypesThatContainType method returns those types contained
        /// within this collection who have the ability to contain the given
        /// type.
        /// </summary>
        /// <param name="containee">
        /// The type for whom container types will be returned.
        /// </param>
        /// <returns>
        /// A new type collection containing those types that can contain the
        /// given containee.
        /// </returns>
        public Types GetTypesThatContainType(Type containee)
        {
            // Create a variable to use as the return value later.
            Types newTypeArray = new Types();

            // Loop through each Type stored within this collection to see
            // if if it can contain the given type.
            foreach (Type type in this.Values)
            {
                RuleSupport.ContainsAttribute attr =
(RuleSupport.ContainsAttribute)type.GetCustomAttributes(typeof(RuleSupport.Cont
ainsAttribute), false)[0];
                if (attr.Contains.ContainsKey(containee.Name) &&
(!newTypeArray.Contains(type)))
                    newTypeArray.Add(type);
            }

            return newTypeArray;
        }

        /// <summary>
        /// The GetTypesContainedInType method returns a new collection of
        /// types for those stored within this collection who can be contained
        /// within the given container. In addition, the previous type is
        /// specified in order that those types are returned who are able to
        /// come after the previous type.
        /// </summary>
        /// <param name="container">
        /// The container Type whose Types that can be contained within it are
        /// returned.
        /// </param>
        /// <param name="previous">
        /// The previous Type provided in order to limit the return of Types
        /// to those that can come after this Type.
        /// </param>
        /// <returns>
        /// A new type collection containing those Types that can be contained
        /// within the container type and who can also follow the previous
type.
        /// </returns>
        public Types GetTypesContainedInType(Type container, Type previous)
        {
            // Create a variable to use as the return value later.
            Types newTypes = new Types();

            // Obtain the string value representing the name of the
            // type as found in the PrePosts collection.
            string prev = "";
            if (previous != null)
                prev = previous.Name;

            // Create a new instance of the container in order that we can
            // gain access to the list of types that can be contained within
            // it.
            RuleSupport.ContainsAttribute containsAttr =
(RuleSupport.ContainsAttribute)container.GetCustomAttributes(typeof(RuleSupport
.ContainsAttribute), false)[0];

            // Loop through each of the keys in the Contains property to
            // look at each of the Types that can be contained within the
            // container.
```

```
            foreach (string contains in containsAttr.Contains.Keys)
            {
                // Check to ensure this is a Type available within the Types
                // collection, and not an element of data.
                if (this.ContainsKey(contains))
                {
                    // Create a new instance of the found type in order that
                    // we can access the PrePost collection for the Type.
                    RuleSupport.PrePostAttribute prepostAttr =
(RuleSupport.PrePostAttribute)((Type)this[contains]).GetCustomAttributes(typeof
(RuleSupport.PrePostAttribute), false)[0];

                    // Loop through the PrePosts in order to ensure this Type
                    // can follow the specified previous Type.
                    foreach (PrePost prepost in prepostAttr.PrePosts)
                    {
                        // If we've found it we can add this Type to the
                        // collection and break from the PrePosts foreach
                        // statement.
                        if (prepost.Pre == prev)
                        {
                            // Add the Type to the new collection.
                            newTypes.Add((Type)this[contains]);

                            // Break from the PrePost foreach interations.
                            break;
                        }
                    }
                }
            }

            // Return the new collection of types containing those requested.
            return newTypes;
        }

        /// <summary>
        /// The GetTypesThatProceedType method returns those types stored
        /// within the collection that would normally come after the given
        /// containee in a work plan.
        /// </summary>
        /// <param name="containee">
        /// The containee Type whose proceeding types will be returned.
        /// </param>
        /// <returns>
        /// A new type collection containing those Types proceeding the
        /// containee type.
        /// </returns>
        public Types GetTypesThatProceedType(Type containee)
        {
            // Create a variable to use as the return value later.
            Types newTypes = new Types();

            // Create a new instance of the containee type in order that
            // we can access the PrePost collection for the Type.
            RuleSupport.PrePostAttribute prepostAttr =
(RuleSupport.PrePostAttribute)containee.GetCustomAttributes(typeof(RuleSupport.
PrePostAttribute), false)[0];

            // Loop through each PrePost entry so we can see those Types
            // the proceed it.
            foreach (RuleSupport.PrePost prepost in prepostAttr.PrePosts)
            {
                // If the Post item is contained within this collection and
                // the key is not already in the new collection, we can add
                // the Type to the new collection.
                if ((this.ContainsKey(prepost.Post)) &&
(!newTypes.ContainsKey(prepost.Post)))
                {
                    // Add this Post Type to the collection.
                    newTypes.Add((Type)this[prepost.Post]);
                }
            }

            // Return the new collection of types containing those requested.
            return newTypes;
        }
```

```
        /// <summary>
        /// The CreatePeriod method is a simple helper method that constructs a
new
        /// instance of the given type using the Types default constructor.
        /// </summary>
        /// <param name="type">
        /// The Type to construct a new instance of an object from.
        /// </param>
        /// <returns>
        /// The return value is the new instance of the type as its base Period
class.
        /// </returns>
        public static RuleSupport.Period CreatePeriod(Type type)
        {
            // Get access to the default constructor of the Type and invoke it
in
            // order to create a new instance of the Type.
            return (RuleSupport.Period)type.GetConstructor(new Type[] {
}).Invoke(new Type[] { });
        }

        #region IEnumerable<Type> Members

        /// <summary>
        /// The standard enumerator for a generic SortedList does not provide
simple
        /// iteration of the collection through a foreach(Type type in ...)
style
        /// statement which is why this new iterator has been defined.
        /// </summary>
        /// <returns></returns>
        public new IEnumerator<Type> GetEnumerator()
        {
            // Return the base classes Values enumerator.
            return base.Values.GetEnumerator();
        }

        #endregion
    }

}
```

11.3.3.11 WorkPattern Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use the these libraries for providing dynamically sizable collections.
using System.Collections.Generic;

// Use the Serialisation libraries to support the conversion of the
// workpattern into some form of streamed data.
using System.Runtime.Serialization;

// Use this library to provide a means of serialising the objects into
// XML for file storage and network transportation.
using System.Xml.Serialization;

/// <summary>
/// RuleSupport Namespace.
/// This namespace provides a range of functionality to support both the
/// compilation and execution of rules.
/// </summary>
namespace RuleSupport
{

    /// <summary>
    /// The WorkPatternCollection class is used to support the Testing
    /// class in carrying out the test. It will contain the built up work
    /// pattern derived by the Testing class.
```

```
    /// </summary>
    [Serializable(), XmlRoot("WorkPattern")]
    public class WorkPattern :
RuleSupport.DirtyList<RuleSupport.IPeriodElement>, ISerializable
    {

        RuleSupport.PeriodRequestDelegate periodRequestDelegate = null;
        RuleSupport.PeriodTypeRequestDelegate periodTypeRequestDelegate = null;

        public WorkPattern()
            : base()
        {
            periodRequestDelegate = new
RuleSupport.PeriodRequestDelegate(Period RequestPeriod);
            periodTypeRequestDelegate = new
RuleSupport.PeriodTypeRequestDelegate(Period_RequestPeriodType);
        }

        /// <summary>
        /// The Period_RequestPeriod method is a callback method
        /// for using by the various PeriodRequestDelegates
        /// stored within the WorkPatternCollection. It's task is
        /// to retrieve the IPeriodElement that is located
        /// relative to the source IPeriodElement and return it.
        /// </summary>
        /// <param name="source">
        /// The source IPeriodElement object whose relative is
        /// to be returned.
        /// </param>
        /// <param name="relativeIndex">
        /// The relativeIndex describes either a positive or
        /// negative position relative to the source
        /// IPeriodElement to return to the caller.
        /// </param>
        /// <returns>
        /// The return value is the IPeriodElement that is
        /// relative to the source.
        /// </returns>
        private RuleSupport.IPeriodElement
Period RequestPeriod(RuleSupport.IPeriodElement source, int relativeIndex)
        {
            RuleSupport.IPeriodElement tmp = null;
            int curIndex = IndexOf(source);

            if (relativeIndex == 0)
                tmp = (RuleSupport.IPeriodElement)this[curIndex];
            else if (relativeIndex < 0)
            {
                if (curIndex + relativeIndex >= 0) tmp =
(RuleSupport.IPeriodElement)this[curIndex + relativeIndex];
            }
            else
            {
                if (curIndex + relativeIndex <= Count - 1) tmp =
(RuleSupport.IPeriodElement)this[curIndex + relativeIndex];
            }

            return tmp;
        }

        /// <summary>
        /// The Period RequestPeriodType method is a callback method
        /// for using by the various PeriodTypeRequestDelegates
        /// stored within the WorkPatternCollection. It's task is
        /// to retrieve the IPeriodElement that is located
        /// relative to the source IPeriodElement, of a given type,
        /// and return it.
        /// </summary>
        /// <param name="source">
        /// The source IPeriodElement object of the given type whose
        /// relative is to be returned.
        /// </param>
        /// <param name="relativeIndex">
        /// The relativeIndex describes either a positive or
        /// negative position relative to the source
        /// IPeriodElement, of the given type to return to the caller.
```

```
        /// </param>
        /// <param name="periodType">
        /// The periodType describes the type of IPeriodElement who
        /// is located relative to the caller, to return.
        /// </param>
        /// <returns>
        /// The return value is the IPeriodElement, of the given type,
        /// that is relative to the source.
        /// </returns>
        private RuleSupport.IPeriodElement
Period_RequestPeriodType(RuleSupport.IPeriodElement source, int relativeIndex,
Type periodType)
        {
            RuleSupport.IPeriodElement tmp = null;
            int curIndex = IndexOf(source);
            int tmpCount = 0;
            if (relativeIndex == 0)
                tmp = (RuleSupport.IPeriodElement)this[curIndex];
            else if (relativeIndex < 0)
            {
                for (int i = curIndex - 1; i > 0; i--)
                {
                    if (this[i].GetType() == periodType)
                    {
                        tmpCount++;
                        if (tmpCount == relativeIndex)
                        {
                            tmp = (RuleSupport.IPeriodElement)this[i];
                            break;
                        }
                    }
                }
            }
            else
            {
                for (int i = curIndex + 1; i > Count; i++)
                {
                    if (this[i].GetType() == periodType)
                    {
                        tmpCount++;
                        if (tmpCount == relativeIndex)
                        {
                            tmp = (RuleSupport.IPeriodElement)this[i];
                            break;
                        }
                    }
                }
            }

            return tmp;
        }

        /// <summary>
        /// The Add method provides the ability to add an
        /// ITestable object to the collection whilst at the
        /// same time assigning the delegates with a method
        /// to handle the request of relative objects.
        /// </summary>
        /// <param name="Period">
        /// The ITestable to add to the collection.
        /// </param>
        /// <returns>
        /// The return value is the index of the ITestable
        /// within the collection.
        /// </returns>
        public virtual int Add(RuleSupport.ITestable Period)
        {
            Period.RequestPeriod += periodRequestDelegate;
            Period.RequestPeriodType += periodTypeRequestDelegate;
            base.Add((RuleSupport.IPeriodElement)Period);
            return base.Capacity - 1;
        }

        /// <summary>
        /// The Remove method provides the ability to remove
        /// an ITestable from the collection whilst at the
```

```
        /// same time removeing the delegate handlers.
        /// </summary>
        /// <param name="Period">
        /// The ITestable to be removed from the collection.
        /// </param>
        public virtual void Remove(RuleSupport.ITestable Period)
        {
            Period.RequestPeriod -= periodRequestDelegate;
            Period.RequestPeriodType -= periodTypeRequestDelegate;
            base.Remove((RuleSupport.IPeriodElement)Period);
        }

        /// <summary>
        /// The Remove method allows the removal of an item
        /// from the collection based upon the items index.
        /// </summary>
        /// <param name="PeriodIndex">
        /// The PeriodIndex describes the index of the period
        /// with the collection.
        /// </param>
        public void Remove(int PeriodIndex)
        {
            Remove(base[PeriodIndex]);
        }

        /// <summary>
        /// The ToXML method is designed to convert the work
        /// pattern into an XML structure representing it. This
        /// allows the calling system to be able to see how the
        /// rule engine has organised the data and to determine
        /// whether the data is organised correctly.
        /// </summary>
        /// <returns>
        /// The return value is a string representing the XML
        /// version of the work pattern.
        /// </returns>
        public String ToXML()
        {
            System.Xml.Serialization.XmlSerializer mySerializer = new
System.Xml.Serialization.XmlSerializer(typeof(WorkPattern));
            System.IO.MemoryStream myMemoryStream = new
System.IO.MemoryStream();
            System.IO.StreamWriter myStreamWriter = new
System.IO.StreamWriter(myMemoryStream);
            System.IO.StreamReader myStreamReader = new
System.IO.StreamReader(myMemoryStream);
            String myResult;

            mySerializer.Serialize(myStreamWriter, this);
            myMemoryStream.Seek(0, System.IO.SeekOrigin.Begin);
            myResult = myStreamReader.ReadToEnd();
            myStreamReader.Close();
            myStreamWriter.Close();
            myMemoryStream.Close();

            return myResult;
        }

        #region ISerializable Members

        public void GetObjectData(SerializationInfo info, StreamingContext
context)
        {
            info.SetType(typeof(WorkPattern));
        }

        #endregion

        /// <summary>
        /// The GetEndStack method return the stack of entries
        /// at the end of the hierachy of rules built up within
        /// this work pattern.
        /// </summary>
        /// <returns>
        /// The return value is the stack containing those entries
        /// at the end of the rule hierachy.
```

```
        /// </returns>
        public Stack<RuleSupport.Period> GetEndStack()
        {
            Stack<RuleSupport.Period> layers = new Stack<RuleSupport.Period>();

            if ((this.Count > 0) && (this[this.Count - 1] is
RuleSupport.Period))
                ((RuleSupport.Period)(this[this.Count -
1])).GetEndStack(layers);

            return layers;
        }

    }

}
```

11.3.3.12 WorkPeriod Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use the these libraries for providing dynamically sizable collections.
using System.Collections.Generic;

/// <summary>
/// RuleSupport Namespace.
/// This namespace provides a range of functionality to support both the
/// compilation and execution of rules.
/// </summary>
namespace RuleSupport
{

    /// <summary>
    /// This is the base WorkPeriod class used to describe
    /// a general work period. It is inherited by classes
    /// generated by the rule compiler and is designed for
    /// the purpose of carrying out some of the basic
    /// functionality that is expected from a standard work
    /// period.
    /// </summary>
    [Serializable()]
    public abstract class WorkPeriod : Period
    {

        private DateTime startTime;
        private DateTime finishTime;
        private TimeSpan workLength;
        private TimeSpan restLength;

        /// <summary>
        /// The constructor initialising the private member variables
        /// of this object to their default values.
        /// </summary>
        public WorkPeriod()
            : base()
        {
            startTime = new DateTime(0);
            finishTime = new DateTime(0);
            workLength = new TimeSpan();
            restLength = new TimeSpan();
        }

        /// <summary>
        /// The refreshDirtyValues method is designed to recalculate the
        /// start, finish, worklength and restlength times only if the
        /// contents of this period changes which would cause an effect
        /// on these times. Otherwise the same values as before can be
        /// used.
        /// </summary>
        private void refreshDirtyValues()
```

```
        {
            workLength = new TimeSpan();
            restLength = new TimeSpan();

            DirtyList<IPeriodElement> combinedList = Contains.GetCombined();

            if (combinedList.Count == 0) return;

            SortedList<string, IPeriodElement> sortedList =
combinedList.Sort(typeof(IPeriodElement).GetProperty("Start"));
            startTime = sortedList[sortedList.Keys[0]].Start;
            finishTime = sortedList[sortedList.Keys[sortedList.Count -
1]].Finish;
            sortedList = null;

            for (int i = 0; i < combinedList.Count; i++)
            {
                workLength += combinedList[i].WorkLength;
                restLength += combinedList[i].RestLength;
            }

            Contains.ClearDirty();
        }

        /// <summary>
        /// The Start property returns the earliest start
        /// time of the items stored within the work period.
        /// </summary>
        public override DateTime Start
        {
            get
            {
                if (Contains.IsDirty())
                    refreshDirtyValues();

                return startTime;
            }
        }

        /// <summary>
        /// The Finish property returns the latest finished
        /// time of the items stored within the work period.
        /// </summary>
        public override DateTime Finish
        {
            get
            {
                if (Contains.IsDirty())
                    refreshDirtyValues();

                return finishTime;
            }
        }

        /// <summary>
        /// The RestLength property returns the accumulated
        /// rest time of the items stored within the work period.
        /// </summary>
        public override TimeSpan RestLength
        {
            get
            {
                if (Contains.IsDirty())
                    refreshDirtyValues();

                return restLength;
            }
        }

        /// <summary>
        /// The WorkLength property returns the accumulated
        /// work time of the items stored within the work period.
        /// </summary>
        public override TimeSpan WorkLength
        {
            get
```

```
            {
                if (Contains.IsDirty())
                    refreshDirtyValues();

