# A Rule Based User Interface Builder for Visual Studio .NET

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Science at Massey University, New Zealand

Richard Harry Wilburn

2007

# Abstract

Current popularity and lack of successful innovation in the field of Graphical User Interface (GUI) builders leads to the question of how we can pave the way for a second generation of GUI builders. This question requires a new approach on GUI builder innovation by changing event handling practices to integrate a Domain Specific Language (DSL). We propose a DSL based on R2ML that can be pre-compiled to .NET framework source code. The adoption of a DSL provides a starting point but offers similar problems with large numbers of rules like other previous unsuccessful innovations. We attempt to mitigate this concern with the adoption of an event correlation architecture which enables the realization of complex events. Complex events allow for the combining of primitive events to gain a higher level event which we propose is easier to relate to user requirements. We further reduce the number of rules developers require by introducing querying techniques to provide indirect referencing, rather than using traditional URI approaches which are more tightly coupled. Comparison of the lines of code our solution requires, against a comparison not using our solution, demonstrates a decrease in effort for developers. We also provide architectural reasoning to show developers the design benefits of our approach.

# Acknowledgements

I would like to thank my supervisor Dr Jens Dietrich, who not only provided a great level of support and knowledge of rule based systems but also provided an enthusiasm to understanding the .NET way of thinking as his preferred environment is currently Java.

Another group of people who deserve thanks is my family. I would like to thank both my parents whose aid has largely put me through university. Towards the end of my research however I required a quiet place to write my thesis and for that I must thank my grandparents who donated their garage, which is now filled with computers and whiteboards.

Help with implementation aspects of this project were provided by the .NET community mainly via the MSDN forums (Microsoft). Another community group that deserves thanks is the Castle Project, who provided a key library for this project which has definitely made the implementation easier than it would have otherwise been.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Research Overview

“User interface software is often large, complex and difficult to implement, debug, and modify” (Myers, User Interface Software Tools, 1994).

While the previous quote is a bold statement, the large number of Graphic User Interface (GUI) builders on the market in comparison to data oriented tools such as class diagram editors could be indicative of this issue. In this chapter we investigate GUI builder tools to see where they fall short in aiding developers.

In this research we investigate a way to abstract events to a level where they are more usable. Event driven programming is an approach to programming that facilitates system and user actions to dictate the flow of a program. The previous way of programming applications was to use batch programming where the flow of the program was dictated by the programmer. The convenience of event driven programming is that it abstracts code closer to user requirements.

A problem with event driven programming currently is that it is not as convenient as it could be in that it works at a low level. An example of low level operation is the events ‘Click’ and ‘KeyPress’. This is compared to user requirements which are often described by high level user actions. An example requirement could be that a user starts to print, hence disable all GUI controls that trigger printing. That example illustrates a gap between the low level nature of event driven programming in comparison to user requirements. This thesis attempts to address this gap by raising the abstraction of programming language events.

There is also a problem that developers face when attempting to separate concerns which arises from the high coupling of the User Interface (UI) to the events which belong to the UI. High coupling prevents developers from creating ‘event wire ups’ until a user interface has been constructed. Wiring up events is a process of a listening object subscribing to an event that is exposed on another object. High coupling can lead to architecturally problematic coding practices.

A modern market push from software products (such as the Microsoft Expression Family) tends to show an increasing ability to enable the option of outsourcing UI design to specialist designers. This would currently leave wiring of events and data binding (linking data with UI) until the user interface has been completed. This gives developers less time to wire up the UI and also gives less time to the UI developers as they have people dependent on their work.

The thesis also investigates rules as a way of specifying relationships. Rules have predominantly been used in software engineering for specifying mappings between two data sources and has been used for specifying higher domain logic. Specifying higher domain logic in the form of rules is often beneficial to software users as they can change the rules often without changing or understanding the software. We investigate both avenues as methods of abstraction from low level event architectures employed by current programming languages and frameworks. If there is one thing to come out of this research, we would like it to be the advancement of event handling of modern programming languages. Modern programming languages such as C# have improved on event handling by decentralizing events from an event loop allowing for a more lazy approach by adopting functional mechanisms. These functional mechanisms allow for easy subscription to events after the source code is compiled. With an event loop, events are declared in a central location making runtime subscription more difficult. While there have been functional improvements to modern languages, many use cases still fall short of their potential due to the gap between user requirements and the hardware mindset of events such as ‘Click’ and ‘KeyPress’.

## 1.2 Overview of Thesis

In this chapter we introduce the research areas of this thesis and discuss aspects such as the goals and scope. In the next chapter we look into the background information relevant to the thesis and extend on some areas introduced in this chapter such as querying and tagging techniques. In chapter 3 we look at the theory involved in the chosen relevant areas mentioned in chapter 2. We look at parts of implementation, but at a conceptual level. In chapter 4 we address the implementation of the project. Chapter 4 also looks at how the source code has been laid out, how it is deployed and the problems that arose. In chapter 5 we look at validating our claims made in chapter 1. This chapter provides the information that enables us to draw conclusions in chapter 6. Beyond chapter 6 we have the appendices which provide additional information, such as use cases and code samples.

## 1.3 Goals and Scope of Research

When taking on this research, goals were established to measure progress and scope was established to maintain course. In the following sections we discuss the goals and scope.

### 1.3.1 Goals

Firstly, we need to approach the design and integration of a Domain Specific Language(DSL) for event handling. The DSL will be based on existing event-condition-action rule standards. The DSL will then be used to generate rules that the .NET framework can understand.

In chapter 2 we discover a requirement for the size of DSL rules to be minimal. To achieve this, support for compact event handling specifications must be investigated. A starting point for compacting the DSL is later identified as: complex events and querying.

With the development of a DSL it is important that life cycle support is provided. Life cycle support refers to the ability to allow users to continuously update the DSL and have those changes reflected in their workspace. Full life cycle support means the support of round tripping and it also means that an approach that compiles away rules will not be used.

Another goal of this thesis is to provide a proof of concept in the form of a working prototype. This prototype will provide insight into the feasibility of the approach taken. The prototype will be later evaluated through code and design metrics to provide validation. The validation will be used to conclude this thesis.

### 1.3.2 Scope

In order for this thesis to not deviate too far from the intent of the research we must define a level of scope. In this section we outline the scope.

We have chosen to manually filter events for this research due to an unexpectedly large volume of primitive events. This would likely require additional functionality to handle. This functionality could possibly be implemented with no more difficulty than any other aspect of the application. The functionality that would be required would be wild card operators for complex event definitions. This would likely require additional unit testing and a large quantity of work to get temporal behaviour acting correctly.

We have chosen to limit the scope of the implementation of the proof of concept application to desktop applications written in C# using the .NET 3.0 libraries. This has been done to provide a realistic objective – in terms of time - to achieve during the course of this research.

We will not be placing importance on Rule Hierarchies (if rules conflict, which ones takes dominance). An example of a situation where two rules would be affected by rule hierarchy:

Rule1: “If button1 is clicked enable label1”

Rule2: “If button1 clicked disable label1”

Rule one and two demonstrate that order of execution can greatly change the outcome of a set of rules. We will assume that verification mechanisms (software and user practice) can be used to avoid these kinds of conflicts.

Equally Speed and concurrency optimizations are not deemed to be of great importance besides demonstrating a probability of a reasonable response time to prove the concept.

We will not be implementing a full solution that is feature complete due to the time it would require to make it. We are attempting to make a proof of concept that will provide enough functionality to draw conclusions from.

In this research we do not investigate the idea of trust as we assume that trust is provided by the programmer's implementation of their application. Due to the open source nature of the project, any strong name keys or product licence keys generated (that could provide elements of trust) would have to be omitted from the repository.

We will also not be considering a non source code way of deployment. It is usual when developing an API to provide the binary forms of libraries and allow users to additionally download source code. As the source is going to be changing rapidly this method of deployment would likely waste a lot of time. To overcome that problem we use a set of build scripts to install binaries which are compiled from the source code provided.

# Chapter 2

# Background

## 2.1 Overview

In the background chapter we start investigating areas of interest, both in research and in industry, that are applicable to either the implementation or that contribute to the theory behind the proof of concept. The first topic of the chapter looks at existing technologies employed in user interface builders. This topic then leads into user interface design patterns. Having established common process for modern graphic user interface programming, the chapter continues by exploring background platforms that could be used for an implementation such as the .NET and web technologies. Investigation of web technologies leads into the next area of rule languages. Rule languages provide background and a platform for storing rules, and are further investigated with a focus on rule visualisation techniques. The chapter then progresses into referencing techniques for locating data of interest. The chapter is finished by an introspection which puts this chapter in context with future chapters.

## 2.2 User Interface Builders

### 2.2.1 Early Approaches

“The concept of direct manipulation interfaces for everyone was envisioned by Alan Kay of Xerox PARC in a 1977 article about the ‘Dynabook’ ” (Myers, 1998)

It is apparent from looking back at the history of user interface builders that user interface builders have been around for over 30 years. The Streamer project which was developed in 1979, was possibly the first object oriented user interface builder. The fact that UI builders have been around this long is particularly interesting as attempts to extend user interface builders to handle user events has never proven to be successful in the long term for various reasons. In this section we will look at many different Integrated Development Environments (IDE’s) which have implemented mechanisms to help provide simplistic GUI driven user interface event handling capabilities.