                return workLength;
            }
        }

    }

}
```

11.3.4 RuleTesting Namespace

11.3.4.1 Tester Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use the these libraries for providing dynamically sizable collections.
using System.Collections.Generic;

// Use the facilities provided by the rule support namespace.
using RuleSupport;

/// <summary>
/// RuleTesting Namespace.
/// This namespace defines the classes used to either perform Rule Testing
/// or to support the process of Rule Testing. The Tester class is a
/// sample testing class which can be used to test rules.
/// </summary>
namespace RuleTesting
{

    /// <summary>
    /// The Tester class provides a sample implementation of a class which
    /// is able to take test the compiled rules against the data by organising
    /// the data into a work pattern.
    /// </summary>
    public class Tester
    {

        /// <summary>
        /// This is the main test method for this approach to test the
        /// provided dataset against the work and rest periods.
        /// </summary>
        /// <param name="PeriodTypes">
        /// The PeriodTypes array provides the work and rest periods that
        /// can be used to both devise the work pattern and test the rules
        /// with.
        /// </param>
        /// <param name="dataset">
        /// The dataset parameter provides the dataset to test against the
        /// different rules types..
        /// </param>
        /// <param name="workPattern">
        /// The workPattern parameter provides a work pattern for the testing
        /// method to start from. This parameter should be null if a new
        /// work pattern is to be constructed.
        /// </param>
        /// <returns>
        /// This method returns a boolean value, true for success and false
        /// for failure.
        /// </returns>
        public bool Test(Type[] periodTypes, RuleSupport.IPeriodElement[]
dataset, ref WorkPattern workPattern)
        {
            // First of all, if no data is provided, or no PeriodTypes, we
            // can give up now.
            if (dataset.Length == 0) return true;
            if (periodTypes.Length == 0) return false;

            // If a workPattern has not been provided, is null, we should
            // create a new WorkPattern.
            if (workPattern == null)
                workPattern = new WorkPattern();

            // Place the Types in a Types object for ease of access and to
            // gain from the benefits provided with some of the Get methods
            // in querying the Types at various times throughout the testing
            // process.
            Types Periods = new Types(periodTypes);
```

```
            // Now we need to iterate through each of the elements of data
            // provided and construct a work pattern based upon the metadata
            // provided within the rules and from the data itself.
            foreach (RuleSupport.IPeriodElement data in dataset)
            {

                // Use this variable to monitor the success in adding this
                // element of data to the work pattern.
                bool returnVal = false;

                // If the work pattern is currently empty, we need to start
                // from scratch. This section of the Test method should only
                // get reached with the first piece of data, if the work
                // pattern provided was null, otherwise it shouldn't get used.
                if (workPattern.Count == 0)
                {

                    // Get a list of those types that can contain this type of
                    // data.
                    Types types =
Periods.GetTypesThatContainType(data.GetType());

                    // Loop through these types, looking for one that will be
                    // able to store the given element of data without breaking
                    // the rules.
                    foreach (Type type in types)
                    {
                        // Create an instance of the current Type.
                        RuleSupport.Period period = Types.CreatePeriod(type);

                        // Add the element of data to the period.
                        period.Add(data);

                        // Add the period to the work pattern.
                        workPattern.Add(period);

                        // Now test the work pattern, with the single period of
                        // time, to see if this breaks the rules as it
currently
                        // stands.
                        if (DoTest(workPattern))
                        {
                            // If we haven't broken the rules, we need to start
                            // adding the parental periods (those which this
                            // period must belong) to the work pattern and
ensure
                            // this does not break the rules.
                            if (RecursivelyAddParentalContainers(Periods,
workPattern, period))
                            {
                                // If we reach this point, the work pattern has
                                // been constructed with the single eleemnt of
data
                                // and includes the appropriate rule hierachy
to
                                // support it. This is it for the first element
of
                                // data, on to the next.
                                returnVal = true;
                                break;
                            }
                        }

                        // If we reach this point, either the data does not fit
within
                        // the work pattern or the parental rule hierachy does
not work
                        // with the element of data and the chosen period. We
can
                        // remove it from the work pattern and try the next, or
fail.
                        workPattern.Remove(period);
                    }


                }
```

```
                // If we've reach the else, we are dealing with any element of
data
                // except for the first element, so a work pattern exists and
we need
                // try and either add this element of data to the existing work
pattern
                // or modify the pattern to accomodate this element of data.
                else
                {

                    // In order to look at the complex rule hierachy that has
probably
                    // built up in the work pattern, we can construct a stack
based
                    // view of the end of the hierachy, from the outer most
elements,
                    // place in the Stack first, through to the smallest form
of detail
                    // that is placed in last.
                    Stack<RuleSupport.Period> layers =
workPattern.GetEndStack();

                    // Loop through the various layers of the stack until we
find a
                    // solution at the most appropriate layer. This would be
starting
                    // at the smallest level of detail, then working our way
outward.
                    while ((layers.Count > 0) && (!returnVal))
                    {
                        // The currentLayer variable is used to help
iteratively look
                        // through the work patterns hierachy. Pop the first
entry
                        // from the stack into our variables, which will also
remove
                        // it from the stack.
                        RuleSupport.Period currentLayer = layers.Pop();

                        // If this layer can contain types of data like the one
we
                        // are trying to add to the work pattern, we can start
by
                        // giving it a try.
                        if (currentLayer.CanContain(data))
                        {
                            // Add the element of data to the work pattern.
                            currentLayer.Add(data);

                            // Now lets test to see if adding the element of
data to
                            // the work pattern works.
                            if (!DoTest(workPattern))
                            {
                                // The test failed so we can remove this
element of
                                // data, although it was worth a try.
                                currentLayer.Remove(data);
                            }
                            else
                            {
                                // The test succeeded so we can set our flag to
true
                                // and we will break out from the while loop we
are
                                // currently in.
                                returnVal = true;
                            }
                        }

                        // The current layer cannot directly contain this type
of data
                        // however we can try some other approaches at this
level to
                        // see if can add additional periods in order to get
```

```
this data
                        // element into the work pattern.
                        else
                        {

                            // We have to be careful from here on as we are
making
                            // calls to recursive methods which have the
potential of
                            // overflowing the stack.

                            // First of all, lets try and add periods into the
current
                            // layer of the work pattern vertical to the
current layer.
                            if (RecursivelyAddInnerPeriods(Periods,
workPattern, currentLayer, data))
                                returnVal = true;

                            // If that fails, lets try and add periods to the
work
                            // pattern horizontal to the current layer.
                            else if (RecursivelyAddPostPeriods(Periods,
workPattern, (layers.Count > 0 ? layers.Peek() : null), currentLayer, data))
                                returnVal = true;

                        }
                    }

                }

                // If we reach this stage and our flag still indicates a false
value,
                // we've tried all we can and just cannot add this element of
data to
                // the work pattern without breaking the rules.
                if (!returnVal)
                    return false;
            }

            // If we reach this point, we must have added all of the elements
of data
            // to the work pattern successfully and can return a positive
result to
            // the caller.
            return true;
        }

        /// <summary>
        /// The RecursivelyAddParentalContainers method has been designed to
        /// start with one of the smallest periods of time and recursively add
        /// them parental style periods until the greatest level of period
        /// exists in the hierachy.
        /// </summary>
        /// <param name="Periods">
        /// The collection of period types (compiled rules) that can be used in
        /// constructing the work pattern.
        /// </param>
        /// <param name="workPattern">
        /// The overarching work pattern object that forms the basis of the
tree
        /// structure with the various instancing of periods branching from it.
        /// </param>
        /// <param name="current">
        /// The current period from which a parent may need to be created.
        /// </param>
        /// <returns>
        /// A boolean value indicating the success of the recursive method.
        /// </returns>
        private bool RecursivelyAddParentalContainers(Types Periods,
WorkPattern workPattern, RuleSupport.Period current)
        {
            // Obtain a string array from the ContainWithin property of the
            // current period which will provide a list of all Types that can
            // contain this type within them.
            string[] containedWithin = current.ContainWithin.GetTypes();
```

```
            // If there are no elements in the array it means we have reached
            // the outer most layer of the rule hierachy and can return
            // success.
            if (containedWithin.Length == 0)
                return true;

            // Otherwise, there are still containers for us to reside in and
            // we must attempt to add ourselves to these containers.
            else
            {
                // First of all, remove the current entry from the root of the
                // work pattern.
                workPattern.Remove(current);

                // Loop through each of the return strings representing the
                // names of the types that we can reside in.
                foreach (string typeName in containedWithin)
                {
                    // Create a new instance of the type, ready for use.
                    RuleSupport.Period container =
Types.CreatePeriod(Periods[typeName]);

                    // Add the current period to the new period, as its
                    // parent.
                    container.Add(current);

                    // Add the new container to the work pattern ready for
                    // testing.
                    workPattern.Add(container);

                    // Carry out the rule testing process on the work pattern.
                    if (!DoTest(workPattern))
                    {
                        // If we fail we should remove the container from the
                        // work Pattern, ready for moving on to the next one.
                        workPattern.Remove(container);
                    }
                    else
                    {
                        // If we succeed we must recursively call this method,
                        // this time look for whether the container has a
parent
                        // container that it must reside within.
                        if (RecursivelyAddParentalContainers(Periods,
workPattern, container))
                        {
                            // If the result of the recursion is true, we can
                            // return true;
                            return true;
                        }
                    }
                }

                // If we reach this point, we have tried each of the
                // potential containers, none of which are suitable, and we
                // have to return a fail.
                return false;
            }
        }

        /// <summary>
        /// The RecursivelyAddInnerPeriods method is a recursive method that
        /// attempts to add periods into the current period object, in the
        /// search for a period that is able to store the data element
provided.
        /// </summary>
        /// <param name="Periods">
        /// The Periods argument provides the collection of period Types used
        /// for selecting the appropriate periods that can be stored within the
        /// current period.
        /// </param>
        /// <param name="workPattern">
        /// The workPattern arguments provides the work pattern object which
        /// is storing the rule hierachy as it currently stands.
        /// </param>
```

```
        /// <param name="currentPeriod">
        /// The currentPeriod describes the period to try and add new period
        /// types within.
        /// </param>
        /// <param name="data">
        /// The data argument provides the data element to try and add to an
        /// inner period of the current period.
        /// </param>
        /// <returns>
        /// The return value indicates the success of trying to locate an
        /// inner period for the current period.
        /// </returns>
        private bool RecursivelyAddInnerPeriods(Types Periods, WorkPattern
workPattern, RuleSupport.Period currentPeriod, RuleSupport.IPeriodElement data)
        {
            // First of all, lets get those types that can be contained within
            // the current type, from the Period collection.
            Types containedTypes =
Periods.GetTypesContainedInType(currentPeriod.GetType(), (currentPeriod.Count >
0 ? currentPeriod[currentPeriod.Count - 1].GetType() : null));

            // Secondly, we want to narrow down the types available to only
            // those that can follow the previous period, if there is one,
            // otherwise we'll look at the complete set provided.
            Types containedAndProceedingTypes = null;
            if (currentPeriod.Count > 0)
                containedAndProceedingTypes =
containedTypes.GetTypesThatProceedType(currentPeriod[currentPeriod.Count -
1].GetType());
            else
                containedAndProceedingTypes = containedTypes;

            // Now lets loop through the valid subset of periods that can be
            // contained within the current Period.
            foreach (Type type in containedAndProceedingTypes)
            {

                // We need to create a new instances of this period type.
                RuleSupport.Period period = Types.CreatePeriod(type);

                // Add the period to the current period, as it will comes
                // within it in the work pattern hierachy.
                currentPeriod.Add(period);

                // Can this new period contain the data element directly?
                // If it can we can add the element of data and test the
                // work pattern to see if it works.
                if (period.CanContain(data))
                {
                    // Add the elment of data to the period.
                    period.Add(data);

                    // Test the work pattern.
                    if (DoTest(workPattern))
                    {
                        // If the work pattern test succeeds, this new
                        // period is able to store the element of data
                        // and the rules testing is successful so we can
                        // return success.
                        return true;
                    }
                    else
                    {
                        // If the test fails we can remove the element of
                        // data from this period and try something else.
                        period.Remove(data);
                    }
                }
                else
                {
                    // If we reach this point it is because the period
                    // cannot contain an element of data like and we need
                    // to look with the new period or after it in order to
                    // find a period type that can contain this type of data.
                    // If either of these approaches succeeds, we can return
                    // success, otherwise we'll have to try the next type of
```

```
                    // period.

                    // First of all, try recursively calling this method only
                    // with the new period as the current period.
                    if (RecursivelyAddInnerPeriods(Periods, workPattern,
period, data))
                        return true;

                    // If that fails, try looking at those periods that may
                    // be able to follow this periods in the hierachy.
                    else if (RecursivelyAddPostPeriods(Periods, workPattern,
currentPeriod, period, data))
                        return true;

                }

                // Reaching this points means that the period we are looking
                // add does not fit with the current work pattern and data
                // combination and we should try the next.
                currentPeriod.Remove(period);

            }

            // If we reach this point it means we cannot find a solution by
            // looking down this path, we may have been luck when navigating
            // back up the recursive call stack.
            return false;
        }

        /// <summary>
        /// The RecursivelyAddPostPeriods method is a recursive method that
        /// looks horizontal at period types that reside at the same layer
        /// as the current period, to see if there are others at the same
        /// layer that can be added to the rule hierachy to help build a
        /// successful work pattern.
        /// </summary>
        /// <param name="Periods">
        /// The Periods argument provides the collection of period Types used
        /// for selecting the appropriate periods that can be stored at the
same
        /// layer as the current period.
        /// </param>
        /// <param name="workPattern">
        /// The workPattern argument provides the work pattern object which
        /// is storing the rule hierachy as it currently stands.
        /// </param>
        /// <param name="parentPeriod">
        /// The parentPeriod argument provides the parentPeriod to which the
        /// new periods that reside at the same level as the current period
        /// will be added. This argument can be null if the work pattern
        /// should be used as the parent.
        /// </param>
        /// <param name="currentPeriod">
        /// The currentPeriod describes the period to try and add new period
        /// types at the same level.
        /// </param>
        /// <param name="data">
        /// The data argument provides the data element to try and add to the
        /// work pattern
        /// </param>
        /// <returns>
        /// The return value indicates the success of trying to locate a
        /// peer for the current period.
        /// </returns>
        private bool RecursivelyAddPostPeriods(Types Periods, WorkPattern
workPattern, RuleSupport.Period parentPeriod, RuleSupport.Period currentPeriod,
RuleSupport.IPeriodElement data)
        {
            // Obtain a collection of period types that can follow the
            // current type within a work pattern. This helps to narrow
            // down our search early on.
            Types types =
Periods.GetTypesThatProceedType(currentPeriod.GetType());

            // Loop through each of the types in the collection.
            foreach (Type type in types)
```

```
            {
                // Create a new instance of the type to for use in our
                // work pattern.
                RuleSupport.Period period = Types.CreatePeriod(type);

                // If a parent is provided then we should be adding the
                // new period to the parent, otherwise we should be
                // adding it to the root of the work pattern.
                if (parentPeriod == null)
                    workPattern.Add(period);
                else
                    parentPeriod.Add(period);

                // Can this period contain the type of data we are
                // trying to add to our work pattern.
                if (period.CanContain(data))
                {
                    // Add the data to the period.
                    period.Add(data);

                    // Test the work pattern as it now stands to see
                    // if the data being contained within our new
                    // period produces a succesful result.
                    if (!DoTest(workPattern))
                    {
                        // The result failed indicated that the data
                        // does not fit in to the work plan in this way
                        // and we need to try something else.
                        period.Remove(data);
                    }
                    else
                    {
                        // The test was successful and we can return
                        // with a successful response up the call stack.
                        return true;
                    }
                }

                // If we reach here it is because we have added the new
                // period to the work pattern however it cannot contain
                // this type of data and therefore we need to try adding
                // additional periods to the work pattern to contain the
                // data.
                else
                {

                    // First of all, try recursively calling this method only
                    // with the new period as the current period.
                    if (RecursivelyAddInnerPeriods(Periods, workPattern,
period, data))
                        return true;

                    // If that fails, try looking at those periods that may
                    // be able to follow this periods in the hierachy.
                    else if((period.Contains.Count == 0) &&
(RecursivelyAddPostPeriods(Periods, workPattern, parentPeriod, period, data)))
                        return true;
                }

                // Reaching this points means that the period we are looking
                // add does not fit with the current work pattern and data
                // combination and we should try the next.
                if (parentPeriod == null)
                    workPattern.Remove(period);
                else
                    parentPeriod.Remove(period);
            }

            // If we reach this point it means we cannot find a solution by
            // looking down this path, we may have been luck when navigating
            // back up the recursive call stack.
            return false;
        }

        /// <summary>
        /// The DoTest method carries out a test on the the given work
```

```
        /// period.
        /// </summary>
        /// <param name="workPattern">
        /// The workPattern parameter is the pattern of work to be tested.
        /// </param>
        /// <returns>
        /// The return value is a boolean value returning the result of
        /// the test. True if the test was successful, or false if it was
        /// not.
        /// </returns>
        private bool DoTest(WorkPattern workPattern)
        {
            // If we are the top level workPattern, we wont have
            // implemented the ITestable interface so we don't test it,
            // otherwise, we call our own test method first.
            if (workPattern is RuleSupport.ITestable)
                if (!((RuleSupport.ITestable)workPattern).Test())
                    return false;

            // We need to test all periods within this period,
            // including those periods contained within those periods,
            // recursively.
            for(int i = 0; i < workPattern.Count; i++)
                if (!DoTest((WorkPattern)workPattern[i]))
                    return false;

            // We've made it this far and therefore we can pass the test.
            return true;

        }

    }

}
```

11.3.5 RuleCompilerTestbench Namespace

11.3.5.1 StopWatch Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

/// <summary>
/// RuleCompilerTestbench Namespace
/// Contains the classes and functionality to support the testing of the
/// RuleCompiler library and its various functionality.
/// </summary>
namespace RuleCompilerTestbench
{

    /// <summary>
    /// The class provides a "stop watch" for applications requiring
    /// accurate timing measurements. It has been designed to be fairly
    /// cross platform with an implementation for the windows platform
    /// as well as for linux running mono.
    /// </summary>
    public class StopWatch
    {

        /// <summary>
        /// The QueryPerformanceCounter method is a platform invocation
        /// style method to the windows based high performance timer
        /// method. This gives much higher accuracy than the built in
        /// .NET Framework TimeSpan and DateTime classes.
        /// </summary>
        /// <param name="lpPerformanceCount">
        /// The value of the high performance timer is stored in the
        /// reference parameter lpPerformanceCount.
        /// </param>
        /// <returns>
        /// The return value is a boolean indicating the success of
        /// the method call.
        /// </returns>
        [System.Runtime.InteropServices.DllImport("KERNEL32")]
        private static extern bool QueryPerformanceCounter(ref long
lpPerformanceCount);

        /// <summary>
        /// The QueryPerformanceFrequency method is a platform invocation
        /// style method to the windows based high performance timer
        /// method. The outcome of this method indicates how many
        /// lpPerformanceCount's there are in a second when used with
        /// QueryPerformanceCounter.
        /// </summary>
        /// <param name="lpFrequency">
        /// The number of lpPerformanceCount's there are in a second when
        /// used with QueryPerformanceCounter.
        /// </param>
        /// <returns>
        /// The return value is a boolean indicating the success of
        /// the method call.
        /// </returns>
        [System.Runtime.InteropServices.DllImport("KERNEL32")]
        private static extern bool QueryPerformanceFrequency(ref long
lpFrequency);

        /// <summary>
        /// The gettimeofday method is a platform invocation style
        /// method to the posix compliant high performance timer
        /// method. This method is used when running the system
        /// on Mono using Linux or another Mono supported OS.
        /// </summary>
        /// <param name="tv">
        /// The reference parameter tv is set by the method to
        /// be the number of seconds and microseconds since the
        /// start of 01/01/1970.
```

```
        /// </param>
        /// <param name="ignore">
        /// The paramter ignore is not used in this instance
        /// and should have IntPtr.Zero passed in.
        /// </param>
        /// <returns>
        /// This method should always return 0.
        /// </returns>
        [System.Runtime.InteropServices.DllImport("MonoPosixHelper", EntryPoint
= "Mono_Posix_Syscall_gettimeofday")]
        private static extern int gettimeofday(out Timeval tv, IntPtr ignore);

        /// <summary>
        /// The Timeval structure is used by the gettimeofday
        /// platform invocation method to store its return
        /// values.
        /// </summary>
        private struct Timeval
        {

            /// <summary>
            /// The variable tv_sec is used to store the number
            /// of whole seconds since the start of 01/01/1970.
            /// </summary>
            public long tv_sec;

            /// <summary>
            /// The variable tv_usec is used to store the number
            /// of microseconds to accompany tv sec.
            /// </summary>
            public long tv_usec;

            /// <summary>
            /// The ToTicks method will convert the values
            /// stored in tv sec and tv usec into ticks that
            /// can be used for comparison with TimeSpan and
            /// DateTime values.
            /// </summary>
            /// <returns>
            /// The number of ticks stored in tv sec and
            /// tv usec.
            /// </returns>
            public long ToTicks()
            {
                return (long)((10000000d * ((double)tv sec)) +
(((double)tv usec) * 10d));
            }

            /// <summary>
            /// The ToTimeSpan method will convert the values
            /// stored in tv sec and tv usec into a TimeSpan
            /// structure for comparison with other normal
            /// TimeSpan and DateTime operations.
            /// </summary>
            /// <returns>
            /// The TimeSpan representation of the values
            /// stored in tv sec and tv usec.
            /// </returns>
            public TimeSpan ToTimeSpan()
            {
                return new TimeSpan(ToTicks());
            }

            /// <summary>
            /// The ToDateTime method will convert the values
            /// stored in tv_sec and tv_usec into a DateTime
            /// structure for comparison with other normal
            /// TimeSpan and DateTime operations.
            /// </summary>
            /// <returns>
            /// The DateTime representation of the values
            /// stored in tv_sec and tv_usec.
            /// </returns>
            public DateTime ToDateTime()
            {
                return new DateTime(1970, 1, 1, 0, 0, 0,
```

```
0).AddSeconds(tv sec).AddTicks(tv usec * 10);
            }

        };

        /// <summary>
        /// The TimerTypes enum is used to help describe the
        /// type of timer that is being used.
        /// </summary>
        private enum TimerTypes : int
        {
            /// <summary>
            /// HighPerformanceWindows is used to illustrate
            /// that the window performance counter is being
            /// used by the stopwatch.
            /// </summary>
            HighPerformanceWindows = 0,

            /// <summary>
            /// HighPerformancePosix is used to illustrate
            /// that the Posix high performance time is
            /// being used by the stopwatch.
            /// </summary>
            HighPerformancePosix = 1,

            /// <summary>
            /// CLRTicks is used to illustrate that the
            /// .NET Frameworks DateTime.Now is being
            /// used by the stopwatch to take
            /// measurements.
            /// </summary>
            CLRTicks = 2
        };

        /// <summary>
        /// The timerType variable is used to determine which
        /// timer is being used to take stopwatch measurements.
        /// </summary>
        private TimerTypes timerType;

        private long totalCount = 0;
        private long lastCount = 0;

        // For HighPerformanceWindows and CLRTicks, the
        // following three variables are used to store and
        // track the stopwatches measurements.
        private long startCount = 0;
        private long stopCount = 0;
        private long freq = 0;

        // For HighPerformancePosix, the following two
        // variables are being used to keep track of the
        // stopwatches measurements.
        private Timeval startTimeval;
        private Timeval stopTimeval;

        // The following four variables are used to keep
        // track of memory usage.
        private long memStartCount = 0;
        private long memStopCount = 0;
        private long memTotalCount = 0;
        private long memLastCount = 0;

        /// <summary>
        /// The constructor is used to initialise the
        /// class by determining which of the timing
        /// methods should be used and setting up some
        /// basic values associated with them.
        /// </summary>
        public StopWatch()
        {
            freq = 0;
            try
            {
                QueryPerformanceFrequency(ref freq);
                timerType = TimerTypes.HighPerformanceWindows;
```

```
            }
            catch
            {
                try
                {
                    gettimeofday(out startTimeval, IntPtr.Zero);
                    timerType = TimerTypes.HighPerformancePosix;
                }
                catch
                {
                    timerType = TimerTypes.CLRTicks;
                }
            }
        }