### 2.2.2 Visual Age

Visual Age for Java was one of many products released as the Visual Age product range. Visual Age for Smalltalk was also a product in the Visual Age line up. The Visual Age products attempted to make a bold progression by allowing the GUI builder visualization to provide more information to the programmer than would be made visible to the application user. This was achieved by visually adding the ability to draw relationships between GUI elements events; exception handlers and their action handlers. The ability to see what user interface elements triggered what code provided visualization advantages. These advantages unfortunately mainly applied to smaller scale applications in that the links of event to action mappings cluttered the GUI builder to the point that it was arguably no better to read than the code itself. IBM attempted to address this concern by adding filters to allow only certain rules to be visible in later versions of Visual Age. Unfortunately the Visual Age family of products was discontinued by IBM; deciding to take a new direction with their IDE's under the name of the Eclipse Project (Rivieres & Wiegand, 2004). The Eclipse Project did not contain a UI builder. This could possibly be due to a trend at the time (early 2000) of a shift towards the web, away from the desktop. Figure 1 shows a screenshot of the Visual Age for Java GUI builder. There is one event rule drawn on a JPanel. The event rule shows a solid line which shows a direction. The direction indicates the item that is being changed (i.e. is having the action performed on it). In the rule given, 'BusinessObject1' is having an action performed on it. The source of the action is an event which will occur on JButton1. The dashed line shown is what happens if an exception occurs.

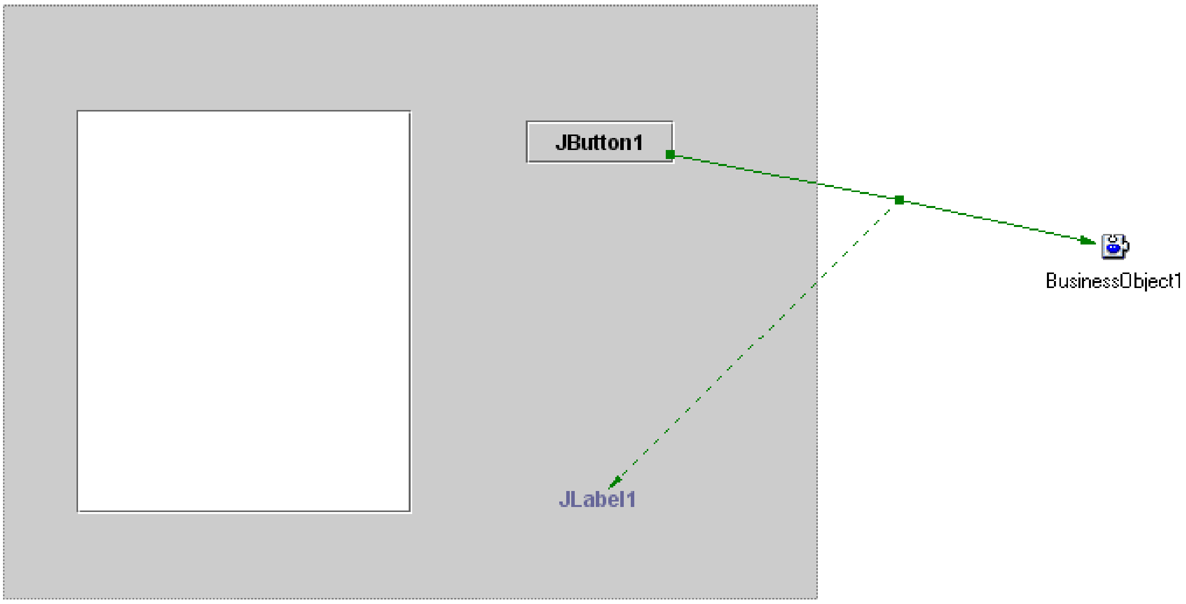


Figure 1 – A Simple Rule in Visual Age for Java

To understand the type of code that Figure 1 could produce, a code listing has been provided below.

```
/**
 * @param arg1 Java.awt.event.ActionEvent
 */
/* WARNING: THIS METHOD WILL BE REGENERATED. */
private void connEtoC2(Java.awt.event.ActionEvent arg1) {
  try {
    // user code begin {1}
    // user code end
    this.SetNameOfBusinessObject(arg1);
    // user code begin {2}
    // user code end
  }
  catch (Java.lang.Throwable Ex) {
    // user code begin {3}
    // user code end
    handleException(Ex);
  }
}
```

In the coding listing previously given, ‘SetNameOfBusinessObject’ would be a method that is performed on BusinessObject1. The method ‘handleException’ would take an input of Exception and would output to JLabel1.  The part not shown in the code listing is the way Java events are ‘wired’ with Visual Age for Java; a code loop is used.  When a single rule is shown in Visual Age such as that shown in Figure 1, there is an apparent clarity of what is happening at an abstract level. The effect of adding many more rules to the Visual Age GUI builder is demonstrated in Figure 2.

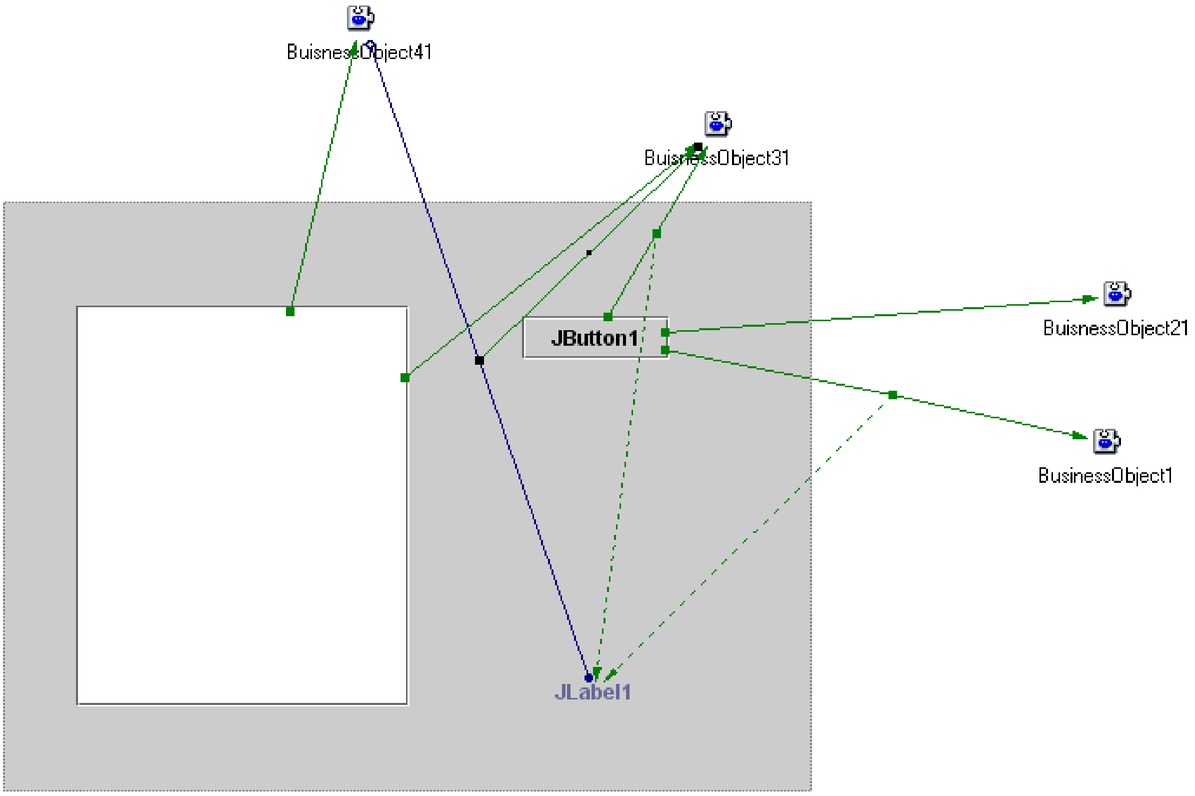


Figure 2 – Many Simple Rules in Visual Age for Java

Figure 2 shows four rules on the same user interface as Figure 1. Unfortunately, the more rules that are added to the GUI builder, the harder it is to use them to gain fast understanding of what is happening. In an enterprise application there could be hundreds of event rules.

### 2.2.3 NetBeans

The NetBeans IDE is released by SUN Microsystems for Java development. NetBeans has a mobility pack that was released in 2007 that has some characteristics that are comparable with the advances made by the Visual Age family. The tools that were released allow a user to have many user interfaces and to link them together like a flow chart. The Links are activated by user interface events. This system is different from the Visual Age family in that it doesn't map action handlers as they are implied from the flow. There also wasn't provision for error handling with the flow entities.

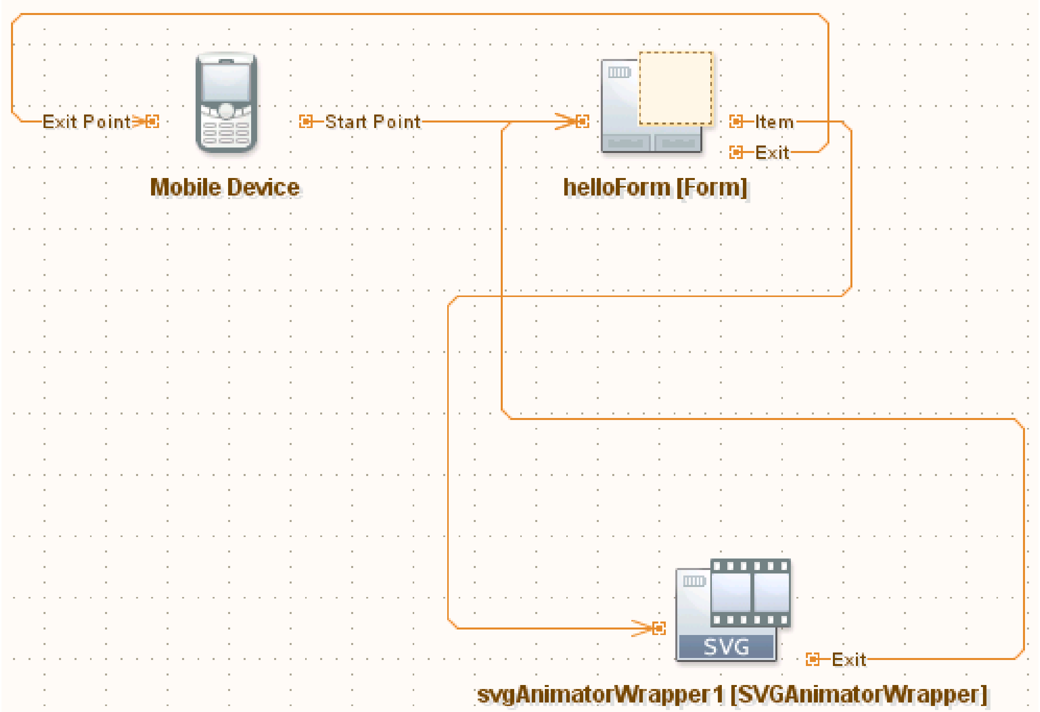


Figure 3 – NetBeans Mobility GUI Designer

### 2.2.4 Delphi

Anders Hejlsberg was the architect behind the creation of Turbo Pascal which later became Delphi. After his creation of the Turbo Pascal compiler he was employed by Borland. Delphi was essentially Pascal which was recreated for the 'Visual Age' of user interface based applications. Delphi also had the ability to do Object Oriented (OO) Programming built into the language. The Delphi IDE contained a user interface builder which had the ability to create action handlers and link them to events via a properties panel. This required the user to select a user interface element and then to click on the properties tab to then edit/create event to action handler mappings. Anders Hejlsberg has submitted the patent for: "Development system with methods for type-safe delegation of object events to event handlers of other objects" (Hejlsberg, 2001) which appears to be used not only in Delphi but in the .NET framework.

### 2.2.5 Visual Studio

Anders Hejlsberg later left Borland and was employed at Microsoft. Since he has been employed by Microsoft he has created the language known as J++ and been the lead architect of C#. It is worth noting that Visual Studio and Delphi are similar in GUI builder design and event handling strategies.

As implied at the beginning of this section Visual Studio is a product developed by Microsoft. Microsoft while mainly known for its market dominance with Windows and Office was actually set up initially as a company to make programming resources. Basic was one of Microsoft first creations. Basic is still alive today however it has been remoulded to satisfy the modern market as Visual Basic and more recently Visual Basic.NET. Visual Basic is considered a third generation of basic due to the introduction of Event driven programming and Object Oriented support. With the creation of .NET a new language was created called C-Sharp (also known as C#).

## 2.3 User Interface Design Patterns

In this project it is important to consider how this research fits into a Software Project. In Software Development there are numerous ways of approaching a solution. Separation through design is a prime goal for software engineers as object oriented design has promoted the use of designs with low coupling and high cohesion. Low coupling helps developers reuse components and minimize risk of cascading failure. Loose coupling also allows for better automated testing of components.

### 2.3.1 Model View Controller (MVC)

Since the use of Smalltalk-80, a design pattern called Model-View-Controller (Goldberg & Robson) has been a popular way to separate design concerns. The Model-View-Controller (MVC) design pattern is shown in Figure 4:

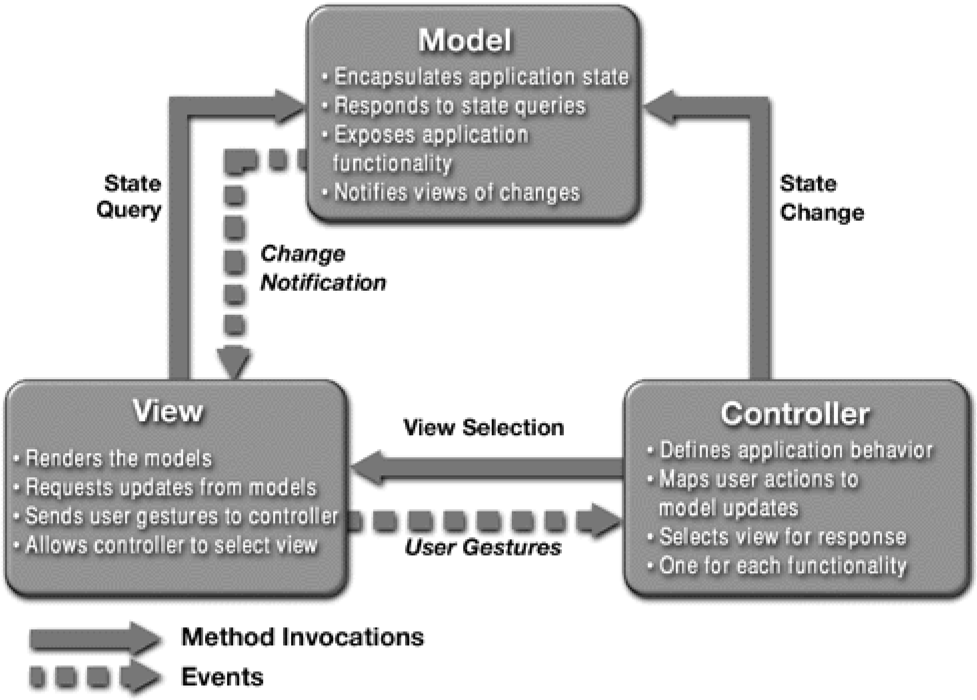

Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
State Change
Change Notification
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

Figure 4 - Model View Controller taken from (Sun Microsystems)

As Smalltalk-80 and the Model View Controller concept were released around 1984 (Goldberg & Robson) there have been improvements on the MVC design pattern since its conception. In the following parts of this section we look at some commonly adopted variants on MVC.

### 2.3.2 Model View Presenter (MVP)

Not to be confused by Microsoft's Most Valuable Professional awards, MVP stands for Model View Presenter. This design variant appears to be most widely adopted by the Microsoft Community but was originally designed by Taligent (Potel, 1996). Unlike the MVC design, the MVP allows for the "Distinguishing the Presenter from Commands and Interactors [which] provides the benefit of reusable logic" (Potel, 1996). The main idea behind MVP over MVC is that it allows widgets to handle User Gestures which would have otherwise been in the controller of the MVC pattern. Allowing widgets to control user interaction definitely makes more sense due to the event model that .NET has adopted where a widget owns the events, rather than the C++ approach of an application wide event loop which would be better suited to MVC.

The MVP design has been extended to allow the possibility of tiers in the applications design. The development community often refer to tiered designs as n-tier designs. Tiered designs would not be as easy to implement if GUI widgets weren't responsible for their own events. Many people who are familiar with MVC often overlook the intention of renaming the presenter instead of keeping the name of controller. With Model-View-Controller, the model and view were clear as to what they were responsible for, leaving the controller doing whatever was left over (e.g. presentation, commands and iteractors). As features such as the ability to undo user actions have become prevalent, it makes sense that the roles in the controller are better defined due to an increase in controlling code. In a modern MVP approach that is tier based, each tier only references one tier, which is the one below it. This enables lower coupling and better understanding of the boundaries of each module. The reason lower coupling is considered a good thing is due to the argument that "Coupling and cohesion significantly influence maintainability, understand-ability, and modifiability, and thus serve as a guide to the development of quality systems" (Eder, Kappel, & Schre, 1992).

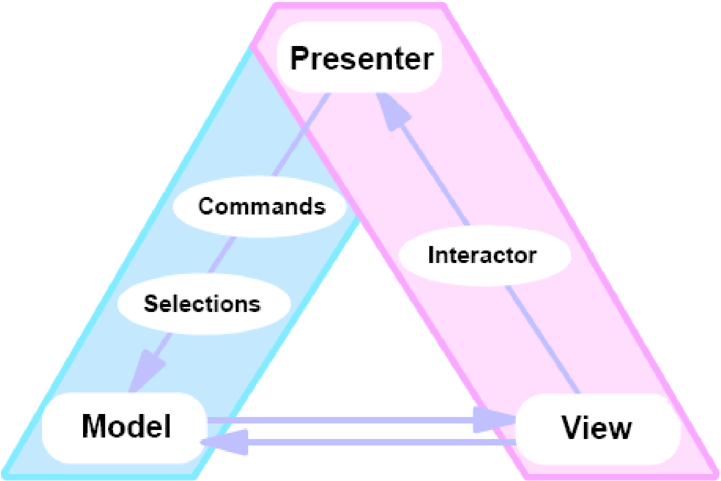


Figure 5 – Model View Presenter (Potel, 1996)

It is worth mentioning that there are variations on Model View Presenter to change coupling directions. Those concepts are still tiered Model View Controller approach's and tend to be chosen based on preference by a developer or a preference caused by requirements of a project. Martin Fowler has provided two variations to MVP which are called 'Passive View' (Fowler, Passive View Design Pattern, 2006) and 'Supervising controller'. He also provides reasoning as to the advantages and disadvantages of the particular approaches.