        /// <summary>
        /// The Start method starts the stopwatch. The
        /// first set of measurements are recorded
        /// depending upon which time of timer is being
        /// used. Once the measurements are taken, the
        /// timer does nothing until the Stop method is
        /// called.
        /// </summary>
        public void Start()
        {
            lastCount = 0;

            if (timerType == TimerTypes.HighPerformanceWindows)
            {
                QueryPerformanceCounter(ref startCount);
            }
            else if (timerType == TimerTypes.HighPerformancePosix)
            {
                gettimeofday(out startTimeval, IntPtr.Zero);
                startCount = startTimeval.ToTicks();
            }
            else
            {
                startCount = DateTime.Now.Ticks;
            }

            memLastCount = 0;
            memStartCount = ApplicationPhysicalMemoryUsage;
        }

        /// <summary>
        /// The Stop method is used to stop a started
        /// stopwatch. The stopwatch will take another
        /// measurement of the current system time with
        /// the appropriate timing method.
        /// </summary>
        public void Stop()
        {
            if (timerType == TimerTypes.HighPerformanceWindows)
            {
                QueryPerformanceCounter(ref stopCount);
            }
            else if (timerType == TimerTypes.HighPerformancePosix)
            {
                gettimeofday(out stopTimeval, IntPtr.Zero);
                stopCount = stopTimeval.ToTicks();
            }
            else
            {
                stopCount = DateTime.Now.Ticks;
            }

            memStopCount = ApplicationPhysicalMemoryUsage;

            lastCount = stopCount - startCount;
            totalCount += lastCount;

            memLastCount = memStopCount - memStartCount;
            memTotalCount += memLastCount;
        }
```

```
        /// <summary>
        /// The Reset method will reset the timer back
        /// to a zero state so timing can begin from
        /// scratch.
        /// </summary>
        public void Reset()
        {
            lastCount = 0;
            totalCount = 0;

            memLastCount = 0;
            memTotalCount = 0;
        }

        /// <summary>
        /// The LastMemory property will return the
        /// amount of memory change between the
        /// stopwatch last starting and stopping.
        /// </summary>
        public long LastMemory
        {
            get
            {
                return memLastCount;
            }
        }

        /// <summary>
        /// The LastTime property will return the
        /// ammount of time that elapsed since the
        /// stopwatch last started and stopped. The
        /// method for calculating this depends on
        /// the timing method used.
        /// </summary>
        public TimeSpan LastTime
        {
            get
            {
                if (timerType == TimerTypes.HighPerformanceWindows)
                {
                    return new TimeSpan((long)(((double)lastCount /
(double)freq) * 10000000d));
                }
                else if (timerType == TimerTypes.HighPerformancePosix)
                {
                    return new TimeSpan(lastCount);
                }
                else
                {
                    return new TimeSpan(lastCount); ;
                }
            }
        }

        /// <summary>
        /// The TotalMemory method is used to return
        /// the total amount of memory change since the
        /// stopwatch was last constructed or reset.
        /// </summary>
        public long TotalMemory
        {
            get
            {
                return memTotalCount;
            }
        }

        /// <summary>
        /// The TotalTimeSpan method is used to return
        /// the total amount of time accumulated since
        /// the stopwatch was last constructed or reset.
        /// </summary>
        public TimeSpan TotalTimeSpan
        {
            get
            {
```

```
                if (timerType == TimerTypes.HighPerformanceWindows)
                {
                    return new TimeSpan((long)(((double)totalCount /
(double)freq) * 10000000d));
                }
                else if (timerType == TimerTypes.HighPerformancePosix)
                {
                    return new TimeSpan(totalCount);
                }
                else
                {
                    return new TimeSpan(totalCount); ;
                }
            }
        }

        /// <summary>
        /// The static MemoryToString method is used to
        /// convert the number of bytes provided in a
        /// string representing the number of bytes in
        /// a more human readable form.
        /// </summary>
        /// <param name="bytes">
        /// The number of bytes to be convereted into a
        /// string representation.
        /// </param>
        /// <returns>
        /// The return value is a string representation of
        /// the number of bytes passed in.
        /// </returns>
        public static string MemoryToString(long bytes)
        {
            long theMem = bytes;
            string theUnit = "bytes";
            if (theMem > 1024)
            {
                theMem = theMem / 1024;
                theUnit = "KB";
            }
            if (theMem > 1024)
            {
                theMem = theMem / 1024;
                theUnit = "MB";
            }
            if (theMem > 1024)
            {
                theMem = theMem / 1024;
                theUnit = "GB";
            }
            return theMem.ToString() + theUnit;
        }

        /// <summary>
        /// The ToString method overrides the base
        /// class ToString method to display the
        /// processing time and memory usage recorded
        /// by the stopwatch.
        /// </summary>
        /// <returns>
        /// The return value is a string displaying the
        /// number of seconds and amount of memory
        /// change recorded by the stopwatch.
        /// </returns>
        public override string ToString()
        {

            return String.Format("{0:F2} seconds, {1} memory",
TotalTimeSpan.TotalSeconds, MemoryToString(memTotalCount));
        }

        /// <summary>
        /// The ApplicationPhysicalMemoryUsage property
        /// is used to isolate the means of determine
        /// application memory usage in order that an
        /// alternative method is used later, if
```

```
        /// appropriate.
        /// </summary>
        private long ApplicationPhysicalMemoryUsage
        {
            get
            {
                return GC.GetTotalMemory(true);
            }
        }

    }

}
```

11.3.5.2 GraphImage Class

```
// Use the following two libraries for providing graphical interface
// capabilities to the class.
using System.Drawing;

// Use the these libraries for providing dynamically sizable collections.
using System.Collections.Generic;

/// <summary>
/// RuleCompilerTestbench Namespace
/// Contains the classes and functionality to support the testing of the
/// RuleCompiler library and its various functionality.
/// </summary>
namespace RuleCompilerTestbench
{

    /// <summary>
    /// The GraphImage class provides a simple means of drawing a basic
    /// line graph from results provided by the stopwatch.
    /// </summary>
    public class GraphImage
    {

        /// <summary>
        /// The GraphPoint describes an individual point that will be
        /// displayed on the line graph.
        /// </summary>
        public struct GraphPoint
        {
            public Point Position;
            public string Description;

            public GraphPoint(Point position, string description)
            {
                Position = position;
                Description = description;
            }
        }

        // The following variables simple describes the colours and
        // typical formating that will be used when drawing the graph.
        private Brush backColour = Brushes.Black;
        private Pen gridColour = Pens.Green;
        private Pen lineColour = Pens.White;
        private Brush textColour = Brushes.White;
        private Font font = TestBench.DefaultFont;

        // The virtual variables are used to translate between the
        // scale the graph is expected to used in relation to the
        // GraphPoint data and the height and width of the image
        // to be drawn.
        private Size virtualSize;
        private Point virtualGridFrequency;

        // The points list is used to store all of the points that
        // will be drawn onto the graph.
        private List<GraphPoint> points;
```

```
        // The bitmap private variable and public property provide
        // a means of getting the image of the graph once drawn,
        // by not setting it as it needs to be read-only.
        private Bitmap bitmap;
        public Bitmap Bitmap
        {
            get
            {
                return bitmap;
            }
        }

        // The VirtualSize property provides a means of changing
        // the size of graph to be produced.
        public Size VirtualSize
        {
            get
            {
                return virtualSize;
            }
            set
            {
                virtualSize = value;
                Redraw();
            }
        }

        /// <summary>
        /// The GraphImage constructor provides a means of
        /// constructing the GraphImage class with all the
        /// necessary values passed in that are required
        /// to be able to draw the graph.
        /// </summary>
        /// <param name="actualSize">
        /// The actualSize parameter describes the width
        /// and height of the bitmap to produce.
        /// </param>
        /// <param name="virtualSize">
        /// The virtualSize parameter describes the size of
        /// the grid to produce based on values relative to
        /// the GridPoints themselves.
        /// </param>
        /// <param name="virtualGridFrequency">
        /// The virtualGridFrequency describes the size
        /// frequency by which the gridlines should be draw
        /// relative to the GridPoint data, not the actual
        /// image size.
        /// </param>
        public GraphImage(Size actualSize, Size virtualSize, Point
virtualGridFrequency)
        {
            bitmap = new Bitmap(actualSize.Width, actualSize.Height);
            this.virtualSize = virtualSize;
            this.virtualGridFrequency = virtualGridFrequency;
            this.points = new List<GraphPoint>();

            this.font = new Font(TestBench.DefaultFont.FontFamily, 6);
            Redraw();
        }

        /// <summary>
        /// The TranslatePoint method will convert a point
        /// from virtual grid space to actual image space.
        /// </summary>
        /// <param name="point">
        /// The point to convert to image space from virtual
        /// grid space.
        /// </param>
        /// <returns>
        /// The return value is the point in image space,
        /// not virtual grid space.
        /// </returns>
        private Point TranslatePoint(Point point)
        {
            return new Point((point.X == 0 ? 0 : (int)((float)bitmap.Width /
```

```
((float)virtualSize.Width / (float)point.X))), (point.Y == 0 ? 0 :
(int)((float)bitmap.Height / ((float)virtualSize.Height / (float)point.Y))));
        }

        /// <summary>
        /// The Redraw method clears the bitmap and redraws
        /// all of its data using the GridPoints provided.
        /// </summary>
        public void Redraw()
        {
            Graphics graphics = Graphics.FromImage(bitmap);
            graphics.FillRectangle(backColour, 0, 0, bitmap.Width,
bitmap.Height);
            graphics.DrawRectangle(gridColour, 0, 0, bitmap.Width,
bitmap.Height);

            Point transLineFrequency = TranslatePoint(virtualGridFrequency);

            for (int i = transLineFrequency.X; i < bitmap.Width; i +=
transLineFrequency.X)
                graphics.DrawLine(gridColour, new Point(i, 0), new Point(i,
bitmap.Height));

            for (int i = transLineFrequency.Y; i < bitmap.Height; i +=
transLineFrequency.Y)
                graphics.DrawLine(gridColour, new Point(0, i), new
Point(bitmap.Width, i));

            for (int i = 0; i < points.Count; i++)
            {
                if (i == 0)
                    DrawPoint(graphics, points[i]);
                else
                    DrawPoint(graphics, points[i - 1], points[i]);
            }

            graphics.Dispose();
        }

        /// <summary>
        /// The DrawPoint method draws a point onto the image.
        /// </summary>
        /// <param name="graphics">
        /// The Graphics object to use to carry out the drawing.
        /// This is already setup to draw to our bitmap.
        /// </param>
        /// <param name="lastPoint">
        /// The lastPoint parameter defines where the last point
        /// was located so that we can draw a line from it to
        /// this point.
        /// </param>
        /// <param name="thisPoint">
        /// The thisPoint parameter defines where this point needs
        /// to be drawn, along with the text that is associated
        /// with it.
        /// </param>
        private void DrawPoint(Graphics graphics, GraphPoint lastPoint,
GraphPoint thisPoint)
        {
            Point transPoint = TranslatePoint(thisPoint.Position);
            graphics.DrawEllipse(lineColour, transPoint.X - 1, bitmap.Height -
transPoint.Y - 1, 3, 3);
            graphics.DrawString(thisPoint.Description, font, textColour, new
PointF(transPoint.X + 1, bitmap.Height - transPoint.Y -
graphics.MeasureString("Ag", font).Height));

            Point transLastPoint = TranslatePoint(lastPoint.Position);
            graphics.DrawLine(lineColour, new Point(transLastPoint.X,
bitmap.Height - transLastPoint.Y), new Point(transPoint.X, bitmap.Height -
transPoint.Y));
        }

        /// <summary>
        /// This DrawPoint method draws a point onto the image
        /// but does draw any lines.
        /// </summary>
```

```
        /// <param name="graphics">
        /// The Graphics object to use to carry out the drawing.
        /// This is already setup to draw to our bitmap.
        /// </param>
        /// <param name="thisPoint">
        /// The thisPoint parameter defines where this point needs
        /// to be drawn, along with the text that is associated
        /// with it.
        /// </param>
        private void DrawPoint(Graphics graphics, GraphPoint thisPoint)
        {
            Point transPoint = TranslatePoint(thisPoint.Position);
            graphics.DrawEllipse(lineColour, transPoint.X - 1, bitmap.Height -
transPoint.Y - 1, 3, 3);
            graphics.DrawString(thisPoint.Description, font, textColour, new
PointF(transPoint.X + 1, bitmap.Height - transPoint.Y -
graphics.MeasureString("Ag", font).Height));
        }

        /// <summary>
        /// The AddPoint method will add the given GraphPoint
        /// into the list to be drawn, then redraws the image to
        /// include the point.
        /// </summary>
        /// <param name="point">
        /// The point parameter provides a GraphPoint to be
        /// added to the list.
        /// </param>
        public void AddPoint(GraphPoint point, bool redraw)
        {
            points.Add(point);

            if(redraw)
                Redraw();
        }

        /// <summary>
        /// The AddPoints method will add an array of GraphPoint
        /// objects into the list to be drawn, then redrews the
        /// image to include the new points.
        /// </summary>
        /// <param name="point">
        /// The point parameter provides the array of GraphPoint
        /// objects to be added to the list.
        /// </param>
        public void AddPoints(GraphPoint[] point)
        {
            points.AddRange(point);
            Redraw();
        }
    }

}
```

11.3.5.3 Appointment Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

/// <summary>
/// RuleCompilerTestbench Namespace
/// Contains the classes and functionality to support the testing of the
/// RuleCompiler library and its various functionality.
/// </summary>
namespace RuleCompilerTestbench
{

    /// <summary>
    /// The Appointment class implements the IPeriodElement interface
    /// defined in the RuleCompiler library in order that it can act
    /// as an element of data that is capable of being tested against
```

```
    /// a set of rules.
    /// </summary>
    public class Appointment : RuleSupport.IPeriodElement
    {

        /// <summary>
        /// The start time of the appointment, stored as a DateTime.
        /// </summary>
        private DateTime start;

        /// <summary>
        /// The finish time of the appointment, stored as a DateTime.
        /// </summary>
        private DateTime finish;

        /// <summary>
        /// The Appointment's constructor takes a start and finish
        /// time for the appointment which can be exposed through the
        /// implemented interface methods.
        /// </summary>
        /// <param name="theStart">
        /// The start time of the appointment.
        /// </param>
        /// <param name="theFinish">
        /// The finish time of the appointment.
        /// </param>
        public Appointment(DateTime theStart, DateTime theFinish)
        {
            start = theStart;
            finish = theFinish;
        }

        /// <summary>
        /// This is an implementation of the interfaces Start
        /// property which return the start time of the appointment.
        /// </summary>
        public DateTime Start { get { return start; } }

        /// <summary>
        /// This is an implementation of the interfaces Finish
        /// property which return the finish time of the appointment.
        /// </summary>
        public DateTime Finish { get { return finish; } }

        /// <summary>
        /// This is an implementation of the interfaces RestLength
        /// property which returns a new TimeSpan as these
        /// appointments contain no rest, only continuous work.
        /// </summary>
        public RuleSupport.TimeSpan RestLength { get { return new
RuleSupport.TimeSpan(); } }

        /// <summary>
        /// This is an implementation of the interfaces WorkLength
        /// property which returns a new TimeSpan describing the
        /// difference between the start and finish times.
        /// </summary>
        public RuleSupport.TimeSpan WorkLength { get { return new
RuleSupport.TimeSpan(finish.Subtract(start)); } }

    }

}
```