### 2.3.3 MVC-2

This is used mostly by the Java community for website designs. It comprises the original MVC design with the addition of the Gang of Four Command Pattern (Data & Object Factory, 2001). MVC Model 2 takes the distributed nature of the web into account in its design. The problem with the distributed nature of the web is that the model can no longer update the view. This is demonstrated in Figure 6.

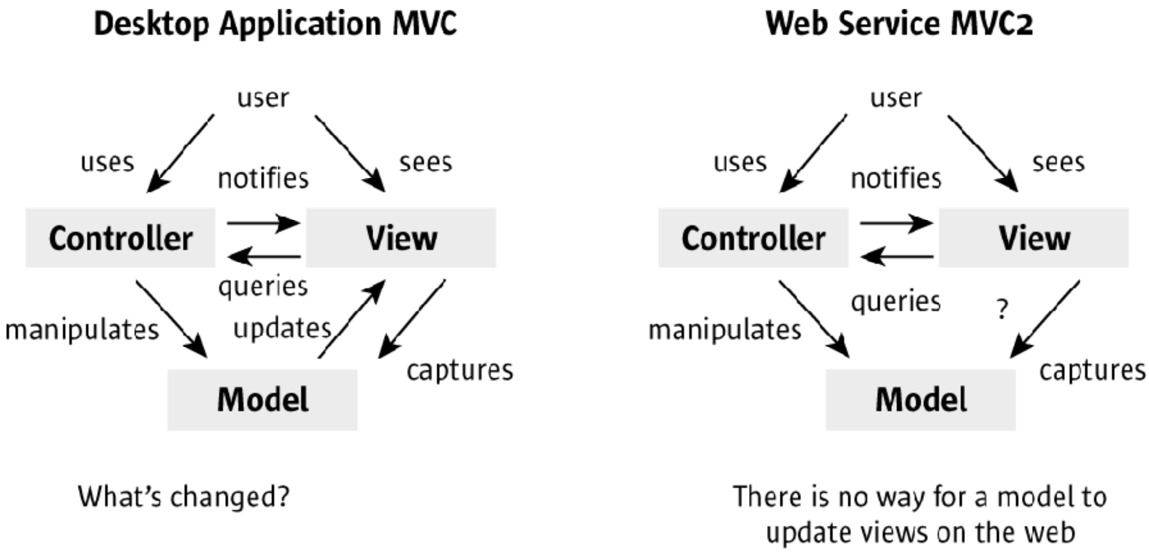


Figure 6 - MVC & MVC2 comparison provided by (Sun Microsystems)

Sun Microsystems claim that the MVC Model 2 redesigns the MVC architecture to be a presenter like approach. This sounds similar to MVP's approach but both have used different models to approach the situation. This may possibly be due to the differences in language features that make certain approaches easier. The web languages such as JSP (Java) and ASP (.NET) are conceptually optimized in different ways; both having their advantages. Having used both, we found the approaches each language offered as best practice differed a lot, however both languages were able to generally match the abilities of the other. This could be a reason for the difference in approach towards a similar goal between the Java and .NET communities. Even with the division of preference between communities neither approach is limited to a particular language.

It was previously noted that the distributed nature of the web changes the MVC model so that MVC-2 does not update the view. Previously it was possible via property change events in a desktop application with MVC.

### 2.3.4 Design Commonalities

The commonality between the shown designs is the use of a Model, View and domain logic (Controller or Presenter) layers. We are particularly interested in the View in that it contains the user interfaces that we are focused on making a GUI builder for; however the View must be connected to the Controller by some form of reference so that it can communicate with the rest of the application. Later in this project we look into techniques that are applied to the Model layer to see if they can provide insight into techniques we can use on the View layer. Another point of commonality worth mentioning is that MVP and MVC both use an observer pattern (Data & Object Factory, 2001) approach for communication between modules.

In the next section we look at rules. Application of rules applies to all aspects of application development and MVC design.

## 2.4 Document Object Model

The Document Object Model (DOM) is a tree like hierarchical model that has mainly been used for Hyper Text Mark up Language (HTML) element referencing. The document object model is of interest to this research as it is provides a standardized design for a user interface. Standardized design allows for the adoption of existing knowledge in a published researched area (W3C, 2004).The DOM also has many similarities to .NET.. C# uses the idea of a tree hierarchy for components within a form (Window), like the DOM. Unfortunately it didn't natively support the advanced features provided by the DOM such as event bubbling. While writing this thesis .NET 3.0 was released which provided additional DOM functionality such as event bubbling. The fact that .NET 3.0 is providing DOM like functionality for desktop programming could be a sign of convergence between the two areas of the desktop and the web. An advantage of using a Document Object Model is demonstrated in **Error! Reference source not found.**, where functional processing / traversing of the tree is shown.

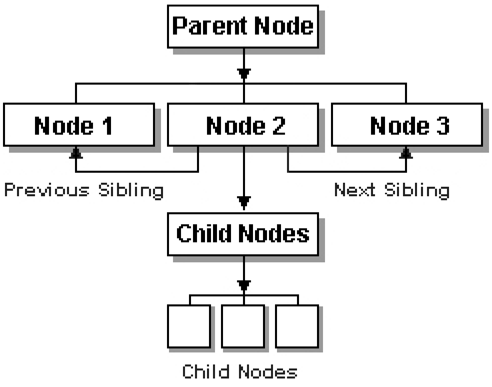


Figure 7 - Document Object Model instance showing Functional Navigation (Microsoft)

If in the previous diagram the 'Parent Node' was the document node, then the following code could be used to describe setting the style property on the child node at index 1.

```
document.getElementsByTagName('Node2').item(1).setAttribute("style","position:relative")
```

The child node could be a table HTML element. A table element can have children which would be table data.

The idea of the Document Object Model (DOM) has possibly become prevalent due to its heavy usage in Dynamic HTML languages such as ECMA script. The adoption of a Document Object Model outside of ECMA script isn't as large as it could be.

## 2.5 ECMAScript

ECMA script was created to standardize the Dynamic HTML (DHTML) languages. The Netscape browser used a language called JavaScript and Microsoft's Internet Explorer used a language called Jscript for providing Dynamic HTML abilities. The problems with competing standards of DHTML languages caused web developers to either facilitate both or risk losing clients of the browser(s) not supported. Both of those languages influenced the ECMA script standard (ECMA General Assembly, 1996). While ECMA script has mostly been adopted client side, it was intended that JavaScript would not only provide additional power to browsers to manipulate HTML but also to perform server computation. ECMA script is an object oriented language that was designed to manipulate objects within a host environment. While ECMA script is similar to Java in syntax, the similarities largely end there as it was intended to only be a scripting language.

## 2.6 .NET Framework

"The .NET Framework is Microsoft's application development platform that enables developers to easily create Windows applications, web applications, and web services using a myriad of different programming languages, and without having to worry about low-level details like memory management and processor-specific instructions." (Avery, 2007)

The .NET Framework achieves hardware abstraction by using an intermediate language called the Common Intermediate Language (CIL) which is defined in the Common Language Specification (CLS). The CLS is an agreement between language designers and the framework to ensure portability between .NET languages. The CLS is part of an ECMA standard (Microsoft, 2006) . The following diagram shows how the common intermediate language fits into the compilation process. In the diagram the CIL is shown as IL module.

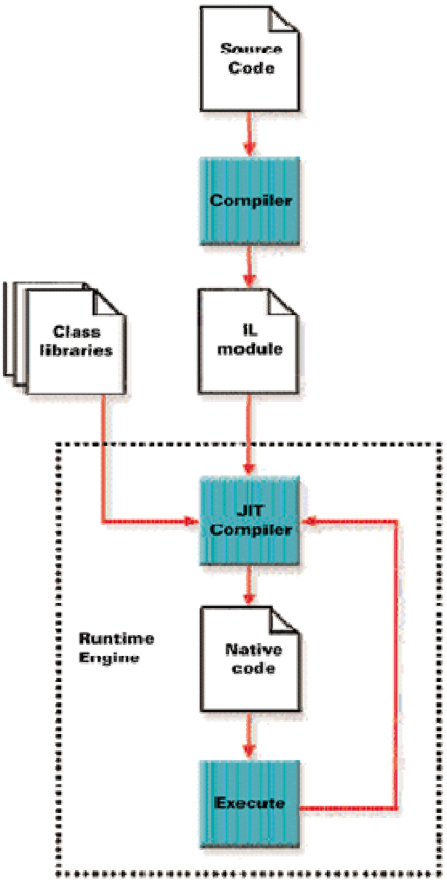


Figure 8 - .NET Framework compilation process

The idea of using an intermediate language is not new, as other languages such as Smalltalk and Java have been using an intermediate language for a long time, however .NET designed their intermediate language to be general for all potential languages that .NET would run. Another advantage that .NET has over Java's intermediate language is that it understands generics. While Java supports generics at a language level, it compiles them away at an intermediate language level. An example could be that a data type using generics made in C# can be understood by Visual Basic.NET as being a generic type. In Java the byte code does not have a concept of generics. This is possibly to keep backward compatibility in the Java Virtual Machine. Microsoft did not maintain backward compatibility between .NET version 1 and version 2 and received criticism for this.

The .NET framework is a pure object oriented language like Smalltalk due to all variable types inheriting from object. In Java this is not the case and primitive types such as integer do not inherit from object. Java is however similar to .NET in that they both offer users benefits that have come to be expected from interpreted languages such as garbage collection, reflection and threading facilities.".NET is Microsoft's platform for XML Web services, the next generation of software that connects our world of information, devices and people in a unified, personalized way." (Rubiolo, Meier, Jezierski, & Mackman)

Due to the nature of W3C Web Services being heavily backed by XML languages (Booth, Haas, & McCabe, 2004), XML was a major design consideration for the framework. In the next section we look into what XML is and why it is helpful in the implementation for this research.

## 2.7 XML

Extensible Mark-up Language (XML) is a W3C recommendation (Bray, Paoli, Sperberg-McQueen, Maler, & Yergeau, 2006) that first surfaced in 1998. XML has become a popular way of storing data due to the ease of understanding and standardization of XML vocabularies. Further on in the thesis we will realise a requirement for storing data. XML is likely to be the underlying concept used, however the exact language (subset) that we use is discussed further in the thesis.

XML appears to be increasingly in industry adoption. This could be due to the increasing number of tools available for the standard. A standard can also provide pre-made solutions and design approaches to known problems. An example of increasing industry adoption is the new way of declaring user interfaces in .NET, which is a form of XML. This XML approach has been suggested for describing user interfaces previously however it has always been a third party option. Now Windows Presentation Foundation (a part of .NET 3.0) has made it the default way for creating C# applications.

XML is likely to play a more central role in this research than just being used in the user interface of an implementation. We believe this to be the case from investigating an IBM paper on "Achieving complex event processing with Active correlation technology" (Biazetti & Gajda) where it was demonstrated as employing XML to store event processing rules.

With the adoption of an XML way of declaring user interfaces, other XML technologies have been employed to provide additional functionality. An example of this extra functionality is the use of XPath as a means of providing references between user interface Objects.

### 2.7.1 XPath

XPath like XML is a W3C standard. The purpose of XPath is to provide a way of referencing XML items. As of the time of writing this thesis there are two versions of XPath. The versions are version 1 and version 2. XPath 1.0 was used primarily as a way of addressing specific elements. XPath 1.0 could be used in combination with XQuery and XSLT to find and modify document elements. XPath 1.0 also only had four supported expression types: node-set, boolean, number and string. XPath 2.0 has taken a different approach and referencing XML elements by adding over 19 new simple types by adding support for all XML Schema primitive types. XPath 2.0 has also added a large amount of XQuery functionality to its repitoire. XPath 2 also provides order to the results it returns.

If we refer back to Figure 7 a comparative XPath 1.0 example could be:

```
/Document/Node2/*[1]/@Style
```

That XPath statement would only select the style attribute. It would then be necessary to assign to that selection. The DOM approach allows for the ability to select and apply methods to the results. XPath alone doesn't allow for this as it only selects results and is not able to modify the resulting query.

### 2.7.2 XML Schema

XML Schemas are a W3C recommendation as an Internet Standard (Fallside & Walmsley, 2004). Schemas can be used to verify that a document conforms to a Schema that you have provided. This allows for a clearer idea of what is a valid document and what is not. Observing an XML document most likely will not reveal all of the possibilities of what can be represented. A schema will reveal that information.

If a schema is not publicly available then it is more difficult to see what is possible. To obtain compliance to an XML syntax without having a schema you would need to see enough different combinations of valid XML documents to extract the information you require.

## 2.8 Semantic Web

The semantic web (W3C, 2008) is an envisioned extension to the world wide web that provides semantic reasoning of web content. While web content is not in the scope of this thesis, the concepts are worth investigating due to the parallelisms of what we are trying to achieve. Both this research and the Semantic web utilize similar thinking over the reasoning of data. The data provided by event handling, applying reasoning by rules and then applying that information to a document object model are similar in concept.

The semantic web can be visualized as thinking of web content as items in a distributed database that are not only able to reference by a Unique Resource Index (URI) but by information describing the data; referred to as meta-data. This is a philosophy that is working towards a new level of understanding by machines of the data they process. Concepts that are illustrated in the semantic web initiative can be seen in websites such as Digg (Digg Inc.), Wikipedia (Wikimedia) and Delicious (Del.icio.us). The semantic web shares similar concepts of providing reasoning over data that are later utilized in this project. The semantic web vision is shown in Figure 9.

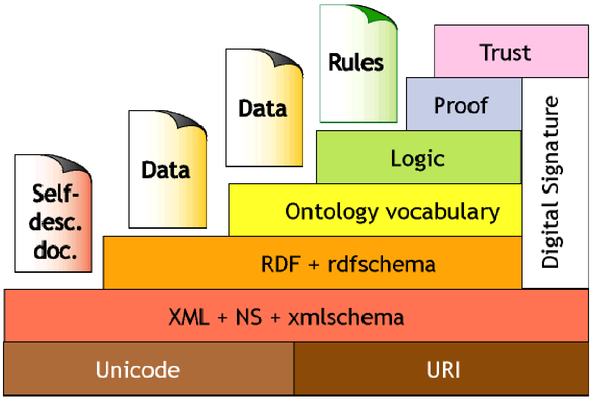


Figure 9 - Semantic Web Architecture (Berners-Lee, 2000)

In architecture we can see a technology stack. In this project the end result conforms with every layer of the semantic web architecture. However due to certain conveniences of working on a local machine and a framework such as the .NET platform some tiers such as trust and digital signature were either provided by the framework or assumed that a user of our system would implement them. Also some tiers were conceptually the same but utilized different technologies due to the use of a compiled language rather than a web environment. This will be investigated in chapters 3 and 4.

In this research we do not investigate the idea of trust but assume that the trust features provided by the .NET framework such as Strong Name Keys and Namespacing could be utilized if required. Strong Name Keys are a way of digitally signing an assembly. For readers familiar with Java, Namespacing is similar to Java Packages. Namespacing in .NET is also similar to XML namespacing. For more information on XML namespacing, see the formal specification for XML (Bray, Paoli, Sperberg-McQueen, Maler, & Yergeau, 2006)

## 2.9 Understanding Windows Presentation Foundation

With the release of the .NET Framework version 3.0, the framework was divided into four foundations. The four foundations as of version 3.0 are: Windows Workflow Foundation (WWF); Windows Communication Foundation (WCF); Windows Presentation Foundation (WPF) and Windows Card Space (which deals with User Authentication). By dividing the .NET Framework into four core areas, it has allowed for better communication on forums regarding problem areas, and it allows the developers of the framework to make libraries easier to find by having dividing categories that functionality can belong to. In this project we are mainly interested in the Windows Presentation Foundation as it is the foundation that specializes in User Interface development.

WPF has introduced a different mindset to .NET for developing user interfaces. Before .NET 3.0, .NET 2.0 offered a library called WinForms. Winforms appeared to be just a .NET version of the Win32 COM based forms that Microsoft traditionally used with the Visual Studio family of languages such as Visual Basic and C++. Winforms were entirely code based and programmed in a heavily imperative way due to their heavy use of code. Winforms were replaced with an application mark up language called extensible application mark up language (XAML). XAML offered an XML definition of user interfaces in a declarative way. With the addition of XAML to .NET came the advantages of increased flexibility in how events were handled due to the realisation of a document object model comparable to that of ECMAScript (ECMA General Assembly, 1996).

### 2.9.1 XAML

XAML stands for extensible application mark up language. XAML is an XML syntax which has been designed for software applications. A project called myXAML is an open source project that allowed developers to express user interfaces in XAML in .NET version 2.0. Microsoft introduced XAML (Microsoft, 2007) to the .NET Framework with the Windows Presentation Foundation (WPF) in version 3.0. WPF did however, provides new event handling techniques that make it possible to handle events in ways that were not previously possible with the .NET framework. Using an XML means to represent a user interface also has advantages in allowing user interfaces to be understood by people not of a coding background but who might provide design services, such as a web developer familiar with html. XML user interfaces also provide benefits of research in the field of XML such as serialization.

### 2.9.2 Event Routing Strategies and Traditional Event Systems.

With .NET 2.0, events were limited to a direct routing strategy which will be explained later in this section. Event handling concepts from ECMAScript were added to .NET with the introduction of XAML(.NET 3.0). Conceptually the new ways of dealing with events is to use one of three routing strategies: Direct, Bubbling and Tunnelling.