11.3.5.4 TestBench Class

```
// Use this library for the most basic functionality such as the use of
// strings.
using System;

// Use the following two libraries for providing graphical interface
// capabilities to the class, essential when a class inherits from a
```

```
// Form class.
using System.Drawing;
using System.Windows.Forms;

// Use the these libraries for providing dynamically sizable collections.
//using System.Collections;
using System.Collections.Generic;

// Use this library to provide a means of serialising the objects into
// XML for file storage and network transportation.
using System.Runtime.Serialization;

// Use this library to provide access to the Thread class.
using System.Threading;

// Use the facilities provided by the rule support namespace.
using RuleSupport;
using System.IO;

/// <summary>
/// RuleCompilerTestbench Namespace
/// Contains the classes and functionality to support the testing of the
/// RuleCompiler library and its various functionality.
/// </summary>
namespace RuleCompilerTestbench
{

    /// <summary>
    /// The TestBench form is designed to provide a graphical means
    /// of producing and running tests on the RuleCompiler.
    /// </summary>
    public partial class TestBench : Form
    {

        /// <summary>
        /// The periodTypes variable is used to stored the Types
        /// that are returned by the compiler ones the rules are
        /// successfully compiled. These types can then be used
        /// in conjunction with the Tester class to do the testing
        /// for us.
        /// </summary>
        Type[] periodTypes = null;

        /// <summary>
        /// The testThread variable is used to store a reference
        /// to the testing thread once it has been created.
        /// </summary>
        Thread testThread = null;

        /// <summary>
        /// The logFile variable store access to the log file, if
        /// it is open.
        /// </summary>
        StreamWriter logFile = null;
        private CheckBox UseGraphCheck;
        private CheckBox UseLogFileCheck;
        private CheckBox OutputPCInfoCheck;

        /// <summary>
        /// The patternFile variables stores access to the log
        /// file which holds the work patterns.
        /// </summary>
        StreamWriter patternFile = null;

        /// <summary>
        /// The DatasetSize property is used to return the size
        /// of the database for the given types of rules.
        /// </summary>
        private int DatasetSize
        {
            get
            {
                if (DrivingRadio.Checked)
                    return 10;
                else
                    return 14;
```

```
            }
        }

        /// <summary>
        /// The WeekIncrements property is used to return the
        /// number of weeks that are span through the data
        /// provided for the given types of rules.
        /// </summary>
        private int WeekIncrements
        {
            get
            {
                if (DrivingRadio.Checked)
                    return 2;
                else
                    return 1;
            }
        }

        /// <summary>
        /// The standard constructor just calls the normal
        /// InitializeComponent method. No other initialization is
        /// requirement at this point.
        /// </summary>
        public TestBench()
        {
            InitializeComponent();
        }

        #region Windows Form Designer generated code

        private System.Windows.Forms.CheckBox OptimisedCheck;
        private System.Windows.Forms.Panel panel1;
        private System.Windows.Forms.ProgressBar TestingProgress;
        private System.Windows.Forms.Button CompileButton;
        private System.Windows.Forms.Button RunTestButton;
        private System.Windows.Forms.TextBox ResultText;
        private System.Windows.Forms.PictureBox ResultGraph;
        private System.Windows.Forms.RadioButton TeachingRadio;
        private System.Windows.Forms.RadioButton DrivingRadio;


        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.OptimisedCheck = new System.Windows.Forms.CheckBox();
            this.panel1 = new System.Windows.Forms.Panel();
            this.TeachingRadio = new System.Windows.Forms.RadioButton();
            this.DrivingRadio = new System.Windows.Forms.RadioButton();
            this.UseGraphCheck = new System.Windows.Forms.CheckBox();
            this.UseLogFileCheck = new System.Windows.Forms.CheckBox();
            this.OutputPCInfoCheck = new System.Windows.Forms.CheckBox();
            this.TestingProgress = new System.Windows.Forms.ProgressBar();
            this.CompileButton = new System.Windows.Forms.Button();
            this.RunTestButton = new System.Windows.Forms.Button();
            this.ResultText = new System.Windows.Forms.TextBox();
            this.ResultGraph = new System.Windows.Forms.PictureBox();
            this.panel1.SuspendLayout();

((System.ComponentModel.ISupportInitialize)(this.ResultGraph)).BeginInit();
            this.SuspendLayout();
            //
            // OptimisedCheck
            //
            this.OptimisedCheck.AutoSize = true;
            this.OptimisedCheck.Checked = true;
            this.OptimisedCheck.CheckState =
System.Windows.Forms.CheckState.Checked;
            this.OptimisedCheck.Location = new System.Drawing.Point(143, 2);
            this.OptimisedCheck.Name = "OptimisedCheck";
            this.OptimisedCheck.Size = new System.Drawing.Size(110, 17);
            this.OptimisedCheck.TabIndex = 2;
            this.OptimisedCheck.Text = "Optimised Testing";
```

```
            this.OptimisedCheck.CheckedChanged += new
System.EventHandler(this.OptimisedCheck CheckedChanged);
            //
            // panel1
            //
            this.panel1.Controls.Add(this.TeachingRadio);
            this.panel1.Controls.Add(this.DrivingRadio);
            this.panel1.Controls.Add(this.UseGraphCheck);
            this.panel1.Controls.Add(this.UseLogFileCheck);
            this.panel1.Controls.Add(this.OutputPCInfoCheck);
            this.panel1.Controls.Add(this.OptimisedCheck);
            this.panel1.Controls.Add(this.TestingProgress);
            this.panel1.Controls.Add(this.CompileButton);
            this.panel1.Controls.Add(this.RunTestButton);
            this.panel1.Dock = System.Windows.Forms.DockStyle.Top;
            this.panel1.Location = new System.Drawing.Point(0, 0);
            this.panel1.Name = "panel1";
            this.panel1.Size = new System.Drawing.Size(629, 46);
            this.panel1.TabIndex = 6;
            //
            // TeachingRadio
            //
            this.TeachingRadio.AutoSize = true;
            this.TeachingRadio.Location = new System.Drawing.Point(67, 2);
            this.TeachingRadio.Name = "TeachingRadio";
            this.TeachingRadio.Size = new System.Drawing.Size(70, 17);
            this.TeachingRadio.TabIndex = 3;
            this.TeachingRadio.Text = "Teaching";
            this.TeachingRadio.UseVisualStyleBackColor = true;
            this.TeachingRadio.CheckedChanged += new
System.EventHandler(this.Radio_CheckedChanged);
            //
            // DrivingRadio
            //
            this.DrivingRadio.AutoSize = true;
            this.DrivingRadio.Checked = true;
            this.DrivingRadio.Location = new System.Drawing.Point(3, 2);
            this.DrivingRadio.Name = "DrivingRadio";
            this.DrivingRadio.Size = new System.Drawing.Size(58, 17);
            this.DrivingRadio.TabIndex = 3;
            this.DrivingRadio.TabStop = true;
            this.DrivingRadio.Text = "Driving";
            this.DrivingRadio.UseVisualStyleBackColor = true;
            this.DrivingRadio.CheckedChanged += new
System.EventHandler(this.Radio CheckedChanged);
            //
            // UseGraphCheck
            //
            this.UseGraphCheck.AutoSize = true;
            this.UseGraphCheck.Checked = true;
            this.UseGraphCheck.CheckState =
System.Windows.Forms.CheckState.Checked;
            this.UseGraphCheck.Location = new System.Drawing.Point(375, 3);
            this.UseGraphCheck.Name = "UseGraphCheck";
            this.UseGraphCheck.Size = new System.Drawing.Size(77, 17);
            this.UseGraphCheck.TabIndex = 2;
            this.UseGraphCheck.Text = "Use Graph";
            this.UseGraphCheck.Visible = false;
            this.UseGraphCheck.CheckedChanged += new
System.EventHandler(this.OptimisedCheck_CheckedChanged);
            //
            // UseLogFileCheck
            //
            this.UseLogFileCheck.AutoSize = true;
            this.UseLogFileCheck.Checked = true;
            this.UseLogFileCheck.CheckState =
System.Windows.Forms.CheckState.Checked;
            this.UseLogFileCheck.Location = new System.Drawing.Point(259, 2);
            this.UseLogFileCheck.Name = "UseLogFileCheck";
            this.UseLogFileCheck.Size = new System.Drawing.Size(85, 17);
            this.UseLogFileCheck.TabIndex = 2;
            this.UseLogFileCheck.Text = "Use Log File";
            this.UseLogFileCheck.Visible = false;
            this.UseLogFileCheck.CheckedChanged += new
System.EventHandler(this.OptimisedCheck_CheckedChanged);
            //
```

```
            // OutputPCInfoCheck
            //
            this.OutputPCInfoCheck.AutoSize = true;
            this.OutputPCInfoCheck.Checked = true;
            this.OutputPCInfoCheck.CheckState =
System.Windows.Forms.CheckState.Checked;
            this.OutputPCInfoCheck.Location = new System.Drawing.Point(494, 3);
            this.OutputPCInfoCheck.Name = "OutputPCInfoCheck";
            this.OutputPCInfoCheck.Size = new System.Drawing.Size(130, 17);
            this.OutputPCInfoCheck.TabIndex = 2;
            this.OutputPCInfoCheck.Text = "Output PC Information";
            this.OutputPCInfoCheck.Visible = false;
            this.OutputPCInfoCheck.CheckedChanged += new
System.EventHandler(this.OptimisedCheck CheckedChanged);
            //
            // TestingProgress
            //
            this.TestingProgress.Anchor =
((System.Windows.Forms.AnchorStyles)(((System.Windows.Forms.AnchorStyles.Top |
System.Windows.Forms.AnchorStyles.Left)
                        | System.Windows.Forms.AnchorStyles.Right)));
            this.TestingProgress.Location = new System.Drawing.Point(268, 23);
            this.TestingProgress.Name = "TestingProgress";
            this.TestingProgress.Size = new System.Drawing.Size(349, 22);
            this.TestingProgress.TabIndex = 1;
            //
            // CompileButton
            //
            this.CompileButton.Location = new System.Drawing.Point(0, 24);
            this.CompileButton.Name = "CompileButton";
            this.CompileButton.Size = new System.Drawing.Size(120, 22);
            this.CompileButton.TabIndex = 0;
            this.CompileButton.Text = "Compile Rules";
            this.CompileButton.Click += new
System.EventHandler(this.CompileButton Click);
            //
            // RunTestButton
            //
            this.RunTestButton.Enabled = false;
            this.RunTestButton.Location = new System.Drawing.Point(126, 24);
            this.RunTestButton.Name = "RunTestButton";
            this.RunTestButton.Size = new System.Drawing.Size(120, 22);
            this.RunTestButton.TabIndex = 0;
            this.RunTestButton.Text = "Run Test";
            this.RunTestButton.Click += new
System.EventHandler(this.RunTestButton Click);
            //
            // ResultText
            //
            this.ResultText.AcceptsReturn = true;
            this.ResultText.Dock = System.Windows.Forms.DockStyle.Fill;
            this.ResultText.Location = new System.Drawing.Point(0, 46);
            this.ResultText.Multiline = true;
            this.ResultText.Name = "ResultText";
            this.ResultText.ReadOnly = true;
            this.ResultText.ScrollBars = System.Windows.Forms.ScrollBars.Both;
            this.ResultText.Size = new System.Drawing.Size(629, 242);
            this.ResultText.TabIndex = 5;
            //
            // ResultGraph
            //
            this.ResultGraph.Dock = System.Windows.Forms.DockStyle.Bottom;
            this.ResultGraph.Location = new System.Drawing.Point(0, 288);
            this.ResultGraph.Name = "ResultGraph";
            this.ResultGraph.Size = new System.Drawing.Size(629, 215);
            this.ResultGraph.TabIndex = 8;
            this.ResultGraph.TabStop = false;
            //
            // TestBench
            //
            this.ClientSize = new System.Drawing.Size(629, 503);
            this.Controls.Add(this.ResultText);
            this.Controls.Add(this.panel1);
            this.Controls.Add(this.ResultGraph);
            this.Name = "TestBench";
            this.Text = "Rule Compiler Test Bench";
```

```
            this.panel1.ResumeLayout(false);
            this.panel1.PerformLayout();

((System.ComponentModel.ISupportInitialize)(this.ResultGraph)).EndInit();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        /// <summary>
        /// The GenerateDrivingDataset method will generate a dataset
        /// containing 10 appointments. These appointments are
        /// set apart in such a way as to be ground truth data
        /// described in more details in the thesis.
        /// </summary>
        /// <param name="IncrementWeeks">
        /// The IncrementWeeks parameter provides a means of
        /// shifting the start and end times of the appointments
        /// by a given number of weeks so that the result can
        /// be used with a previously generated collection.
        /// </param>
        /// <returns>
        /// The return value is an array of IPeriodElement
        /// objects, for which the Appointment class implements
        /// and that can be used with the rule testing engine.
        /// </returns>
        private RuleSupport.IPeriodElement[] GenerateDrivingDataset(int
IncrementWeeks)
        {
            // Create the 10 appointments
            Appointment app1 = new Appointment((new DateTime(2007, 01, 01, 09,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 01, 14, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app2 = new Appointment((new DateTime(2007, 01, 01, 15,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 01, 16, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app3 = new Appointment((new DateTime(2007, 01, 02, 09,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 02, 14, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app4 = new Appointment((new DateTime(2007, 01, 02, 15,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 02, 16, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app5 = new Appointment((new DateTime(2007, 01, 03, 09,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 03, 14, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app6 = new Appointment((new DateTime(2007, 01, 03, 15,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 03, 16, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app7 = new Appointment((new DateTime(2007, 01, 05, 09,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 05, 14, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app8 = new Appointment((new DateTime(2007, 01, 05, 15,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 05, 16, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app9 = new Appointment((new DateTime(2007, 01, 06, 09,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 06, 14, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app10 = new Appointment((new DateTime(2007, 01, 06, 15,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 06, 16, 00,
00)).AddDays(7 * IncrementWeeks));
            return new Appointment[] { app1, app2, app3, app4, app5, app6,
app7, app8, app9, app10 };
        }

        /// <summary>
        /// The GenerateTeachingDataset method will generate a dataset
        /// containing 14 appointments. These appointments are
        /// set apart in such a way as to be ground truth data
        /// described in more details in the thesis.
        /// </summary>
        /// <param name="IncrementWeeks">
        /// The IncrementWeeks parameter provides a means of
        /// shifting the start and end times of the appointments
        /// by a given number of weeks so that the result can
```

```
        /// be used with a previously generated collection.
        /// </param>
        /// <returns>
        /// The return value is an array of IPeriodElement
        /// objects, for which the Appointment class implements
        /// and that can be used with the rule testing engine.
        /// </returns>
        private RuleSupport.IPeriodElement[] GenerateTeachingDataset(int
IncrementWeeks)
        {
            // Create the 14 appointments
            Appointment app1 = new Appointment((new DateTime(2007, 01, 01, 09,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 01, 11, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app2 = new Appointment((new DateTime(2007, 01, 01, 11,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 01, 13, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app3 = new Appointment((new DateTime(2007, 01, 01, 14,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 01, 16, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app4 = new Appointment((new DateTime(2007, 01, 01, 16,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 01, 18, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app5 = new Appointment((new DateTime(2007, 01, 02, 09,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 02, 11, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app6 = new Appointment((new DateTime(2007, 01, 02, 11,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 02, 13, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app7 = new Appointment((new DateTime(2007, 01, 03, 14,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 03, 16, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app8 = new Appointment((new DateTime(2007, 01, 03, 16,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 03, 18, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app9 = new Appointment((new DateTime(2007, 01, 03, 19,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 03, 21, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app10 = new Appointment((new DateTime(2007, 01, 03, 21,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 03, 23, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app11 = new Appointment((new DateTime(2007, 01, 04, 14,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 04, 16, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app12 = new Appointment((new DateTime(2007, 01, 04, 16,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 04, 18, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app13 = new Appointment((new DateTime(2007, 01, 05, 09,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 05, 11, 00,
00)).AddDays(7 * IncrementWeeks));
            Appointment app14 = new Appointment((new DateTime(2007, 01, 05, 11,
00, 00)).AddDays(7 * IncrementWeeks), (new DateTime(2007, 01, 05, 13, 00,
00)).AddDays(7 * IncrementWeeks));
            return new Appointment[] { app1, app2, app3, app4, app5, app6,
app7, app8, app9, app10, app11, app12, app13, app14 };
        }

        /// <summary>
        /// The GenerateDataset method will generate a dataset
        /// depending on which of the rule types has been
        /// selected.
        /// </summary>
        /// <param name="IncrementWeeks">
        /// The IncrementWeeks parameter provides a means of
        /// shifting the start and end times of the appointments
        /// by a given number of weeks so that the result can
        /// be used with a previously generated collection.
        /// </param>
        /// <returns>
        /// The return value is an array of IPeriodElement
        /// objects, for which the Appointment class implements
        /// and that can be used with the rule testing engine.
        /// </returns>
        private RuleSupport.IPeriodElement[] GenerateDataset(int
IncrementWeeks)
        {
```

```
            if (DrivingRadio.Checked)
                return GenerateDrivingDataset(IncrementWeeks);
            else
                return GenerateTeachingDataset(IncrementWeeks);
        }

        /// <summary>
        /// The DumpIPeriodElementArray method is used to
        /// output the contents of an IPeriodElement array to
        /// the debug output window to aid with debugging.
        /// </summary>
        /// <param name="array">
        /// The array parameter describes the IPeriodElements
        /// to output to the debug window.
        /// </param>
        public void DumpIPeriodElementArray(RuleSupport.IPeriodElement[] array)
        {
            WriteLine(string.Format("{0}:\t\t{1}\t\t{2}", "Index", "Start
Date/Time", "End Date/Time"));

            for (int i = 0; i < array.Length; i++)
                WriteLine(string.Format("{0}:\t\t{1}\t\t{2}", i.ToString(),
array[i].Start.ToString(), array[i].Finish.ToString()));

            WriteLine("");
        }

        /// <summary>
        /// The WriteXMLLine method is used to write out
        /// a line of XML to the currently opened XML
        /// file.
        /// </summary>
        /// <param name="line">
        /// The line of XML to be written to the XML file.
        /// </param>
        public void WriteXMLLine(string line)
        {
            Console.WriteLine(line);

            UpdateGUI(ResultText, "Text", GetGUI(ResultText, "Text").ToString()
+ "\r\n" + line);
            UpdateGUI(ResultText, "SelectionStart", GetGUI(ResultText,
"Text").ToString().Length);
            MethodInvoke(ResultText, "ScrollToCaret", null);

            if (patternFile != null)
                patternFile.WriteLine(line);

#if DEBUG
            System.Diagnostics.Debug.WriteLine(line);
#endif
        }

        /// <summary>
        /// The WriteLine method is used to write out
        /// a line of log info to the currently opened log
        /// file.
        /// </summary>
        /// <param name="line">
        /// The line of log info to be written to the log file.
        /// </param>
        public void WriteLine(string line)
        {
            Console.WriteLine(line);

            MethodInvoke(ResultText, "SuspendLayout", null);
            UpdateGUI(ResultText, "Text", GetGUI(ResultText, "Text").ToString()
+ "\r\n" + line);
            UpdateGUI(ResultText, "SelectionStart", GetGUI(ResultText,
"Text").ToString().Length);
            MethodInvoke(ResultText, "ScrollToCaret", null);
            MethodInvoke(ResultText, "ResumeLayout", null);

            if (logFile != null)
                logFile.WriteLine(line);
```

```
#if DEBUG
            System.Diagnostics.Debug.WriteLine(line);
#endif
        }

        /// <summary>
        /// The DumpUnknown method is a helper method used
        /// to determine the type of object passed in and
        /// then calls the most appropriate method for
        /// outputing that type of object.
        /// </summary>
        /// <param name="o">
        /// The o parameter is the object to put output
        /// into XML.
        /// </param>
        /// <param name="depth">
        /// The depth parameter describes the number of
        /// tabs to be preceed the appointment XML details.
        /// </param>
        public void DumpUnknown(object o, int depth)
        {
            if (o is Appointment)
                DumpAppointment((Appointment)o, depth);
            else if (o is RuleSupport.Period)
                DumpPeriod((Period)o, depth);
            else if (o is RuleSupport.WorkPattern)
                DumpWorkPattern((WorkPattern)o, depth);
        }

        /// <summary>
        /// The DumpAppointment method is a helper method
        /// used to display appointment information to the
        /// output log in an XML style format.
        /// </summary>
        /// <param name="appointment">
        /// The appointment parameter describes the
        /// appointment to be output to the XML file.
        /// </param>
        /// <param name="depth">
        /// The depth parameter describes the number of
        /// tabs to be preceed the appointment XML details.
        /// </param>
        private void DumpAppointment(Appointment appointment, int depth)
        {
            WriteXMLLine((new string('\t', depth)) + "<Appointment>");

            WriteXMLLine((new string('\t', depth + 1)) + "<Start>" +
appointment.Start.ToString() + "</Start>");
            WriteXMLLine((new string('\t', depth + 1)) + "<Finish>" +
appointment.Finish.ToString() + "</Finish>");
            WriteXMLLine((new string('\t', depth + 1)) + "<WorkLength>" +
appointment.WorkLength.ToString() + "</WorkLength>");
            WriteXMLLine((new string('\t', depth + 1)) + "<RestLength>" +
appointment.RestLength.ToString() + "</RestLength>");

            WriteXMLLine((new string('\t', depth)) + "</Appointment>");
        }

        /// <summary>
        /// The DumpPeriod method is a helper method
        /// used to display period information to the
        /// output log in an XML style format.
        /// </summary>
        /// <param name="period">
        /// The period parameter describes the
        /// period to be output to the XML file.
        /// </param>
        /// <param name="depth">
        /// The depth parameter describes the number of
        /// tabs to be preceed the appointment XML details.
        /// </param>
        public void DumpPeriod(RuleSupport.Period period, int depth)
        {
            WriteXMLLine((new string('\t', depth)) + "<" +
period.GetType().Name + ">");
            WriteXMLLine((new string('\t', depth + 1)) + "<Start>" +
```

```
period.Start.ToString() + "</Start>");
            WriteXMLLine((new string('\t', depth + 1)) + "<Finish>" +
period.Finish.ToString() + "</Finish>");
            WriteXMLLine((new string('\t', depth + 1)) + "<Length>" +
period.Length.ToString() + "</Length>");
            WriteXMLLine((new string('\t', depth + 1)) + "<WorkLength>" +
period.WorkLength.ToString() + "</WorkLength>");
            WriteXMLLine((new string('\t', depth + 1)) + "<RestLength>" +
period.RestLength.ToString() + "</RestLength>");

            List<RuleSupport.IPeriodElement> list = new
List<RuleSupport.IPeriodElement>();

            if (period.Count > 0)
                WriteXMLLine((new string('\t', depth + 1)) + "<Contains>");

            for (int i = 0; i < period.Count; i++)
            {
                if (!list.Contains(period[i]))
                    DumpUnknown(period[i], depth + 2);
            }

            if (period.Count > 0)
                WriteXMLLine((new string('\t', depth + 1)) + "</Contains>");

            WriteXMLLine((new string('\t', depth)) + "</" +
period.GetType().Name + ">");
        }

        /// <summary>
        /// The DumpWorkPattern method is a helper method
        /// used to display work pattern information to the
        /// output log in an XML style format.
        /// </summary>
        /// <param name="workPattern">
        /// The workPattern parameter describes the
        /// work pattern to be output to the XML file.
        /// </param>
        /// <param name="depth">
        /// The depth parameter describes the number of
        /// tabs to be preceed the appointment XML details.
        /// </param>
        public void DumpWorkPattern(RuleSupport.WorkPattern workPattern, int
depth)
        {
            WriteXMLLine((new string('\t', depth)) + "<Work Pattern>");

            for (int i = 0; i < workPattern.Count; i++)
                DumpUnknown(workPattern[i], depth + 3);

            WriteXMLLine((new string('\t', depth)) + "</Work Pattern>");
        }

        /// <summary>
        /// The GetDrivingRules method is used to return the rules
        /// relating to the transport scheduling domain. It is
        /// possible to read the rules from an XML file or, more
        /// simply, construct them on the fly.
        /// </summary>
        /// <returns>
        /// The return value of this method is a Periods collection
        /// describing the rules.
        /// </returns>
        private RuleDefinitionLanguage.Periods GetDrivingRules()
        {
            // It is possible to just load the rules from an XML file.
            //RuleDefinitionLanguage.Periods theRules = new
RuleDefinitionLanguage.Periods(System.IO.Path.Combine(Environment.CurrentDirect
ory, @"Rules\DrivingRules.xml"));

            // Create the test rules.
            RuleDefinitionLanguage.Periods theRules = new
RuleDefinitionLanguage.Periods();

            // A daily work period can be up to 8 hours long.
            RuleDefinitionLanguage.WP myDWP = new RuleDefinitionLanguage.WP();
```

```
            myDWP.Description.Name = "DWP";
            myDWP.Description.Description = "Daily Work Period";
            myDWP.Description.PrePosts = new RuleDefinitionLanguage.PrePost[] {
new RuleDefinitionLanguage.PrePost("", "DRP"), new
RuleDefinitionLanguage.PrePost("DRP", "DRP"), new
RuleDefinitionLanguage.PrePost("DRP", ""), new
RuleDefinitionLanguage.PrePost("", "") };
            myDWP.Description.Contains = new RuleDefinitionLanguage.Contain[] {
new RuleDefinitionLanguage.Contain("Appointment") };
            myDWP.Rules = /*new RuleDefinitionLanguage.AndCondition(
                            new RuleDefinitionLanguage.Comparitor[]
                            {*/
                                new RuleDefinitionLanguage.Comparison(

RuleDefinitionLanguage.ComparisonOperation.LessThanOrEqualTo,
                                    new RuleDefinitionLanguage.Value[]
                                    {
                                        new
RuleDefinitionLanguage.Value("WorkLength",
RuleDefinitionLanguage.ValueType.Parameter),
                                        new RuleDefinitionLanguage.Value("8",
RuleDefinitionLanguage.ValueType.Hours)
                                    },
                                    null
                                )/*,
                                new RuleDefinitionLanguage.Comparison(

RuleDefinitionLanguage.ComparisonOperation.GreaterThan,
                                    new RuleDefinitionLanguage.Value[]
                                    {
                                        new
RuleDefinitionLanguage.Value("Appointment.Count",
RuleDefinitionLanguage.ValueType.Parameter),
                                        new RuleDefinitionLanguage.Value("0",
RuleDefinitionLanguage.ValueType.Number)
                                    },
                                    null
                                )
                            }
                        )*/;
            theRules.Add(myDWP);

            // A weekly work period can be up to 6 daily work periods long.
            RuleDefinitionLanguage.WP myWWP = new RuleDefinitionLanguage.WP();
            myWWP.Description.Name = "WWP";
            myWWP.Description.Description = "Weekly Work Period";
            myWWP.Description.PrePosts = new RuleDefinitionLanguage.PrePost[] {
new RuleDefinitionLanguage.PrePost("WRP", "WRP"), new
RuleDefinitionLanguage.PrePost("", "WRP"), new
RuleDefinitionLanguage.PrePost("WRP", ""), new
RuleDefinitionLanguage.PrePost("", "") };
            myWWP.Description.Contains = new RuleDefinitionLanguage.Contain[] {
new RuleDefinitionLanguage.Contain("DWP"), new
RuleDefinitionLanguage.Contain("DRP") };
            myWWP.Rules = new RuleDefinitionLanguage.Comparison(

RuleDefinitionLanguage.ComparisonOperation.LessThanOrEqualTo,
                                    new RuleDefinitionLanguage.Value[]
                                    {
                                        new
RuleDefinitionLanguage.Value("DWP.Count",
RuleDefinitionLanguage.ValueType.Parameter),
                                        new RuleDefinitionLanguage.Value("6",
RuleDefinitionLanguage.ValueType.Number)
                                    },
                                    null
                                );
            theRules.Add(myWWP);

            // A weekly rest period must be at least 30 hours long.
            RuleDefinitionLanguage.RP myWRP = new RuleDefinitionLanguage.RP();
            myWRP.Description.Name = "WRP";
            myWRP.Description.Description = "Weekly Rest Period";
            myWRP.Description.PrePosts = new RuleDefinitionLanguage.PrePost[] {
new RuleDefinitionLanguage.PrePost("WWP", "WWP") };
            myWRP.Description.Contains = new RuleDefinitionLanguage.Contain[] {
```

```
};
            myWRP.Rules = new RuleDefinitionLanguage.Comparison(

RuleDefinitionLanguage.ComparisonOperation.GreaterThanOrEqualTo,
                                new RuleDefinitionLanguage.Value[]
                                {
                                        new
RuleDefinitionLanguage.Value("Length",
RuleDefinitionLanguage.ValueType.Parameter),
                                        /*new
RuleDefinitionLanguage.Value("RestLength",
RuleDefinitionLanguage.ValueType.Parameter),*/
                                    new RuleDefinitionLanguage.Value("30",
RuleDefinitionLanguage.ValueType.Hours)
                                },
                                null
                             );
            theRules.Add(myWRP);

            // A daily rest period must be at least 8 hours long.
            // A daily rest period must be less than 30 hours long.