### 2.9.3 Direct

This is the default way that the .NET framework has always supported. If an instance of a button called button1 is clicked, then you must have subscribed before the click to the button1 click event in order to handle it. This technique of direct handling is a one to many mapping in that there is one event that many handlers can subscribe to. Events cannot be dealt with in a lazy way like event bubbling. The following code snippet demonstrates a subscription to an event. There can be none, one or many of these subscription statements. Figure 10 demonstrates a model of how a framework like .NET handles events.

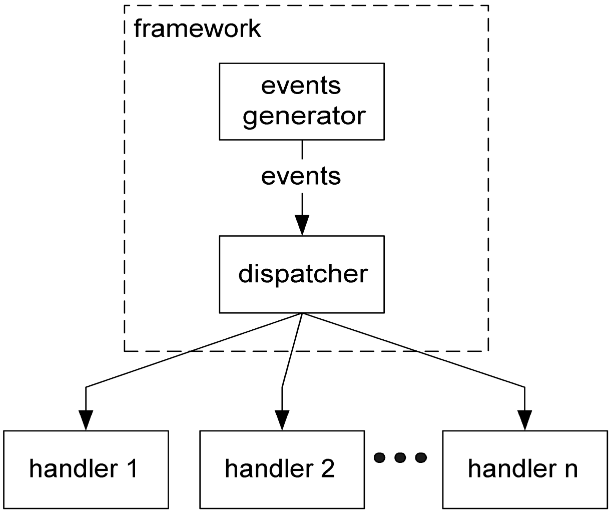


Figure 10 - Framework Event Handling (Ferg, 2006)

As Figure 10 may leave many readers with the question as to how code would look for a directly handled event, code has been provided in the following text.

Code:

To subscribe to the event of button1.Click the following line of code is employed. Note that the 'myClickHandler' is a delegate (a function pointer) to a function that acts as the action handler. Some readers with a functional programming background may wonder if delegates are comparable with Closures. While Delegates themselves are not closures, Anonymous Delegates are closures. The functionality of anonymous delegates is demonstrated by (Walnes, 2006). The main reason why a delegate is not a closure is because "closures can refer to variables visible at the time they were defined" (Fowler, Closure, 2004). A delegate is however like an Anonymous class in Java in that they both achieve Closure like functionality.

```
Button1.Click += new EventHandler(myClickHandler);
```

The following is the method that is the action handler for the event. The parameters of the method are determined by the definition of 'EventHandler', see (Microsoft, 2003). It is possible to customize the event handler by sub classing it.

```
myClickHandler(object sender, EventArgs e)

{

//our action handler
}
```

Event Bubbling would be difficult to do with this technique as you would have to throw the event to the parent object. If using a third party UI component you would have to then extend and override the functionality of their component to get this behaviour. Fortunately in .NET version 3 event bubbling was incorporated allowing for bubbling of events unless explicitly handled. This made the need for extending a third party component for the purpose of obtaining an event no longer necessary.

### 2.9.4 Bubbling & Tunnelling

Previously in this chapter we looked at ECMA script and the importance of the Document Object Model. Both of those standards were noted as having contributed to event bubbling. There are a lot of parallelisms in this research with the ideas ECMA script has used for many years. Event bubbling is not a concept that can be applied without some sort of an object model that is at least similar to a tree like the DOM. The reason for this is that event bubbling requires a parent element to bubble an unhandled event towards. Bubbling is an unhandled event rising up a tree, where as tunnelling is an event occurring on a parent object and travelling down to its children. This is shown in Figure 11.

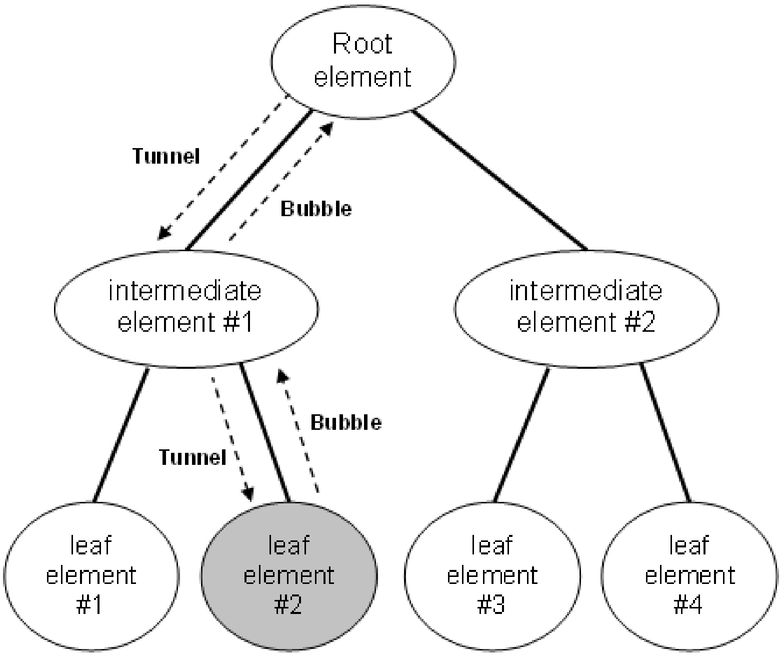


Figure 11 - Event Bubbling and Tunnelling (Microsoft, 2004)

## 2.10 Rule Based Software Development

“Three basic types of business rules have been identified in the literature: integrity constraints (also called ‘constraint rules’ or ‘integrity rules’), derivation rules, and reaction rules (also called ‘stimulus response rules’, ‘action rules’, ‘event-action rules’, or ‘automation rules’).” (Taveter & Wagner, 2001)

In this thesis we have a specific interest in reaction rules as it is likely we will use them for ‘wiring’ events to actions. We later discover a need for production rules to facilitate ‘rule chaining' for evaluation of data. Production rules are a form of derivation rule. Within this thesis we will not investigate integrity constraints as they are considered outside of the scope. They could be worth investigating in future research however.

There is also a "fourth type, deontic assignments, [that] has only been partially identified (in the proposal of considering 'authorizations' as business rules)" (Taveter & Wagner, 2001). That fourth type of rule is probably also outside the scope of this project as we are not going to be dealing with user authentication systems, however it is worth keeping in mind for either future research or extending this research.

### 2.10.1 Rule Summary

Currently in software engineering the first thing that comes to mind when thinking about rules are rule engines. Rule engines are libraries that are used to process high level often domain specific logic. The advantage of using rule engines is that customers can update and easily understand high level rules without touching the rest of their program. Rule engines fit in to the controller layer of the model view controller patterns. Rule engines are not the only use for rules in application development. Object Relational Mapping(ORM) is also common in application development and fits into the model layer. Generally speaking there is little or nothing offered in terms of rule based processing over the view layer as a reusable library.

## 2.11 Rule Languages

When looking for rule languages there are many to choose from. The fact that there are many to choose requires us to provide scope towards what we want our language to express. Earlier it was stated that we have a particular interest in reaction rules due to the nature of our event to action mapping focus. We did also state a future requirement for production rules which is also taken into account when researching options for a rule language.

### 2.11.1 RuleML

"RuleML covers the entire rule spectrum, from derivation rules to transformation rules to reaction rules. RuleML can thus specify queries and inferences in Web ontologies, mappings between Web ontologies, and dynamic Web behaviours of workflows, services, and agents." (Boley & Tabet).

RuleML states on their website that they are not focusing on academic research prototypes and are working more towards interoperability with existing standards. For the purpose of this research their language was suitable in that it was quite abstract so it could potentially represent what was required. However it also suffered from being too general to use easily. An example RuleML datalog fact taken from a tutorial on the RuleML website (Boley, Grosof, & Tabet, 2005) is provided in the following listing:

```
<Atom>

 <Rel>spending</Rel>

 <Ind>Peter Miller</Ind>

 <Ind>min 5000 euro</Ind>

 <Ind>previous year</Ind>

</Atom>
```

Figure 12 shows how the translation of the XML given previously is interpreted.

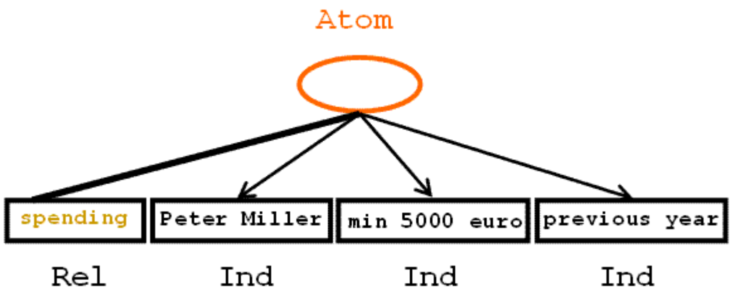


Figure 12 – An Atom structure in RuleML (Boley, Grosof, & Tabet, 2005)

The meaning of the rule is not inherently obvious from reading the XML. It could take a lot of work to write code in an imperative language such as C# that can make sense of this syntax. To demonstrate this difficulty we will look at the fact that ‘Peter Miller’ and ‘previous year’ are both under the ‘Ind’ tag but have quite different meanings. This would provide unnecessary complication to processing for a type safe language like C#. The RuleML language seems to be closely based on the principles utilized by Prolog (Deransart, Ed-Dbali, & Cervoni, 1996) of establishing quite general facts to deduce new facts. This is often referred to as rule inferencing.