            RuleDefinitionLanguage.RP myDRP = new RuleDefinitionLanguage.RP();
            myDRP.Description.Name = "DRP";
            myDRP.Description.Description = "Daily Rest Period";
            myDRP.Description.PrePosts = new RuleDefinitionLanguage.PrePost[] {
new RuleDefinitionLanguage.PrePost("DWP", "DWP") };
            myDRP.Description.Contains = new RuleDefinitionLanguage.Contain[] {
};
            myDRP.Rules = new RuleDefinitionLanguage.AndCondition(
                            new RuleDefinitionLanguage.Comparitor[]
                            {
                                new RuleDefinitionLanguage.Comparison(

RuleDefinitionLanguage.ComparisonOperation.GreaterThanOrEqualTo,
                                    new RuleDefinitionLanguage.Value[]
                                    {
                                        new
RuleDefinitionLanguage.Value("Length",
RuleDefinitionLanguage.ValueType.Parameter),
                                        /*new
RuleDefinitionLanguage.Value("RestLength",
RuleDefinitionLanguage.ValueType.Parameter),*/
                                        new RuleDefinitionLanguage.Value("8",
RuleDefinitionLanguage.ValueType.Hours)
                                    },
                                    null
                                ),
                                new RuleDefinitionLanguage.Comparison(

RuleDefinitionLanguage.ComparisonOperation.LessThan,
                                    new RuleDefinitionLanguage.Value[]
                                    {
                                        new
RuleDefinitionLanguage.Value("Length",
RuleDefinitionLanguage.ValueType.Parameter),
                                        /*new
RuleDefinitionLanguage.Value("RestLength",
RuleDefinitionLanguage.ValueType.Parameter),*/
                                        new RuleDefinitionLanguage.Value("30",
RuleDefinitionLanguage.ValueType.Hours)
                                    },
                                    null
                                )
                            }
                        );
            theRules.Add(myDRP);

            return theRules;
        }

        /// <summary>
        /// The GetTeachingRules method is used to return the rules
        /// relating to the classroom scheduling domain. It is
        /// possible to read the rules from an XML file or, more
        /// simply, construct them on the fly.
        /// </summary>
```

```
        /// <returns>
        /// The return value of this method is a Periods collection
        /// describing the rules.
        /// </returns>
        private RuleDefinitionLanguage.Periods GetTeachingRules()
        {
            // It is possible to just load the rules from an XML file.
            //RuleDefinitionLanguage.Periods theRules = new
RuleDefinitionLanguage.Periods(System.IO.Path.Combine(Environment.CurrentDirect
ory, @"Rules\TeachingRules.xml"));

            // Create the test rules.
            RuleDefinitionLanguage.Periods theRules = new
RuleDefinitionLanguage.Periods();

            // Shift
            // Cannot be longer than 4 hours
            // Must start after 9am
            // Must finish before 9pm
            RuleDefinitionLanguage.WP myShift = new
RuleDefinitionLanguage.WP();
            myShift.Description.Name = "S";
            myShift.Description.Description = "A shift of work";
            myShift.Description.PrePosts = new RuleDefinitionLanguage.PrePost[]
{ new RuleDefinitionLanguage.PrePost("", ""), new
RuleDefinitionLanguage.PrePost("", "SB"), new
RuleDefinitionLanguage.PrePost("SB", "SB"), new
RuleDefinitionLanguage.PrePost("SB", "") };
            myShift.Description.Contains = new RuleDefinitionLanguage.Contain[]
{ new RuleDefinitionLanguage.Contain("Appointment") };
            myShift.Rules = new RuleDefinitionLanguage.AndCondition(
                                new RuleDefinitionLanguage.Comparitor[] {
                                    new RuleDefinitionLanguage.Comparison(

RuleDefinitionLanguage.ComparisonOperation.LessThanOrEqualTo,
                                        new RuleDefinitionLanguage.Value[] {
new RuleDefinitionLanguage.Value("Length",
RuleDefinitionLanguage.ValueType.Parameter), new
RuleDefinitionLanguage.Value("4", RuleDefinitionLanguage.ValueType.Hours) },
                                        null
                                    ),
                                    new RuleDefinitionLanguage.Comparison(

RuleDefinitionLanguage.ComparisonOperation.GreaterThanOrEqualTo,
                                        new RuleDefinitionLanguage.Value[] {
new RuleDefinitionLanguage.Value("Start.Hour",
RuleDefinitionLanguage.ValueType.Parameter), new
RuleDefinitionLanguage.Value("9", RuleDefinitionLanguage.ValueType.Number) },
                                        null
                                    ),
                                    new RuleDefinitionLanguage.Comparison(

RuleDefinitionLanguage.ComparisonOperation.LessThanOrEqualTo,
                                        new RuleDefinitionLanguage.Value[] {
new RuleDefinitionLanguage.Value("Finish.Hour",
RuleDefinitionLanguage.ValueType.Parameter), new
RuleDefinitionLanguage.Value("23", RuleDefinitionLanguage.ValueType.Number) },
                                        null
                                    )
                                }
                            );
            theRules.Add(myShift);

            // Shift Break
            // Must be at least 1 hour long
            RuleDefinitionLanguage.RP myShiftBreak = new
RuleDefinitionLanguage.RP();
            myShiftBreak.Description.Name = "SB";
            myShiftBreak.Description.Description = "A break between work
shifts";
            myShiftBreak.Description.PrePosts = new
RuleDefinitionLanguage.PrePost[] { new RuleDefinitionLanguage.PrePost("S", "S")
};
            myShiftBreak.Description.Contains = new
RuleDefinitionLanguage.Contain[] { };
            myShiftBreak.Rules = new RuleDefinitionLanguage.Comparison(
```

```
RuleDefinitionLanguage.ComparisonOperation.GreaterThanOrEqualTo,
                                new RuleDefinitionLanguage.Value[] { new
RuleDefinitionLanguage.Value("RestLength",
RuleDefinitionLanguage.ValueType.Parameter), new
RuleDefinitionLanguage.Value("1", RuleDefinitionLanguage.ValueType.Hours) },
                                null
                            );
            theRules.Add(myShiftBreak);

            // Daily Work Period
            // Cannot be longer than 9 hours
            // cannot be more than 2 shifts long
            RuleDefinitionLanguage.WP myDailyWorkPeriod = new
RuleDefinitionLanguage.WP();
            myDailyWorkPeriod.Description.Name = "DWP";
            myDailyWorkPeriod.Description.Description = "A daily work period";
            myDailyWorkPeriod.Description.PrePosts = new
RuleDefinitionLanguage.PrePost[] { new RuleDefinitionLanguage.PrePost("", ""),
new RuleDefinitionLanguage.PrePost("", "DRP"), new
RuleDefinitionLanguage.PrePost("DRP", "DRP"), new
RuleDefinitionLanguage.PrePost("DRP", "") };
            myDailyWorkPeriod.Description.Contains = new
RuleDefinitionLanguage.Contain[] { new RuleDefinitionLanguage.Contain("S"), new
RuleDefinitionLanguage.Contain("SB") };
            myDailyWorkPeriod.Rules = new RuleDefinitionLanguage.AndCondition(
                                new RuleDefinitionLanguage.Comparitor[] {
                                    new RuleDefinitionLanguage.Comparison(

RuleDefinitionLanguage.ComparisonOperation.LessThanOrEqualTo,
                                        new RuleDefinitionLanguage.Value[] {
new RuleDefinitionLanguage.Value("Length",
RuleDefinitionLanguage.ValueType.Parameter), new
RuleDefinitionLanguage.Value("9", RuleDefinitionLanguage.ValueType.Hours) },
                                        null
                                    ),
                                    new RuleDefinitionLanguage.Comparison(

RuleDefinitionLanguage.ComparisonOperation.LessThanOrEqualTo,
                                        new RuleDefinitionLanguage.Value[] {
new RuleDefinitionLanguage.Value("S.Count",
RuleDefinitionLanguage.ValueType.Parameter), new
RuleDefinitionLanguage.Value("3", RuleDefinitionLanguage.ValueType.Number) },
                                        null
                                    )
                                }
                            );
            theRules.Add(myDailyWorkPeriod);

            // Daily Rest Period
            // Must be at least 20 hours long
            RuleDefinitionLanguage.RP myDailyRestPeriod = new
RuleDefinitionLanguage.RP();
            myDailyRestPeriod.Description.Name = "DRP";
            myDailyRestPeriod.Description.Description = "A daily rest period";
            myDailyRestPeriod.Description.PrePosts = new
RuleDefinitionLanguage.PrePost[] { new RuleDefinitionLanguage.PrePost("DWP",
"DWP") };
            myDailyRestPeriod.Description.Contains = new
RuleDefinitionLanguage.Contain[] { };
            myDailyRestPeriod.Rules = new RuleDefinitionLanguage.Comparison(

RuleDefinitionLanguage.ComparisonOperation.GreaterThanOrEqualTo,
                                new RuleDefinitionLanguage.Value[] { new
RuleDefinitionLanguage.Value("Length",
RuleDefinitionLanguage.ValueType.Parameter), new
RuleDefinitionLanguage.Value("15", RuleDefinitionLanguage.ValueType.Hours) },
                                null
                            );
            theRules.Add(myDailyRestPeriod);

            // Weekly Work Period
            // Must be less than 5 DWPs
            RuleDefinitionLanguage.WP myWeeklyWorkPeriod = new
RuleDefinitionLanguage.WP();
            myWeeklyWorkPeriod.Description.Name = "WWP";
```

```
            myWeeklyWorkPeriod.Description.Description = "A weekly work
period";
            myWeeklyWorkPeriod.Description.PrePosts = new
RuleDefinitionLanguage.PrePost[] { new RuleDefinitionLanguage.PrePost("", ""),
new RuleDefinitionLanguage.PrePost("", "WRP"), new
RuleDefinitionLanguage.PrePost("WRP", ""), new
RuleDefinitionLanguage.PrePost("WRP", "WRP") };
            myWeeklyWorkPeriod.Description.Contains = new
RuleDefinitionLanguage.Contain[] { new RuleDefinitionLanguage.Contain("DWP"),
new RuleDefinitionLanguage.Contain("DRP") };
            myWeeklyWorkPeriod.Rules = new RuleDefinitionLanguage.Comparison(

RuleDefinitionLanguage.ComparisonOperation.LessThanOrEqualTo,
                                new RuleDefinitionLanguage.Value[] { new
RuleDefinitionLanguage.Value("DWP.Count",
RuleDefinitionLanguage.ValueType.Parameter), new
RuleDefinitionLanguage.Value("5", RuleDefinitionLanguage.ValueType.Number) },
                                null
                            );
            theRules.Add(myWeeklyWorkPeriod);

            // Weekly Rest Period
            // Must be at least 2 days long
            RuleDefinitionLanguage.RP myWeeklyRestPeriod = new
RuleDefinitionLanguage.RP();
            myWeeklyRestPeriod.Description.Name = "WRP";
            myWeeklyRestPeriod.Description.Description = "A weekly rest
period";
            myWeeklyRestPeriod.Description.PrePosts = new
RuleDefinitionLanguage.PrePost[] { new RuleDefinitionLanguage.PrePost("WWP",
"WWP") };
            myWeeklyRestPeriod.Description.Contains = new
RuleDefinitionLanguage.Contain[] { };
            myWeeklyRestPeriod.Rules = new RuleDefinitionLanguage.Comparison(

RuleDefinitionLanguage.ComparisonOperation.GreaterThanOrEqualTo,
                                new RuleDefinitionLanguage.Value[] { new
RuleDefinitionLanguage.Value("Length",
RuleDefinitionLanguage.ValueType.Parameter), new
RuleDefinitionLanguage.Value("2", RuleDefinitionLanguage.ValueType.Days) },
                                null
                            );
            theRules.Add(myWeeklyRestPeriod);

            return theRules;
        }

        /// <summary>
        /// The CompileButon_Click method is called when the user
        /// clicks on the compile button. This method will then
        /// construct a set of rules, in code. Rules can also come
        /// from a seperate XML file although this is adequest for
        /// the purposed of testing.
        /// </summary>
        private void CompileButton_Click(object sender, EventArgs e)
        {
            // If the log files are not open yet, make sure we
            // open them now.
            if (!LogsOpen && UseLogFileCheck.Checked)
                OpenLogs();

            // Define a variable to be used to store the rules once
            // loaded/created.
            RuleDefinitionLanguage.Periods theRules = null;

            // Determine if we are working with driving or teaching
            // rules.
            if (DrivingRadio.Checked)
                // Create the test rules.
                theRules = GetDrivingRules();
            else
                // Create the test rules.
                theRules = GetTeachingRules();

            // Serialise the rules into a string which gets them in
            // their XML representation that can be used to display
```

```
            // or transport.
            //ResultText.Text = theRules.ToString();
            WriteLine(theRules.ToString());
            WriteXMLLine(ResultText.Text);
            Application.DoEvents();

            // Now it is time to use the rule compiler to compile
            // the rules into Types that can be used to test against
            // a set of data.
            try
            {
                // Create a stopwatch that can be used to measure
                // how long the compilation process will take.
                StopWatch stopWatch = new StopWatch();

                // Start the stopwatch before compilation.
                stopWatch.Start();

                // Compile the rules used the Compiler classes
                // static Compile method. The result of the method
                // call is an array of Types that can be used for
                // rule testing.
                periodTypes = RuleCompiler.Compiler.Compile(theRules, new
Type[] { typeof(Appointment) });

                // Stop the stopwatch now compilation is finished.
                stopWatch.Stop();

                // Now we are finished, we can notify the user of
                // our success and compile time and enable the
                // Run Test button.
                WriteLine("The compile method returned: true, and took: " +
stopWatch.ToString());
                System.Windows.Forms.MessageBox.Show("The compile method
returned: true, and took: " + stopWatch.ToString());
                this.RunTestButton.Enabled = true;
                this.RunTestButton.Focus();
            }
            catch (RuleCompiler.RuleCompilerException ex)
            {
                // A problem occured at some point, more than
                // likely at the compile stage. A normal
                // application should catch the Exception and
                // display some information to the user.
                WriteLine("The compile method returned: false\n\nException
details:\n" + ex.ToString());
                System.Windows.Forms.MessageBox.Show("The compile method
returned: false\n\nException details:\n" + ex.ToString());
            }
        }

        // Because we are doing some cross-thread GUI updates,
        // we need to make these GUI changes on the correct
        // thread. To do this, we have got ourselves a delegate
        // and method that are used together to update a
        // controls property using loose binding. It is a bit
        // of a hack but it seems to work for this purpose.
        private delegate void UpdateGUIDelegate(object control, string
propertyName, object value);
        private void UpdateGUI(object control, string propertyName, object
value)
        {
            if (this.InvokeRequired)
                this.Invoke(new UpdateGUIDelegate(UpdateGUI), control,
propertyName, value);
            else
            {
                control.GetType().GetProperty(propertyName).SetValue(control,
value, null);
                Application.DoEvents();
            }
        }
        private delegate object GetGUIDelegate(object control, string
propertyName);
        private object GetGUI(object control, string propertyName)
        {
```

```
            if (this.InvokeRequired)
                return this.Invoke(new GetGUIDelegate(GetGUI), control,
propertyName);
            else
            {
                return
control.GetType().GetProperty(propertyName).GetValue(control, null);
            }
        }
        private delegate void MethodInvokeDelegate(object control, string
methodName, object[] args);
        private void MethodInvoke(object control, string methodName, object[]
args)
        {
            if (this.InvokeRequired)
                this.Invoke(new MethodInvokeDelegate(MethodInvoke), control,
methodName, args);
            else
            {
                List<Type> types = new List<Type>();
                if(args != null)
                    foreach (object o in args)
                        types.Add(o.GetType());
                control.GetType().GetMethod(methodName,
types.ToArray()).Invoke(control, args);
                Application.DoEvents();
            }
        }

        /// <summary>
        /// The TestMethod is used by the testing thread to
        /// carrying out the testing process. It is done in
        /// a seperate thread in order that the GUI and testing
        /// process can be treated independantly.
        /// </summary>
        private void TestMethod()
        {
            // If the log files are not open yet, make sure we
            // open them now.
            if (!LogsOpen && UseLogFileCheck.Checked)
                OpenLogs();

            // Firstly we'll set up some variables that are
            // use to define our testing parameters. Both must
            // be divisable by 10 and 14 as we generate either
            // 10 or 14 appointments for the various datasets
            // we are working with through the GenerateDataset
            // method.
            int Increments = 350;
            int MaxDatasetSize = Increments * 28;

            // Update our GUIs progress bar.
            UpdateGUI(TestingProgress, "Maximum", MaxDatasetSize);

            // We need to create ourselves a stopwatch so we
            // can take measurements of the overall testing
            // process. This stopwatch provides times for the
            // overall process, another stopwatch is used for
            // individual testing cycles.
            double maxTimeForGraph = 0.0f;
            StopWatch testWatch = new StopWatch();
            testWatch.Start();

            // Create a variable where we can stored the results
            // in a CSV style format that can be used by a
            // spreadsheet later.
            string ResultsString = "Ruleset Size, Dataset Size, Execution Time
(ms), Memory Increase (bytes)" + Environment.NewLine;
            WriteLine("Ruleset Size,Dataset Size,Execution Time (ms),Memory
Increase (bytes)");

            // Construct ourselves a new GraphImage object that
            // can be used for producing a graph. We also need
            // do display a graph now.
            GraphImage image = new GraphImage(ResultGraph.Size, new Size(1,
MaxDatasetSize), new Point(2, Increments / 2));
```

```
            image.AddPoint(new GraphImage.GraphPoint(new Point(0, 0), ""),
UseGraphCheck.Checked);
            if(UseGraphCheck.Checked)
                UpdateGUI(ResultGraph, "Image", image.Bitmap);

            // Try catch all of the processing. This is where
            // the heavy work is and if the user hits stop,
            // we'll be somewhere in here carrying out testing
            // work.
            try
            {

                // Keep a track as to which week we are working
                // on so that we can add additional weeks to the
                // list of data elements as we go through the
                // testing process, that well not clash with the
                // already generated weeks.
                int curWeek = 0;

                // Create a means of storing our work pattern
                // that is generated by the testing process.
                WorkPattern workPattern = null;

                // Create a new list of IPeriodElements, the data
                // that will be tested against the rules.
                List<IPeriodElement> theseDataElements = new
List<IPeriodElement>();

                // Now loop through to the max dataset size using
                // the given increments and undertake tests at
                // each increment.
                for (int ii = Increments; ii <= MaxDatasetSize; ii = ii +
Increments)
                {

                    // If we have chosen to carry out an
                    // optimised testing of the data, the dataset
                    // is now in the workpattern (we don't need to
                    // keep it). Otherwise, we leave the data in
                    // the array for the work pattern to be
                    // reconstructed and, in this case, we need
                    // to set our workpattern back to null.
                    if (OptimisedCheck.Checked)
                        theseDataElements.Clear();
                    else
                        workPattern = null;

                    // Now we need to generate our test data to be
                    // used during this iteration. We'll do this by
                    // looping the appropriate number of times.
                    for (int iii = 0; iii < (Increments / DatasetSize); iii++)
                    {
                        theseDataElements.AddRange(GenerateDataset(curWeek));
                        curWeek = curWeek + WeekIncrements;
                    }

                    // We now have the data and we have the rules,
                    // we can carry out the test using the Tester
                    // class.
                    RuleTesting.Tester Tester = new RuleTesting.Tester();

                    // We need to give ourselves a variable that
                    // will be used to store the return result as
                    // to the success of the rule testing process.
                    bool testSuccess;

                    // Collect garbage before we start so the
                    // garbage collector does not interfere so
                    // much with our times.
                    GC.Collect();

                    // Create ourselves a stopwatch for using during
                    // this iteration and start the timer.
                    StopWatch stopWatch = new StopWatch();
                    stopWatch.Start();
```

```
                    // Do the test, passing in the compiled rules
                    // (the periodTypes) the array of data and the
                    // workpattern.
                    testSuccess = Tester.Test(periodTypes,
theseDataElements.ToArray(), ref workPattern);

                    // The test is complete, stop the watch.
                    stopWatch.Stop();

                    // The result should be successfully as we've
                    // specified ground truth data. If it wasn't
                    // we need to dump our data so we can determine
                    // what went wrong and debug.
                    if (!testSuccess)
                    {
                        // Output some information about the current
                        // test including both the data and the
                        // currently generated work pattern.
                        DumpIPeriodElementArray(theseDataElements.ToArray());

                        WriteXMLLine("\n\n<!--\n\tTest executed: Dataset Size:
" + theseDataElements.Count.ToString() + ", Result: " + testSuccess.ToString()
+ "\n-->");
                        DumpWorkPattern(workPattern, 0);
                        WriteXMLLine("");
                    }


                    // Update our GUIs progress bar and display
                    // various updates to the user as to our status.
                    UpdateGUI(TestingProgress, "Value", (TestingProgress.Value
+ Increments));
                    string newStatus = periodTypes.Length.ToString() + "," +
theseDataElements.Count.ToString() + "," +
stopWatch.TotalTimeSpan.TotalSeconds.ToString() + "," +
stopWatch.TotalMemory.ToString();
                    ResultsString += newStatus + Environment.NewLine;
                    WriteLine(newStatus);
                    //UpdateGUI(ResultText, "Text", ResultsString);

                    // Update our graph with the new results and
                    // display the new graph image to the user.
                    if (stopWatch.TotalTimeSpan.TotalSeconds > maxTimeForGraph)
                        maxTimeForGraph = stopWatch.TotalTimeSpan.TotalSeconds;
                    if (UseGraphCheck.Checked)
                        image.VirtualSize = new Size((int)(maxTimeForGraph < 1
? 1 : maxTimeForGraph), image.VirtualSize.Height);
                    image.AddPoint(new GraphImage.GraphPoint(new
Point((int)stopWatch.TotalTimeSpan.TotalSeconds, ii), ii.ToString() + ": " +
stopWatch.ToString()), UseGraphCheck.Checked);
                    if (UseGraphCheck.Checked)
                        UpdateGUI(ResultGraph, "Image", image.Bitmap);
                }

            }
            catch (System.Threading.ThreadAbortException) { }
            finally
            {
                try
                {
                    // At the end of testing we need to stop the
                    // stopwatch which is keeping track of the
                    // overall time.
                    testWatch.Stop();

                    // Display some summarising statiscal
                    // information to the user before leaving the
                    // thread.
                    WriteLine("The test process has finished and took: " +
testWatch.ToString());
                    System.Windows.Forms.MessageBox.Show("The test process has
finished and took: " + testWatch.ToString());

                    // Return everything back to the way it was.
                    UpdateGUI(TestingProgress, "Value", 0);
                    UpdateGUI(RunTestButton, "Text", "Run Test");
```

```
                    UpdateGUI(CompileButton, "Enabled", true);
                    UpdateGUI(OptimisedCheck, "Enabled", true);
                    testThread = null;

                    if (UseLogFileCheck.Checked)
                        CloseLogs();
                }
                catch (Exception) { }
            }
        }

        /// <summary>
        /// The RunTestButton Click method is call when the
        /// Run Test button is clicked. It is firstly
        /// responsible for instantiating the thread that is
        /// used for carrying out the testing process and
        /// secondly as a stop button if the testing process
        /// must be stopped.
        /// </summary>
        private void RunTestButton_Click(object sender, EventArgs e)
        {
            // It is important to ensure we have some rules
            // compiled before we can try and used them.
            if ((periodTypes != null) && (periodTypes.Length > 0))
            {

                // If the testThread is null then it means
                // the test is not already running and we
                // can act like we a start button.
                if (testThread == null)
                {
                    CompileButton.Enabled = false;
                    OptimisedCheck.Enabled = false;
                    testThread = new System.Threading.Thread(new
System.Threading.ThreadStart(TestMethod));
                    testThread.Start();
                    RunTestButton.Text = "Stop Test";
                }
                // Otherwise we need to act like a stop
                // button and stop the currently running
                // thread.
                else
                {
                    // Abort the thread.
                    testThread.Abort();
                }

            }
            else
            {
                // Notify the user that they need to compile
                // the rules first beforing pressing this
                // button.
                System.Windows.Forms.MessageBox.Show("You must first compile
the rules before they can be used.");
            }
        }

        /// <summary>
        /// The Radio_CheckedChanged event handler deals
        /// with the user changing the rule type which
        /// leads to the log files requiring closure and
        /// the rules recompiling prior to the execution
        /// of the test.
        /// </summary>
        private void Radio_CheckedChanged(object sender, EventArgs e)
        {
            RunTestButton.Enabled = false;

            if (LogsOpen)
                CloseLogs();
        }

        /// <summary>
        /// The OptimisedCheck_CheckChanged event handler
        /// deals with the user changing the rule type which
```

```
        /// leads to the log files requiring closure.
        /// </summary>
        private void OptimisedCheck_CheckedChanged(object sender, EventArgs e)
        {
            if (LogsOpen)
                CloseLogs();
        }

        /// <summary>
        /// The LogFilename property returns the full name
        /// of the log file (including path), excluding the
        /// files extension which is added later.
        /// </summary>
        private string LogFilename
        {
            get
            {
                string filename = (DrivingRadio.Checked ? "Driving" :
"Teaching") + "-" + (OptimisedCheck.Checked ? "Optimised" : "Not Optimised") +
"-" + Environment.MachineName;
                string directory =
System.IO.Path.Combine(Environment.CurrentDirectory, "Results");
                if (!System.IO.Directory.Exists(directory))
                    System.IO.Directory.CreateDirectory(directory);
                return System.IO.Path.Combine(directory, filename);
            }
        }

        /// <summary>
        /// The LogsOpen property returns a boolean
        /// describing whether or not the log files are
        /// currently open.
        /// </summary>
        private bool LogsOpen
        {
            get
            {
                return (logFile != null) && (patternFile != null);
            }
        }

        /// <summary>
        /// The OpenLogs method opens up the log files
        /// and prepares them for data.
        /// </summary>
        private void OpenLogs()
        {
            string logFilename = LogFilename;
            logFile = new StreamWriter(logFilename + ".csv", false);
            patternFile = new StreamWriter(logFilename + ".xml", false);

            WriteLine(GatherPCInfo());
        }

        /// <summary>
        /// The CloseLogs method closes the log files
        /// and disposes of any consumed resources.
        /// </summary>
        private void CloseLogs()
        {
            logFile.Close();
            logFile.Dispose();
            logFile = null;

            patternFile.Close();
            patternFile.Dispose();
            patternFile = null;
        }

        /// <summary>
        /// The GatherPCInfo method attempts to gather
        /// as much useful information about the PC on
        /// which the tests are being run as possible.
        /// The result is returned in CSV format.
        /// </summary>
        /// <returns>
```

```
        /// The return value is the PC information in
        /// CSV format.
        /// </returns>
        private string GatherPCInfo()
        {
            string PCInfo = "Rule Type,Optimised,Machine Name,OS
Version,Processor Count,Physical Memory,Processor Name,Processor
Architecture\n";

            PCInfo = (DrivingRadio.Checked ? "Driving" : "Teaching");
            PCInfo += "," + OptimisedCheck.Checked.ToString();
            PCInfo += "," + Environment.MachineName;
            PCInfo += "," + Environment.OSVersion.VersionString;
            PCInfo += "," + Environment.ProcessorCount.ToString();

            try
            {
                System.Management.ManagementScope scope = new
System.Management.ManagementScope(@"\\" + Environment.MachineName +
@"\root\cimv2");
                System.Management.ObjectQuery query = new
System.Management.ObjectQuery("SELECT TotalPhysicalMemory FROM
Win32 ComputerSystem");
                System.Management.ManagementObjectSearcher searcher = new
System.Management.ManagementObjectSearcher(scope, query);
                foreach (System.Management.ManagementObject manObject in
searcher.Get())
                    PCInfo += "," +
manObject["TotalPhysicalMemory"].ToString();
                query = new System.Management.ObjectQuery("SELECT Name,
AddressWidth FROM Win32_Processor");
                searcher = new
System.Management.ManagementObjectSearcher(scope, query);
                bool first = true;
                foreach (System.Management.ManagementObject manObject in
searcher.Get())
                {
                    if (!first)
                        PCInfo += "\n,,,,,,";
                    else
                        first = false;

                    PCInfo += "," + manObject["Name"].ToString().Trim();
                    PCInfo += "," +
manObject["AddressWidth"].ToString().Trim();
                }
            }
            catch (Exception){}


            PCInfo += "\n\n";
            return PCInfo;
        }

    }

}
```

11.4 Statistical Results

11.4.1 Non-Optimized Computational Performance Comparison between Different Systems Types

Dataset Size	A (ms)	B (ms)	C (ms)	D (ms)
250	2.121131	2.1144106	2.6054923	1.3076564
500	8.1978924	7.9926757	8.6268013	3.5852018
750	19.1669599	18.0037708	18.9059303	7.8951996
1000	32.8151193	32.2817301	33.4188347	14.2619564
1250	51.843355	51.782829	52.0582488	22.561753
1500	75.0367179	74.7958349	75.8231029	32.5905983
1750	103.3021251	102.632663	103.9087126	45.2243479
2000	136.1053152	135.5874543	137.3087879	58.3525804
2250	174.7682654	173.7262386	175.923315	75.7844256
2500	217.6239514	217.4988535	219.8100418	93.4561651
2750	273.2851524	267.5252966	274.2543505	114.6944904
3000	322.4277147	319.4072474	320.2797368	136.3386778
3250	380.5875245	378.4219758	382.2504744	161.5018241
3500	446.4783619	443.3373372	447.6035129	189.6480015
3750	511.706879	514.0705029	520.4708841	220.2501979
4000	589.9725778	591.7112564	598.0968001	253.5856205
4250	670.8721042	674.7248775	680.0368186	288.9188873
4500	759.0128928	765.2705554	772.2280004	327.1477147
4750	853.3551213	860.8725217	883.4990068	368.9246418
5000	954.8093779	964.7089148	974.107296	414.5897301
5250	1062.328443	1071.290858	1157.540748	461.0228689
5500	1174.567447	1189.7846	1306.598238	511.145673
5750	1296.810302	1312.948845	1386.33493	565.3978012
6000	1425.991125	1698.191417	1519.913738	621.4998092
6250	1571.45296	1879.437985	1657.776317	678.0658941
6500	1708.82946	2048.870693	1897.064694	741.0957948
6750	1855.977129	2272.064477	2060.625683	808.4446357
7000	2016.607565	2468.263954	2212.055744	883.0498376
7250	2175.50681	2680.622538	2561.137908	952.4107217
7500	2348.415683	2864.337802	2799.302159	1027.150615
7750	2543.120858	3005.231644	2940.729688	1113.152282
8000	2731.072022	3283.582762	3158.815875	1194.861925
8250	2912.222702	3240.96924	3520.600259	1281.190706
8500	3130.453128	3605.431754	3912.535619	1372.911803
8750	3367.729581	3723.995938	4330.491049	1468.950957
9000	3576.096247	3802.624605	4524.246588	1566.136846
9250	3818.797722	3854.005141	5197.440863	1672.440165
9500	4069.977992	4108.646172	5144.947749	1780.491122
9750	4339.91405	4342.745546	5965.402563	1887.615766
10000	4584.35188	4630.677551	6460.421876	2008.201329

11.4.2 Non-Optimized Memory Usage Comparison between Different System Types

Dataset Size	A (bytes)	B (bytes)	C (bytes)	D (bytes)
250	1892352	1466368	2068480	1449984
500	-520192	-200704	-770048	405504
750	28672	8192	-172032	86016
1000	0	233472	380928	520192
1250	118784	249856	98304	745472
1500	122880	151552	-135168	966656
1750	159744	-4096	360448	-1392640
2000	303104	-90112	45056	970752
2250	155648	253952	204800	-872448
2500	221184	200704	81920	258048
2750	49152	28672	229376	90112
3000	110592	180224	376832	139264
3250	245760	143360	114688	335872
3500	409600	155648	32768	-12288
3750	0	147456	208896	192512
4000	512000	163840	0	167936
4250	335872	188416	745472	221184
4500	237568	122880	49152	53248
4750	208896	233472	12288	176128
5000	-274432	151552	462848	450560
5250	143360	446464	-163840	-4096
5500	380928	61440	499712	110592
5750	204800	581632	-200704	229376
6000	249856	192512	434176	98304
6250	442368	159744	204800	188416
6500	270336	24576	65536	385024
6750	151552	122880	565248	188416
7000	323584	475136	-503808	139264
7250	299008	294912	389120	188416
7500	-819200	-143360	1044480	471040
7750	376832	434176	-77824	-327680
8000	249856	294912	-61440	192512
8250	286720	126976	-446464	180224
8500	159744	876544	385024	487424
8750	401408	-1101824	376832	131072
9000	339968	630784	90112	446464
9250	106496	241664	892928	-81920
9500	450560	-397312	-212992	335872
9750	425984	1245184	495616	49152
10000	323584	143360	-139264	139264

11.4.3 Optimized Computational Performance Comparison between Different Systems Types

Dataset Size	Ao (ms)	Bo (ms)	Co (ms)	Do (ms)
250	2.3214772	2.3199325	2.3108856	0.9973897
500	6.3086134	6.2920378	6.1381153	3.0384333
750	10.6791754	10.1471454	10.3788959	4.6658972
1000	15.0999852	14.3149544	14.59387	6.3043122
1250	19.6320347	18.7895509	18.9949388	8.2836141
1500	24.3383795	23.4240204	23.3381594	10.1369453
1750	29.2259553	28.118798	27.9535522	12.2594699
2000	34.0704105	32.9102408	32.9196247	14.7651465
2250	39.4948142	37.9844083	37.5923492	16.7341064
2500	44.7579122	43.1627167	42.6504508	19.2062232
2750	50.1647659	48.3107115	48.2990736	21.6638722
3000	56.1471216	53.7633117	54.5122232	23.9249371
3250	61.5078614	59.3450617	61.5991592	26.3754718
3500	66.997542	65.0738188	67.4458331	29.1392337
3750	72.9414267	71.4492515	73.7848273	31.6186158
4000	79.0083531	77.3670413	80.2959634	34.7724468
4250	85.3640209	82.8944853	88.0537153	36.8611504
4500	91.9473068	89.5383069	94.6162201	40.1460977
4750	98.4358151	96.0120618	103.0609943	42.903016
5000	104.5196004	102.6967023	118.2865162	45.7830836
5250	126.8958611	109.5162142	124.8440606	49.0656324
5500	120.2529596	126.9846921	131.3310059	51.9593727
5750	125.7306893	128.5850586	144.2375101	55.5484001
6000	133.5353997	130.4679727	156.5546857	57.8576487
6250	140.7630752	137.7585263	168.3008808	61.6195625
6500	148.1366396	145.6196585	178.0999352	65.6942723
6750	156.7677017	152.9561021	193.8302959	68.2733142
7000	163.6535102	160.6736286	206.7726095	71.9794876
7250	171.6392498	168.8987687	225.4659177	76.4792972
7500	178.8073018	176.9834868	247.7542793	79.8694518
7750	187.5948855	185.0822481	262.3499612	83.3411852
8000	195.559667	193.7304404	287.5864372	87.0561486
8250	205.1578467	202.049403	309.3298002	90.7879625
8500	212.9210193	210.7326892	334.2154761	94.021657
8750	223.631949	220.1550131	363.1418082	97.7400615
9000	233.7543992	228.3274464	385.8108397	101.6817358
9250	243.1498488	237.9274667	408.0126657	106.4492621
9500	253.8723231	246.8324842	440.9294809	110.406726
9750	263.5054274	256.3891297	472.9164355	115.4634155
10000	275.7516997	265.6141127	508.2205174	120.1221319

11.4.4 Optimized Memory Usage Comparison between Different System Types

Dataset Size	A (bytes)	B (bytes)	C (bytes)	D (bytes)
250	1892352	1515520	1892352	1454080
500	-520192	-225280	-495616	237568
750	28672	4096	-159744	151552
1000	0	4096	0	253952
1250	118784	0	53248	65536
1500	122880	139264	0	188416
1750	159744	167936	479232	143360
2000	303104	196608	32768	147456
2250	155648	151552	126976	176128
2500	221184	159744	196608	126976
2750	49152	180224	102400	176128
3000	110592	253952	241664	180224
3250	245760	204800	290816	151552
3500	409600	167936	282624	167936
3750	0	0	299008	176128
4000	512000	167936	-24576	135168
4250	335872	126976	4096	167936
4500	237568	155648	348160	155648
4750	208896	155648	57344	204800
5000	-274432	212992	339968	131072
5250	143360	118784	352256	180224
5500	380928	598016	266240	167936
5750	204800	339968	282624	143360
6000	249856	208896	290816	135168
6250	442368	188416	-532480	196608
6500	270336	-36864	0	163840
6750	151552	40960	389120	417792
7000	323584	126976	327680	278528
7250	299008	1093632	204800	-8192
7500	-819200	-405504	278528	237568
7750	376832	225280	360448	4096
8000	249856	135168	282624	516096
8250	286720	417792	397312	0
8500	159744	241664	315392	360448
8750	401408	327680	266240	32768
9000	339968	-724992	-970752	323584
9250	106496	40960	90112	0
9500	450560	602112	225280	32768
9750	425984	294912	290816	376832
10000	323584	270336	266240	8192