This language claims to offer the ability to express rules for forward and backward chaining. There are many extensions to RuleML which facilitate specific rule types such as reaction rules.

RuleML could be a candidate for use for storing rules, however it would not be the easiest to use as it is an extremely general/abstract syntax. As our problem domain is likely to be well defined to framework event mapping and processing, unnecessary domain abstraction could be problematic for an implementation. This is possibly a reason why Object Oriented RuleML was developed; to provide more type ownership. In the next sections we look to see if there could be another syntax better suited to our problem area.

### 2.11.1 SWRL

Semantic Web Rule Language (SWRL) “is a member submission, offered by the National Research Council of Canada, Network Inference and Stanford University” (W3C, 2004) made to the W3C. SWRL is unique as it joins rule languages with OWL. This is unique as the power of OWL is not compromised in the process. This is because SWRL attempts to join languages rather than take a subset of them. fore

An example of a rule that demonstrates SWRL’s inferencing ability is given below:

hasParent(?x1,?x2) ∧ hasBrother(?x2,?x3) ⇒ hasUncle(?x1,?x3)

It is important to note that a rule in SWRL is serialized in XML, so the code for such a rule is said to be “verbose and not particularly easy to read” (W3C, 2004). It is interesting to observe that the way the rule previously given is almost identical to how the rule would be represented in SWI Prolog. The downside to a Prolog like rule language could be that we are quite focused toward wanting our rules to react to Framework events. SWRL can however be used to write custom rules for ontologies, that complement built in rules by allowing sub-classing and symmetry of relationships. This means that SWRL could be used to extend a chosen syntax.

A language that can be used to specify relationships is OWL. OWL could be considered for extending an existing XML syntax. Web Ontology Language (OWL) is a W3C recommendation (OWL Web Ontology Language, 2004) that provides information for processing data, rather than just leaving information for display purposes. OWL extends the vocabulary of RDF. OWL offers relationship information that allows descriptions of ordinarily such as a ‘one to many’ relationship. OWL can be used in conjunction with not only SWRL but other rule languages also. We look at R2ML next to see if it is closer aligned to our problem domain. SWRL like RuleML can almost certainly provide this functionality but as it seems to be aimed at a use case closer to where a developer might employ Prolog, there could be another syntax more appropriate to our domain.

### 2.11.2 R2ML

R2ML stands for Rewerse Rule Mark-up Language. REWERSE is a group that is researching the area of Semantic-based knowledge systems. Their goals include attempting to propose “languages to the level of open pre-standards amenable to submissions to standardisation bodies such as the W3C” (Rewerse, 2004).

As mentioned in the previous section R2ML has support for integrity rules, derivation rules, production rules and reaction rules. When considering R2ML for this project the aspect that we are interested in is primarily reaction rules. While it is possible we could require other aspects of a rule language the reaction rules are the part that is going to enable the representation of ECA rules which were mentioned in the previous chapter. R2ML reaction rules syntactically are written in XML as ECAP rules making them conform to abstract concepts that are preferable for ease of understanding. A Major goal of R2ML was to merge functional languages like OCL with relational languages such as RDF, OWL, SWRL. This merging of these languages is mentioned in the following statement: "R2ML is *comprehensive* in the sense that it integrates the *Object Constraint Language (OCL)...*, the *Semantic Web Rule Language (SWRL)...*, the *Rule Markup Language (RuleML)* a proposal based on Datalog/Prolog" (Giurca & Wagner)

## 2.12 Reaction Rules

"Reaction rules are concerned with the invocation of actions in response to events. They state the conditions under which actions must be taken; this includes triggering event conditions, pre-conditions, and post-conditions (effects)." (Taveter & Wagner, 2001)

In this thesis, R2ML is later used to describe reaction rules. The form that R2ML expresses reaction rules is as ECA-P(Event, Condition, Action, Post-condition) rules. ECA rules are not a new concept in rule based software development; "they have been studied quite extensively in the area of Active Database Management Systems" (Norman & Paton, 1998). ECA rules were used mainly in databases as they provided a means to check constraints (sometimes conditions dictated by other Rules) on the data being used. Reaction rules used in database management systems are known as 'database triggers'. Research in the area of databases has suggested the use of ECAP rules and complex events. ECAP rules are ECA rules with a post condition.

ECAP is not the only variation on the original concept of ECA rules, it has been suggested that Event-Condition-$\text{Action}_1$-$\text{Action}_2$ (ECAA) rules could be used (Knolmayer, Endl, & Pfahrer, 2000). ECAA rules provide the convenience of using the condition like an if-else statement however "any ECAA rule can generally be rewritten as two ECA rules, one with the original condition and one with the negated condition" (Berstel, Bonnard, Bry, Eckert, & Patranjan, 2007)

Another variant of ECA rules is $EC^nA^n$. After the rule is triggered the rule evaluates each condition until one is true. If for example the third condition was true then the third and only the third action would be executed. If the fourth condition was true then it usually would not be processed; see (Berstel, Bonnard, Bry, Eckert, & Patranjan, 2007). It is worth noting that with all of the variations to ECA that have been mentioned previously in this section all can be translated to plain ECA rules.

## 2.13 Event Algebra

Reaction rules can be used to respond to an event if it occurs, however what if we are expecting a compound event made up of many events? To answer this question . Event algebra allows us to compound many events in a relationship that forms a single event that represents that group of events. Event algebra and composite events have been previously concentrated in the area of relation databases. An example of such research is: “On the Semantics of Complex Events in Active Database Management Systems” (Zimmer & Unland, 1999).

### 2.13.1 Event categories

From investigation, we have identified three event types in the domain of databases:

- Database event
- Temporal event
- External event

Identifying these events types is of interest when considering the context in which each of the listed events would be fired. Database events are events that are raised by actions performed on a database. Temporal events are events that are time based. External events are events that occur from the environment outside of the database. The relevance of considering context when working with events is that context must be retained across transformations of events into composite events. The database arena of research also shows us that different event types can typically have different weights of importance on the context in which the event was triggered (Zimmer & Unland, 1999). This is likely to be similar when considering the domain we are working with - application development.

### 2.13.2 Atomic event types

Atomic events are primitive events. All events are either primitive or complex and the two definitions are mutually exclusive. In this research we consider framework provided events as primitive event. Events that are provided to programmers in both C# and Java are a mixture of primitive and complex events. ‘Double Click’ is an example of a complex event as it is comprised of two ‘Click’ events.

### 2.13.3 Complex Events

Complex events are events that are comprised of other events. Shown below is an example of a complex event: Event3, that is composed of events 2 and 1. Events 2 and 1 might or might not be complex events; it doesn't change Event3 from being complex.

e.g. Event3 = Event2; Event1

In the following text we look at the notation behind the complex events.

### 2.13.4 Event Algebra Notation

The following text demonstrates how an event is represented with event algebra.

| | |
|---|---|
| $e^{n}_{i}$ | nth event e where n is the number of times the event has previously occurred<br>i = instance number |

We also have the following operators available.

| | | |
|---|---|---|
| ; | means sequence | |
| == | means simultaneous or parallel | |
| ^ | means conjunction | (AND) |
| v | means disjunction | (OR) |
| ¬ | means negation | (NOT) |

An example complex event is provided following:

$E_4$ := ^($E_1$ , $E_2$ , $E_3$ ) /* $\text{Event}_4$ is comprised of $\text{Event}_1$ and $\text{Event}_2$ and $\text{Event}_3$ */

### 2.13.5 Event Correlation Service

When considering 'event algebra', investigation of existing research that applies 'event algebra' to the application development domain provided strong leads to relevant information. Research in the area of event correlation services was of particular relevance to this thesis.

The first research that was investigated was "A Configurable Event Correlation Architecture for Adaptive J2EE Applications" (Liu, Gorton, & Le, 2007). This research investigated events between distributed applications and applied an event correlation architecture to those events. Although distributed applications falls outside of the scope of this research, the research does investigate the use of an event algebra for use with source code events and the different correlation options that can be applied to event processing. Futhur investigation led to a paper (Motakis & Zaniolo, 1995) that proposes the use of an Event Pattern Language (EPL). The event pattern language that they propose uses event algebra to describe patterns of relationships between events. Although this second paper proposing an event pattern language doesn't address event correlation services it does provide context to the first paper as it address use cases that are applicable to the various suggested correlations in the first paper. The correlation options that were highlighted were:

- Threshold-based correlation
- Filter-based correlation
- Sequence-based correlation
- Time-based correlation

Of the fore-mentioned correlation areas sequence-based correlation was of particular relevance. Sequence based correlation allows for the concept of complex events. Threshold-based correlation is likely outside of the scope for this thesis. Filter and Time based correlation will only be implemented if they are required to achieve or help sequence based correlation.

The paper also provides information as to how the authors implemented the internal structure of their event correlation service. The demonstration that they provided pointed event sources toward a singleton event buffer. Events were then filtered by use of regular expression. After the filtering process events were transformed using event algebra. The last step was to execute the actions mapped to the transformed events.