11.4.5 Computational Performance Comparison between Original and Incremental Testing Approaches

Dataset Size	A (ms)	Ao (ms)	B (ms)	Bo (ms)	C (ms)	Co (ms)	D (ms)	Do (ms)
250	2.121131	2.3214772	2.1144106	2.3199325	2.6054923	2.3108856	1.3076564	0.9973897
500	8.1978924	6.3086134	7.9926757	6.2920378	8.6268013	6.1381153	3.5852018	3.0384333
750	19.1669599	10.6791754	18.0037708	10.1471454	18.9059303	10.3788959	7.8951996	4.6658972
1000	32.8151193	15.0999852	32.2817301	14.3149544	33.4188347	14.59387	14.2619564	6.3043122
1250	51.843355	19.6320347	51.782829	18.7895509	52.0582488	18.9949388	22.561753	8.2836141
1500	75.0367179	24.3383795	74.7958349	23.4240204	75.8231029	23.3381594	32.5905983	10.1369453
1750	103.3021251	29.2259553	102.632663	28.118798	103.9087126	27.9535522	45.2243479	12.2594699
2000	136.1053152	34.0704105	135.5874543	32.9102408	137.3087879	32.9196247	58.3525804	14.7651465
2250	174.7682654	39.4948142	173.7262386	37.9844083	175.923315	37.5923492	75.7844256	16.7341064
2500	217.6239514	44.7579122	217.4988535	43.1627167	219.8100418	42.6504508	93.4561651	19.2062232
2750	273.2851524	50.1647659	267.5252966	48.3107115	274.2543505	48.2990736	114.6944904	21.6638722
3000	322.4277147	56.1471216	319.4072474	53.7633117	320.2797368	54.5122232	136.3386778	23.9249371
3250	380.5875245	61.5078614	378.4219758	59.3450617	382.2504744	61.5991592	161.5018241	26.3754718
3500	446.4783619	66.997542	443.3373372	65.0738188	447.6035129	67.4458331	189.6480015	29.1392337
3750	511.706879	72.9414267	514.0705029	71.4492515	520.4708841	73.7848273	220.2501979	31.6186158
4000	589.9725778	79.0083531	591.7112564	77.3670413	598.0968001	80.2959634	253.5856205	34.7724468
4250	670.8721042	85.3640209	674.7248775	82.8944853	680.0368186	88.0537153	288.9188873	36.8611504
4500	759.0128928	91.9473068	765.2705554	89.5383069	772.2280004	94.6162201	327.1477147	40.1460977
4750	853.3551213	98.4358151	860.8725217	96.0120618	883.4990068	103.0609943	368.9246418	42.903016
5000	954.8093779	104.5196004	964.7089148	102.6967023	974.107296	118.2865162	414.5897301	45.7830836
5250	1062.328443	126.8958611	1071.290858	109.5162142	1157.540748	124.8440606	461.0228689	49.0656324
5500	1174.567447	120.2529596	1189.7846	126.9846921	1306.598238	131.3310059	511.145673	51.9593727
5750	1296.810302	125.7306893	1312.948845	128.5850586	1386.33493	144.2375101	565.3978012	55.5484001
6000	1425.991125	133.5353997	1698.191417	130.4679727	1519.913738	156.5546857	621.4998092	57.8576487
6250	1571.45296	140.7630752	1879.437985	137.7585263	1657.776317	168.3008808	678.0658941	61.6195625
6500	1708.82946	148.1366396	2048.870693	145.6196585	1897.064694	178.0999352	741.0957948	65.6942723
6750	1855.977129	156.7677017	2272.064477	152.9561021	2060.625683	193.8302959	808.4446357	68.2733142
7000	2016.607565	163.6535102	2468.263954	160.6736286	2212.055744	206.7726095	883.0498376	71.9794876
7250	2175.50681	171.6392498	2680.622538	168.8987687	2561.137908	225.4659177	952.4107217	76.4792972
7500	2348.415683	178.8073018	2864.337802	176.9834868	2799.302159	247.7542793	1027.150615	79.8694518
7750	2543.120858	187.5948855	3005.231644	185.0822481	2940.729688	262.3499612	1113.152282	83.3411852
8000	2731.072022	195.559667	3283.582762	193.7304404	3158.815875	287.5864372	1194.861925	87.0561486
8250	2912.222702	205.1578467	3240.96924	202.049403	3520.600259	309.3298002	1281.190706	90.7879625
8500	3130.453128	212.9210193	3605.431754	210.7326892	3912.535619	334.2154761	1372.911803	94.021657
8750	3367.729581	223.631949	3723.995938	220.1550131	4330.491049	363.1418082	1468.950957	97.7400615
9000	3576.096247	233.7543992	3802.624605	228.3274464	4524.246588	385.8108397	1566.136846	101.6817358
9250	3818.797722	243.1498488	3854.005141	237.9274667	5197.440863	408.0126657	1672.440165	106.4492621
9500	4069.977992	253.8723231	4108.646172	246.8324842	5144.947749	440.9294809	1780.491122	110.406726
9750	4339.91405	263.5054274	4342.745546	256.3891297	5965.402563	472.9164355	1887.615766	115.4634155
10000	4584.35188	275.7516997	4630.677551	265.6141127	6460.421876	508.2205174	2008.201329	120.1221319

11.4.6 Memory Comparison between Original and Incremental Testing Approaches

Dataset Size	A (bytes)	Ao (bytes)	B (bytes)	Bo (bytes)	C (bytes)	Co (bytes)	D (bytes)	Do (bytes)
250	1892352	1892352	1466368	1515520	2068480	1892352	1449984	1454080
500	-520192	-520192	-200704	-225280	-770048	-495616	405504	237568
750	28672	28672	8192	4096	-172032	-159744	86016	151552
1000	0	0	233472	4096	380928	0	520192	253952
1250	118784	118784	249856	0	98304	53248	745472	65536
1500	122880	122880	151552	139264	-135168	0	966656	188416
1750	159744	159744	-4096	167936	360448	479232	-1392640	143360
2000	303104	303104	-90112	196608	45056	32768	970752	147456
2250	155648	155648	253952	151552	204800	126976	-872448	176128
2500	221184	221184	200704	159744	81920	196608	258048	126976
2750	49152	49152	28672	180224	229376	102400	90112	176128
3000	110592	110592	180224	253952	376832	241664	139264	180224
3250	245760	245760	143360	204800	114688	290816	335872	151552
3500	409600	409600	155648	167936	32768	282624	-12288	167936
3750	0	0	147456	0	208896	299008	192512	176128
4000	512000	512000	163840	167936	0	-24576	167936	135168
4250	335872	335872	188416	126976	745472	4096	221184	167936
4500	237568	237568	122880	155648	49152	348160	53248	155648
4750	208896	208896	233472	155648	12288	57344	176128	204800
5000	-274432	-274432	151552	212992	462848	339968	450560	131072
5250	143360	143360	446464	118784	-163840	352256	-4096	180224
5500	380928	380928	61440	598016	499712	266240	110592	167936
5750	204800	204800	581632	339968	-200704	282624	229376	143360
6000	249856	249856	192512	208896	434176	290816	98304	135168
6250	442368	442368	159744	188416	204800	-532480	188416	196608
6500	270336	270336	24576	-36864	65536	0	385024	163840
6750	151552	151552	122880	40960	565248	389120	188416	417792
7000	323584	323584	475136	126976	-503808	327680	139264	278528
7250	299008	299008	294912	1093632	389120	204800	188416	-8192
7500	-819200	-819200	-143360	-405504	1044480	278528	471040	237568
7750	376832	376832	434176	225280	-77824	360448	-327680	4096
8000	249856	249856	294912	135168	-61440	282624	192512	516096
8250	286720	286720	126976	417792	-446464	397312	180224	0
8500	159744	159744	876544	241664	385024	315392	487424	360448
8750	401408	401408	-1101824	327680	376832	266240	131072	32768
9000	339968	339968	630784	-724992	90112	-970752	446464	323584
9250	106496	106496	241664	40960	892928	90112	-81920	0
9500	450560	450560	-397312	602112	-212992	225280	335872	32768
9750	425984	425984	1245184	294912	495616	290816	49152	376832
10000	323584	323584	143360	270336	-139264	266240	139264	8192

11.5 Additional Statistical Results

11.5.1 Debugging Comparison Data

OS	Windows	Windows	Windows	Windows
CLI	.NET	.NET	.NET	.NET
Type	Release	Debug	Release	Debug
Rule Type	Driving	Driving	Driving	Driving
Type	Execution Time	Execution Time	Memory Usage	Memory Usage
Ruleset Size	4	4	4	4
350	2.2708115	4.7984122	123668	125492
700	6.7068211	13.6877015	123920	123932
1050	11.4093391	22.8257272	122896	122908
1400	16.4235667	33.6709557	124944	124956
1750	21.7416021	42.287626	122844	122696
2100	27.4684932	45.753133	122896	122720
2450	33.2454361	58.0970082	122896	122920
2800	39.5354209	64.9266247	127084	127016
3150	46.0853273	75.723661	122896	122920
3500	53.5157617	90.1459084	122896	122932
3850	59.8879092	96.7866207	122896	122932
4200	67.3466773	108.1861377	122896	122696
4550	75.0248593	126.1768859	122896	122920
4900	83.6507775	132.8685535	122896	122920
5250	91.4086785	151.7547595	131100	131112
5600	99.7602862	163.0050182	122908	122908
5950	109.1481726	176.9747577	122884	122708
6300	117.560076	192.1325016	122908	122696
6650	128.3772642	206.9079621	122908	122908
7000	136.8501832	223.0690923	122908	122932
7350	147.6278337	234.4729184	122856	122908
7700	157.1367263	253.5754819	122908	122616
8050	167.951855	271.8297694	122908	122696
8400	178.9325868	293.0755435	122908	122696
8750	189.6954543	305.6181414	122884	122696
9100	201.4841755	321.9852067	122896	122920
9450	213.0496504	342.0356244	122920	122628
9800	222.4056473	366.334375	122908	122908

11.5.2 Driving and Teaching Comparison Data

PC	Laptop	Laptop	Laptop	Laptop	Laptop	Laptop
OS	Windows	Windows	Windows	Windows	Windows	Windows
CLI	.NET	.NET	.NET	.NET	.NET	.NET
Rule Type	Driving	Teaching	Driving	Teaching	Driving	Teaching
Type	Execution Time	Execution Time	Memory Usage	Memory Usage	Execution Time (Per Rule)	Execution Time (Per Rule)
Ruleset Size	4	6	4	6	4	6
350	2.2708115	6.7355109	123668	162660	0.567702875	1.12258515
700	6.7068211	19.3681432	123920	161144	1.676705275	3.228023867
1050	11.4093391	32.4878015	122896	161388	2.852334775	5.414633583
1400	16.4235667	45.9341657	124944	160876	4.105891675	7.655694283
1750	21.7416021	58.2492159	122844	160888	5.435400525	9.70820265
2100	27.4684932	71.4208377	122896	161912	6.8671233	11.90347295
2450	33.2454361	84.4929088	122896	160848	8.311359025	14.08215147
2800	39.5354209	98.247774	127084	160836	9.883855225	16.374629
3150	46.0853273	110.612434	122896	160900	11.52133183	18.43540567
3500	53.5157617	124.3647354	122896	160900	13.37894043	20.7274559
3850	59.8879092	137.8270234	122896	162948	14.9719773	22.97117057
4200	67.3466773	151.7478355	122896	160900	16.83666933	25.29130592
4550	75.0248593	164.4954714	122896	160900	18.75621483	27.4159119
4900	83.6507775	177.199175	122896	160912	20.91269438	29.53319583
5250	91.4086785	188.9595634	131100	160900	22.85216963	31.49326057
5600	99.7602862	203.4744266	122908	160888	24.94007155	33.91240443
5950	109.1481726	218.2452412	122884	160900	27.28704315	36.37420687
6300	117.560076	232.1783336	122908	160900	29.390019	38.69638893
6650	128.3772642	246.5556372	122908	160888	32.09431605	41.0926062
7000	136.8501832	259.4383615	122908	160912	34.2125458	43.23972692
7350	147.6278337	272.3713778	122856	164996	36.90695843	45.39522963
7700	157.1367263	285.8541732	122908	160888	39.28418158	47.6423622
8050	167.951855	301.7770287	122908	160924	41.98796375	50.29617145
8400	178.9325868	315.2897988	122908	160888	44.7331467	52.5482998
8750	189.6954543	329.7649752	122884	160912	47.42386358	54.9608292
9100	201.4841755	340.6321735	122896	160888	50.37104388	56.77202892
9450	213.0496504	353.0107616	122920	160888	53.2624126	58.83512693
9800	222.4056473	367.3325198	122908	160912	55.60141183	61.22208663

11.5.3 .NET Framework and Mono Comparison

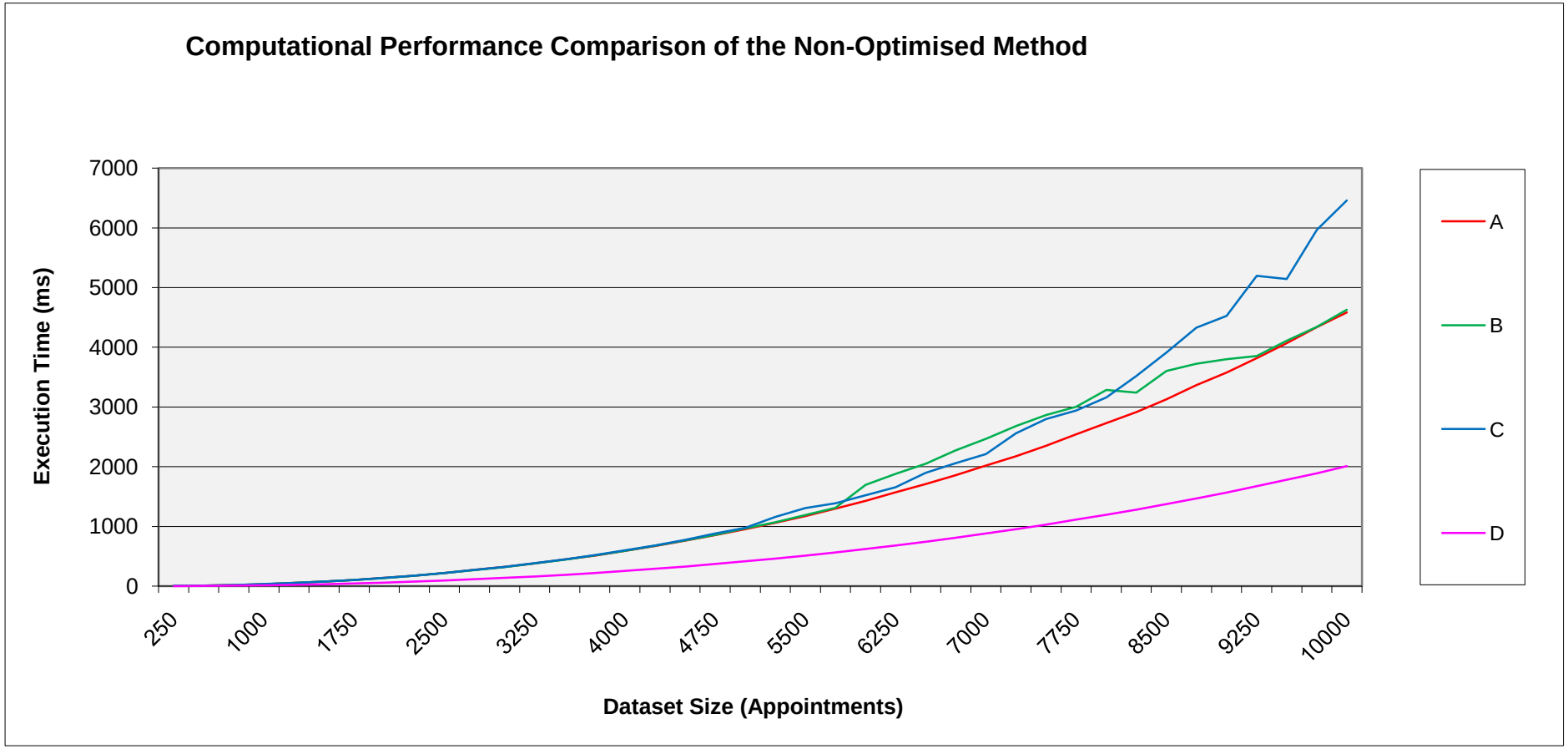
OS	Windows	Windows	Windows	Windows	Windows	Windows	Windows	Windows
CLI	.NET	Mono	.NET	Mono	.NET	Mono	.NET	Mono
Rule Type	Driving	Driving	Teaching	Teaching	Driving	Driving	Teaching	Teaching
Type	Execution Time	Execution Time	Execution Time	Execution Time	Memory Usage	Memory Usage	Memory Usage	Memory Usage
Ruleset Size	4	4	6	6	4	4	6	6
350	0.940367	2.1356257	2.512043	5.9967929	125480	684032	162672	811008
700	2.4244283	6.1643071	7.1877475	17.2681175	123920	507904	161144	868352
1050	4.1640293	10.2986466	11.9758951	28.7088971	122896	577536	161388	872448
1400	5.9633657	14.5003125	16.7126051	40.06055	124944	675840	160876	880640
1750	7.8493779	18.8311513	21.5240209	51.7900194	122896	892928	160876	487424
2100	9.9428135	23.5770786	26.4473386	63.4245442	122896	446464	161912	983040
2450	12.0693448	28.4929187	31.3082004	74.8445591	122884	286720	160876	606208
2800	14.409064	32.7349734	36.2136692	86.7814866	127016	770048	160876	348160
3150	16.7422045	37.7156676	41.1577076	98.9721839	122908	524288	160888	1064960
3500	19.2503183	42.866005	46.0421899	110.0372197	122908	360448	160924	704512
3850	21.903423	48.1852345	50.9382799	122.455867	122908	286720	162936	393216
4200	24.9346463	53.6688353	55.9422836	134.0914155	122908	196608	160912	360448
4550	27.5042807	58.8736596	60.8706037	147.5382012	122908	716800	160888	253952
4900	30.4847577	64.1276497	65.9459293	156.5253127	122908	491520	160888	1019904
5250	33.5198588	69.9474188	71.0116934	169.9591239	131100	393216	160888	679936
5600	36.7588595	75.976255	76.0632192	181.0123301	122908	319488	160912	552960
5950	40.1552117	82.3122912	81.2392792	194.4971002	122920	253952	160876	462848
6300	43.4988587	88.3417201	86.3857638	207.0985041	122708	192512	160900	307200
6650	46.7029583	95.1026631	91.4367819	219.315664	122908	172032	160888	258048
7000	50.28415	100.7880526	96.5020238	233.4232276	122920	786432	160876	212992
7350	54.036004	107.4059336	101.762502	240.0555882	122908	643072	165008	978944
7700	57.8224444	114.1855473	107.0122109	254.1005611	122908	507904	160888	823296
8050	61.6963958	120.8672123	111.6681824	265.4703642	122908	339968	160912	589824
8400	65.6528581	128.38958	116.9501447	278.1429597	122908	294912	160876	520192
8750	69.8200607	136.010257	122.3060444	290.143252	122908	278528	160912	434176
9100	74.0862254	143.7450961	127.5383723	303.2372554	122908	212992	160900	425984
9450	78.3983528	149.2690066	132.6939145	318.4178241	122908	180224	160900	290816
9800	82.8901948	156.8186315	138.1377541	330.2126064	122908	172032	160900	253952

11.5.4 Mono Cross Platform Comparison

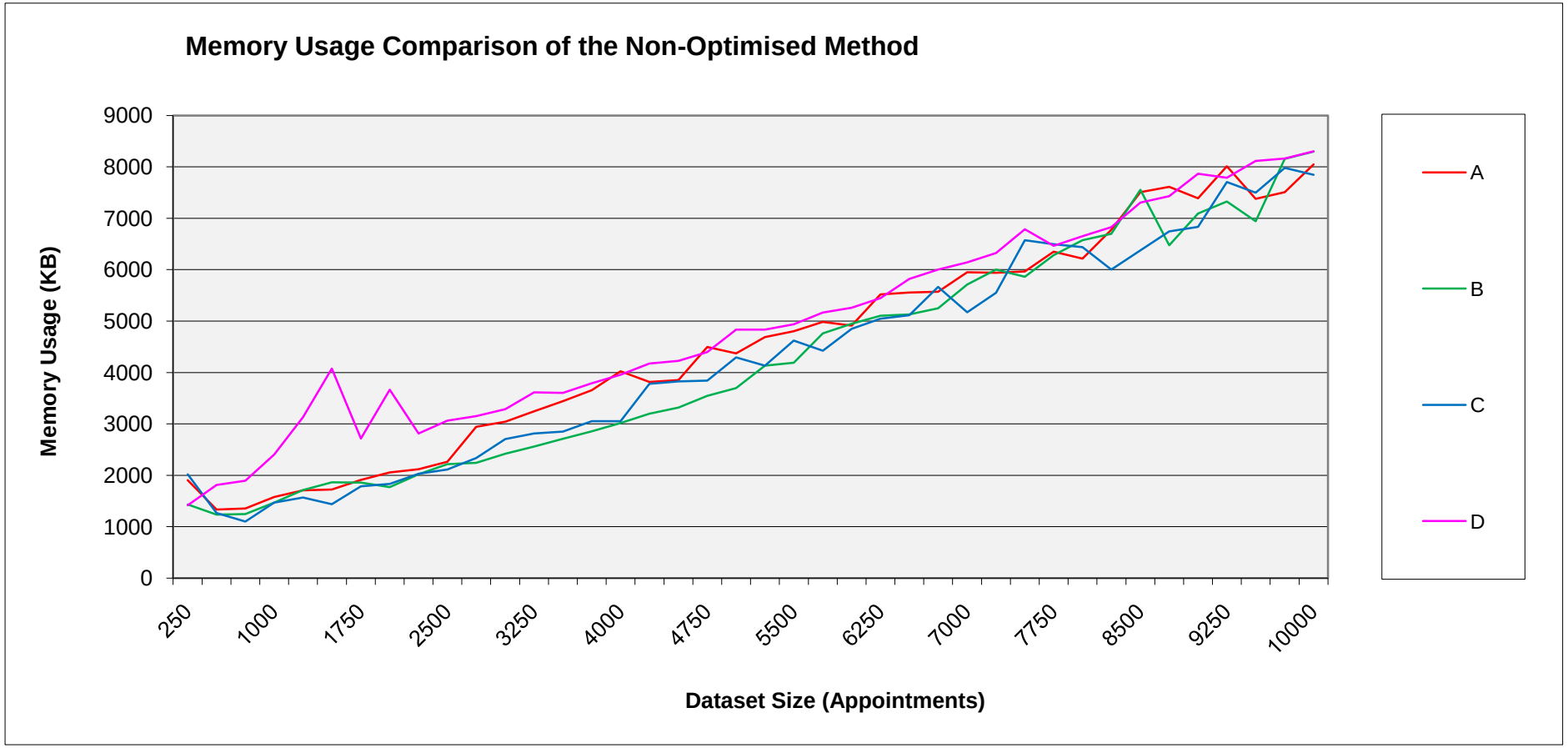
OS	Mac	Linux	Windows	Mac	Linux	Windows
CLI	Mono	Mono	Mono	Mono	Mono	Mono
Rule Type	Driving	Driving	Driving	Driving	Driving	Driving
Type	Execution Time	Execution Time	Execution Time	Memory Usage	Memory Usage	Memory Usage
Ruleset Size	4	4	4	4	4	4
350	3.256027	2.997562	2.8520146	606208	475136	667648
700	9.607472	8.789322	8.3588928	360448	294912	532480
1050	16.264927	15.402301	13.9745554	282624	225280	598016
1400	23.081787	20.964458	19.6221635	356352	258048	634880
1750	30.187402	27.386591	25.8133186	270336	258048	368640
2100	37.184974	33.842762	31.914713	364544	266240	745472
2450	44.721522	40.538606	38.6429681	290816	237568	471040
2800	52.38203	47.411971	45.3683806	253952	229376	241664
3150	60.182381	55.261759	51.3329312	249856	335872	745472
3500	67.696776	62.401649	58.0630355	417792	241664	499712
3850	75.947754	68.675682	65.354792	286720	225280	348160
4200	84.311568	77.546543	72.9895625	270336	245760	245760
4550	92.846972	83.482525	79.6552969	253952	221184	753664
4900	101.436775	91.339421	86.9808264	262144	208896	499712
5250	110.389199	98.796981	94.3014782	299008	319488	503808
5600	118.491398	106.864605	102.1372986	368640	233472	331776
5950	127.546114	114.864416	110.7330398	299008	245760	233472
6300	136.907924	123.099349	119.2910254	282624	233472	217088
6650	146.128813	131.456938	128.1504718	237568	258048	184320
7000	156.119686	140.086192	135.300561	299008	200704	798720
7350	165.568019	148.746469	145.2213854	270336	217088	577536
7700	175.944338	157.638874	154.4275641	253952	225280	421888
8050	186.255216	166.613302	162.6278624	229376	270336	430080
8400	195.027877	175.069012	170.3600927	401408	278528	339968
8750	205.634389	184.309118	179.8047175	327680	249856	282624
9100	216.286848	193.676518	189.6472196	274432	225280	204800
9450	226.796797	203.141792	200.5756898	274432	233472	172032
9800	237.487685	212.779476	210.2036643	282624	217088	172032

11.6 Charts

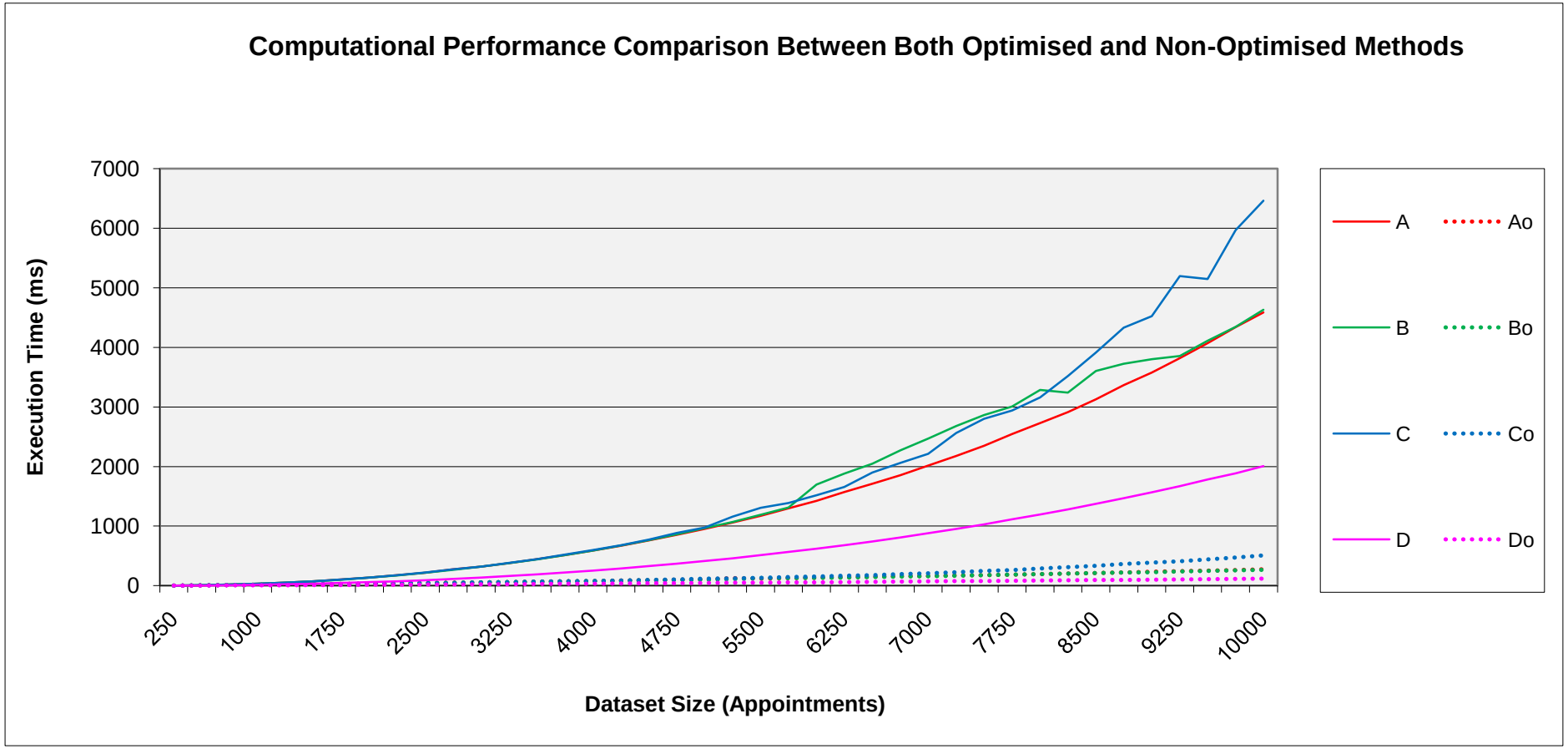
11.6.1 Computational Performance Comparison of the Non-Optimised Method



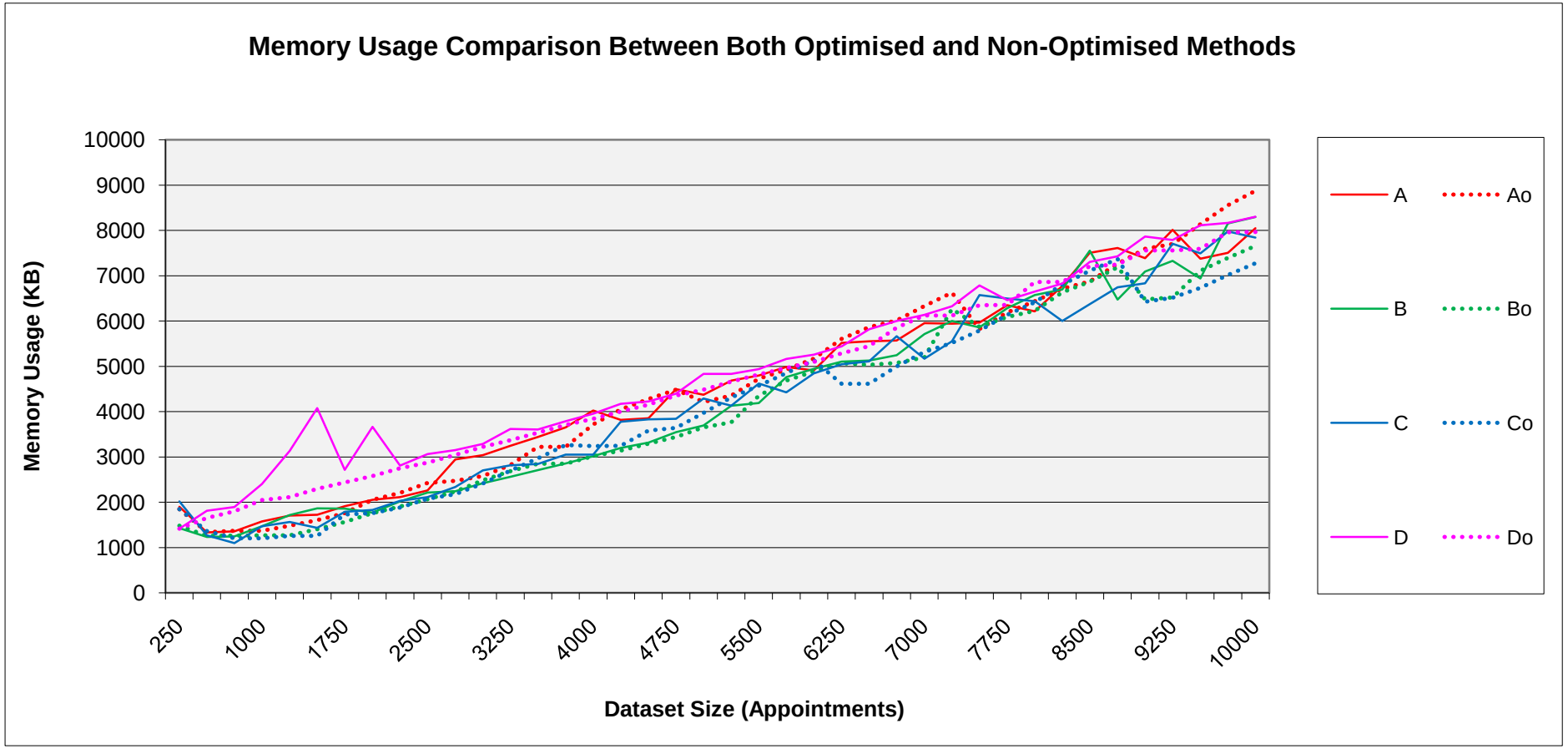
11.6.2 Memory Usage Comparison of the Non-Optimised Method



11.6.3 Computational Performance Comparison Between Both Optimised and Non-Optimised Methods



11.6.4 Memory Usage Comparison Between Both Optimised and Non-Optimised Methods



11.7 Rule Descriptions

11.7.1 Driving Rules

```
<?xml version="1.0"?>
<ArrayOfPeriod xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Period xsi:type="WP">
    <Description>
      <Name>DWP</Name>
      <Description>Daily Work Period</Description>
      <PrePosts>
        <PrePost>
          <Pre />
          <Post>DRP</Post>
        </PrePost>
        <PrePost>
          <Pre>DRP</Pre>
          <Post>DRP</Post>
        </PrePost>
        <PrePost>
          <Pre>DRP</Pre>
          <Post />
        </PrePost>
        <PrePost>
          <Pre />
          <Post />
        </PrePost>
      </PrePosts>
      <Contains>
        <Contain>
          <Name>Appointment</Name>
        </Contain>
      </Contains>
    </Description>
    <Rules xsi:type="Comparison">
      <Operation>LessThanOrEqualTo</Operation>
      <Values>
        <Value>
          <Description>WorkLength</Description>
          <Type>Parameter</Type>
        </Value>
        <Value>
          <Description>8</Description>
          <Type>Hours</Type>
        </Value>
      </Values>
    </Rules>
  </Period>
  <Period xsi:type="WP">
    <Description>
      <Name>WWP</Name>
      <Description>Weekly Work Period</Description>
      <PrePosts>
        <PrePost>
          <Pre>WRP</Pre>
          <Post>WRP</Post>
        </PrePost>
        <PrePost>
          <Pre />
          <Post>WRP</Post>
        </PrePost>
        <PrePost>
          <Pre>WRP</Pre>
          <Post />
        </PrePost>
        <PrePost>
          <Pre />
          <Post />
        </PrePost>
```

```
      </PrePosts>
      <Contains>
        <Contain>
          <Name>DWP</Name>
        </Contain>
        <Contain>
          <Name>DRP</Name>
        </Contain>
      </Contains>
    </Description>
    <Rules xsi:type="Comparison">
      <Operation>LessThanOrEqualTo</Operation>
      <Values>
        <Value>
          <Description>DWP.Count</Description>
          <Type>Parameter</Type>
        </Value>
        <Value>
          <Description>6</Description>
          <Type>Number</Type>
        </Value>
      </Values>
    </Rules>
  </Period>
  <Period xsi:type="RP">
    <Description>
      <Name>WRP</Name>
      <Description>Weekly Rest Period</Description>
      <PrePosts>
        <PrePost>
          <Pre>WWP</Pre>
          <Post>WWP</Post>
        </PrePost>
      </PrePosts>
      <Contains />
    </Description>
    <Rules xsi:type="Comparison">
      <Operation>GreaterThanOrEqualTo</Operation>
      <Values>
        <Value>
          <Description>Length</Description>
          <Type>Parameter</Type>
        </Value>
        <Value>
          <Description>30</Description>
          <Type>Hours</Type>
        </Value>
      </Values>
    </Rules>
  </Period>
  <Period xsi:type="RP">
    <Description>
      <Name>DRP</Name>
      <Description>Daily Rest Period</Description>
      <PrePosts>
        <PrePost>
          <Pre>DWP</Pre>
          <Post>DWP</Post>
        </PrePost>
      </PrePosts>
      <Contains />
    </Description>
    <Rules xsi:type="AndCondition">
      <Conditions>
        <Comparitor xsi:type="Comparison">
          <Operation>GreaterThanOrEqualTo</Operation>
          <Values>
            <Value>
              <Description>Length</Description>
              <Type>Parameter</Type>
            </Value>
            <Value>
              <Description>8</Description>
              <Type>Hours</Type>
            </Value>
          </Values>
```

```
        </Comparitor>
        <Comparitor xsi:type="Comparison">
          <Operation>LessThan</Operation>
          <Values>
            <Value>
              <Description>Length</Description>
              <Type>Parameter</Type>
            </Value>
            <Value>
              <Description>30</Description>
              <Type>Hours</Type>
            </Value>
          </Values>
        </Comparitor>
      </Conditions>
    </Rules>
  </Period>
</ArrayOfPeriod>
```

11.7.2 Teaching Rules

```
<?xml version="1.0"?>
<ArrayOfPeriod xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Period xsi:type="WP">
    <Description>
      <Name>S</Name>
      <Description>A shift of work</Description>
      <PrePosts>
        <PrePost>
          <Pre />
          <Post />
        </PrePost>
        <PrePost>
          <Pre />
          <Post>SB</Post>
        </PrePost>
        <PrePost>
          <Pre>SB</Pre>
          <Post>SB</Post>
        </PrePost>
        <PrePost>
          <Pre>SB</Pre>
          <Post />
        </PrePost>
      </PrePosts>
      <Contains>
        <Contain>
          <Name>Appointment</Name>
        </Contain>
      </Contains>
    </Description>
    <Rules xsi:type="AndCondition">
      <Conditions>
        <Comparitor xsi:type="Comparison">
          <Operation>LessThanOrEqualTo</Operation>
          <Values>
            <Value>
              <Description>Length</Description>
              <Type>Parameter</Type>
            </Value>
            <Value>
              <Description>4</Description>
              <Type>Hours</Type>
            </Value>
          </Values>
        </Comparitor>
        <Comparitor xsi:type="Comparison">
          <Operation>GreaterThanOrEqualTo</Operation>
          <Values>
            <Value>
```

```
              <Description>Start.Hour</Description>
              <Type>Parameter</Type>
            </Value>
            <Value>
              <Description>9</Description>
              <Type>Number</Type>
            </Value>
          </Values>
        </Comparitor>
        <Comparitor xsi:type="Comparison">
          <Operation>LessThanOrEqualTo</Operation>
          <Values>
            <Value>
              <Description>Finish.Hour</Description>
              <Type>Parameter</Type>
            </Value>
            <Value>
              <Description>23</Description>
              <Type>Number</Type>
            </Value>
          </Values>
        </Comparitor>
      </Conditions>
    </Rules>
  </Period>
  <Period xsi:type="RP">
    <Description>
      <Name>SB</Name>
      <Description>A break between work shifts</Description>
      <PrePosts>
        <PrePost>
          <Pre>S</Pre>
          <Post>S</Post>
        </PrePost>
      </PrePosts>
      <Contains />
    </Description>
    <Rules xsi:type="Comparison">
      <Operation>GreaterThanOrEqualTo</Operation>
      <Values>
        <Value>
          <Description>RestLength</Description>
          <Type>Parameter</Type>
        </Value>
        <Value>
          <Description>1</Description>
          <Type>Hours</Type>
        </Value>
      </Values>
    </Rules>
  </Period>
  <Period xsi:type="WP">
    <Description>
      <Name>DWP</Name>
      <Description>A daily work period</Description>
      <PrePosts>
        <PrePost>
          <Pre />
          <Post />
        </PrePost>
        <PrePost>
          <Pre />
          <Post>DRP</Post>
        </PrePost>
        <PrePost>
          <Pre>DRP</Pre>
          <Post>DRP</Post>
        </PrePost>
        <PrePost>
          <Pre>DRP</Pre>
          <Post />
        </PrePost>
      </PrePosts>
      <Contains>
        <Contain>
          <Name>S</Name>
```

```
        </Contain>
        <Contain>
          <Name>SB</Name>
        </Contain>
      </Contains>
    </Description>
    <Rules xsi:type="AndCondition">
      <Conditions>
        <Comparitor xsi:type="Comparison">
          <Operation>LessThanOrEqualTo</Operation>
          <Values>
            <Value>
              <Description>Length</Description>
              <Type>Parameter</Type>
            </Value>
            <Value>
              <Description>9</Description>
              <Type>Hours</Type>
            </Value>
          </Values>
        </Comparitor>
        <Comparitor xsi:type="Comparison">
          <Operation>LessThanOrEqualTo</Operation>
          <Values>
            <Value>
              <Description>S.Count</Description>
              <Type>Parameter</Type>
            </Value>
            <Value>
              <Description>3</Description>
              <Type>Number</Type>
            </Value>
          </Values>
        </Comparitor>
      </Conditions>
    </Rules>
  </Period>
  <Period xsi:type="RP">
    <Description>
      <Name>DRP</Name>
      <Description>A daily rest period</Description>
      <PrePosts>
        <PrePost>
          <Pre>DWP</Pre>
          <Post>DWP</Post>
        </PrePost>
      </PrePosts>
      <Contains />
    </Description>
    <Rules xsi:type="Comparison">
      <Operation>GreaterThanOrEqualTo</Operation>
      <Values>
        <Value>
          <Description>Length</Description>
          <Type>Parameter</Type>
        </Value>
        <Value>
          <Description>15</Description>
          <Type>Hours</Type>
        </Value>
      </Values>
    </Rules>
  </Period>
  <Period xsi:type="WP">
    <Description>
      <Name>WWP</Name>
      <Description>A weekly work period</Description>
      <PrePosts>
        <PrePost>
          <Pre />
          <Post />
        </PrePost>
        <PrePost>
          <Pre />
          <Post>WRP</Post>
        </PrePost>
```

```
        <PrePost>
          <Pre>WRP</Pre>
          <Post />
        </PrePost>
        <PrePost>
          <Pre>WRP</Pre>
          <Post>WRP</Post>
        </PrePost>
      </PrePosts>
      <Contains>
        <Contain>
          <Name>DWP</Name>
        </Contain>
        <Contain>
          <Name>DRP</Name>
        </Contain>
      </Contains>
    </Description>
    <Rules xsi:type="Comparison">
      <Operation>LessThanOrEqualTo</Operation>
      <Values>
        <Value>
          <Description>DWP.Count</Description>
          <Type>Parameter</Type>
        </Value>
        <Value>
          <Description>5</Description>
          <Type>Number</Type>
        </Value>
      </Values>
    </Rules>
  </Period>
  <Period xsi:type="RP">
    <Description>
      <Name>WRP</Name>
      <Description>A weekly rest period</Description>
      <PrePosts>
        <PrePost>
          <Pre>WWP</Pre>
          <Post>WWP</Post>
        </PrePost>
      </PrePosts>
      <Contains />
    </Description>
    <Rules xsi:type="Comparison">
      <Operation>GreaterThanOrEqualTo</Operation>
      <Values>
        <Value>
          <Description>Length</Description>
          <Type>Parameter</Type>
        </Value>
        <Value>
          <Description>2</Description>
          <Type>Days</Type>
        </Value>
      </Values>
    </Rules>
  </Period>
</ArrayOfPeriod>
```