## 2.14 Visually Representing Rules

In this project, a number of ways to visually represent our rules are investigated. The assumption we made, however not limited to, was that rules might be best drawn over a user interface, much like the Visual Age products mentioned earlier in the chapter. In future chapters this assumption is investigated to assert its validity.

The following technologies were investigated as possible ways to visually represent rules: UML, URML, JBoss Rules, BPMN, Visual Age for Java. The Visual Age of products were considered, however they are less standardized than the other candidates and the language appeared quite domain specific.

### 2.14.1 UML

Unified Modelling Language (UML) is a widely used modelling language aimed to provide a means of conveying an idea or design through modelling so that many people can work under the same plan. UML was made and is maintained by the Object Management Group (OMG). UML is not specific to the domain of software development in that it is also used for business modelling. For this thesis UML 2.0 was investigated. Information about UML and the various modelling options that were available were located by looking at the UML 2 superstructure (Object Managment Group, 2004). Table 1 shows the candidates that were identified from the superstructure document.

**Table 1 - UML Candidates for a Visual Rule Language**

| Potential Diagrams | Candidacy | Application |
|---|---|---|
| Activity Diagrams | Medium | This could visually represent rules. The focus is on actions, not the messages passed between them. This will require the addition of a constraint language to generate code. |
| Communication Diagram (formerly Collaboration Diagram in UML 1.x) | Low | This could visually represent rules, however is more abstract than the activity diagram. It could be used to extend an activity diagram for debugging purposes where messages are of higher interest. |
| Sequence Diagrams | Low | Could be used for a visual extension for debugging. |
| State Machines | Medium | This type of diagram is similar to a flow chart. It could be used to represent rules but it could require a larger diagram than we would want. State machines are very similar to activity diagrams. |
| Use Cases | Low | Could be used for creating a list of rules based on planned user interactions. |

While UML didn't provide exactly what was required, there were features to note such as: The UML communication diagrams message notation which provides a way of displaying data transfer via methods on a class diagram like structure.

Message Notation

```
[sequenceNumber:] methodName(parameters) [: returnValue]
```

(Scott W. Ambler Ambysoft Inc.)

The return value is the type that is returned e.g. string.

### 2.14.2 URML

URML is a UML based rule modelling language, designed by the Rewerse Working Group I1 (Rewerse, 2006).

An example of URML is shown in Figure 13, where the situation of the following statement occurs:

*On customer book request, if the book is available, then approve the order and decrease the quantity of books in stock.*

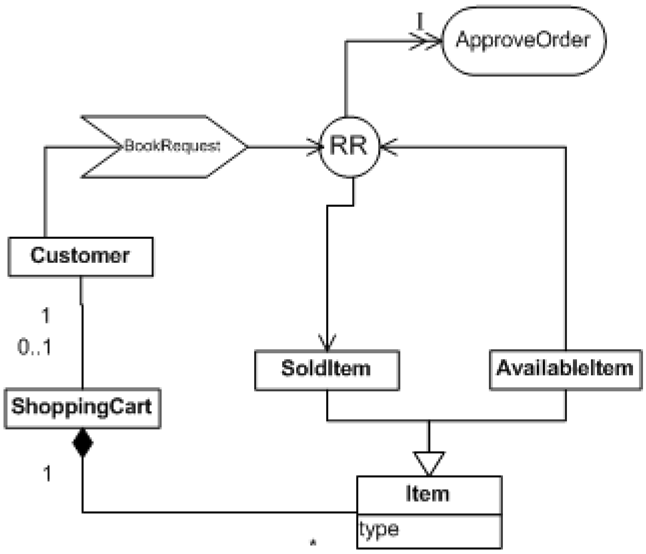


Figure 13 – URML based Reaction Rule example (Rewerse Working Group, 2006)

As URML is proposed by the Rewerse working group, it is particularly relevant to other technologies that they are working on such as R2ML. URML is certainly closer to what this project requires however it is quite an abstract diagramming language. Abstract diagramming is not suitable for an implementation of this research due to a requirement of having visual representation of real objects, rather than just classes.

### 2.14.3 JBoss Rules

The JBoss Rule Engine known as Drools, provides tools to use a visualization that is based on the RETE algorithm (Forgy, 1982). The appeal of this approach is that it is comparable to flow charts. It makes sense to have a similar approach to flow charts for a rule based system as flow charts are a simple form of a rule based system.

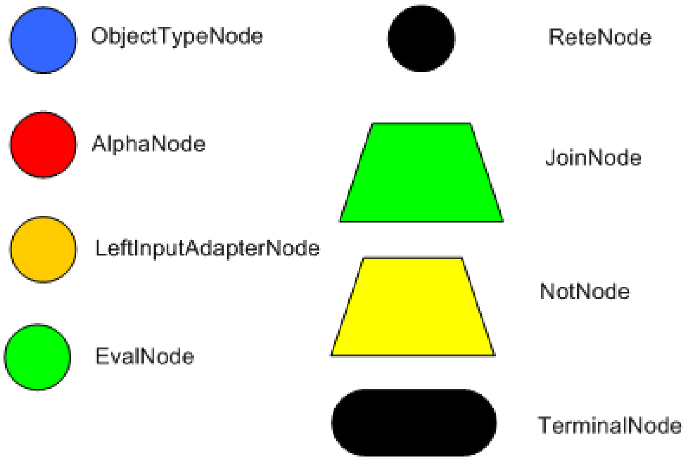


Figure 14 – JBOSS symbols (Proctor, Neale, Lin, & Frandsen)

The goal of these diagrams is to try and filter data with each node in the diagram so that eventually terminal nodes would extract outcomes that are wanted. These diagrams appear to be closely related to finite state machines if not fully conformant.

### 2.14.4 Business Process Modelling Notation

Business Process Modelling Notation (BPMN) is an initiative led by the Object Management Group, which was earlier mentioned for their creation of UML. BPMN was designed to bridge "the gap between the business process design and process implementation" (S. White IBM). BPMN is a flow chart like diagramming approach to data modelling. Unfortunately, it is higher level than UML, making it not low level enough for use with this research. The following diagram demonstrates an example segment of a process to highlight its high-level flowchart-like approach to diagramming.

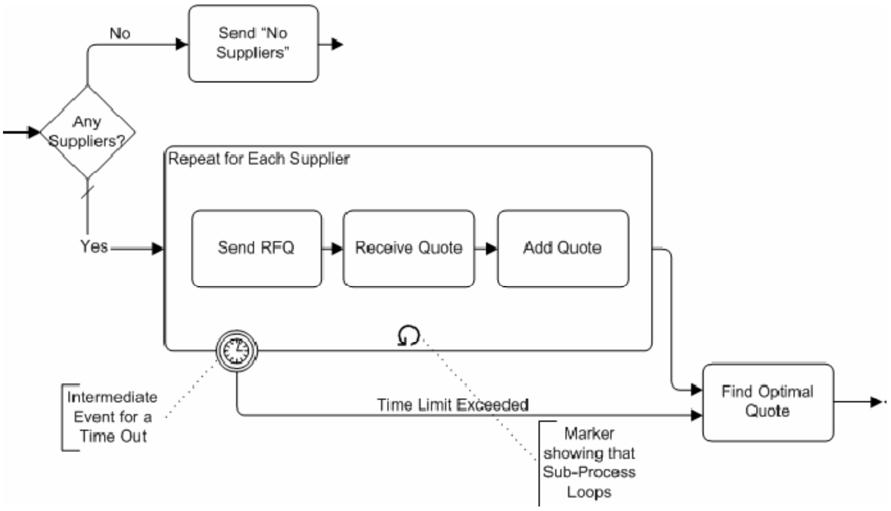


Figure 15 - An example BPMN diagram (S. White IBM)

## 2.15 Reference Techniques

Referencing is relevant to the research we are doing in that once an event occurs, we are going to require a way to perform an action from an ECA rule back on the GUI. We could also require reference to the model or controller tiers to update data.

### 2.15.1 Unique Resource Identifier

In order to perform an action, a rule will require a Unique Resource Identifier (URI Planning Interest Group, W3C/IETF, 2001) to use as a reference to access and most likely manipulate that object. Unique Resource Naming is not a unique problem in that the Internet has been using a unique naming scheme in the form of both the World Wide Web which uses Unique Resource Locators (URL'S) and with the underlying Internet Protocol (IP) addresses.

### 2.15.2 .NET namespacing

For those not familiar with .NET namespaces, they are a similar concept to Java packages. The concept of namespacing has been introduced with the use of virtual machines. The .NET namespacing allows for disambiguation by specifying what namespaces are being used. If for example, two classes are both called the same name, they can be clearly identified by specifying the namespace in front of the reference to the class.

## 2.16 LINQ

LINQ stands for Language Integrated Query and is a new feature provided by the .NET 3.5 Libraries. LINQ provides a way of querying collection such as arrays and lists. LINQ uses an SQL-like syntax which is demonstrated in the following example:

The collection numbers is an array of numbers that will be queried.

```
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
```

The code below is the LINQ query.

```
    var lowNums =
        from n in numbers
        where n < 5
        select n;
```

The result of that query would be held in the ‘lowNums’ variable which is a list of the following numbers:

```
    4, 1, 3, 2, 0
```

As we mentioned XPath earlier in this chapter as a way of querying, we decided to that a side by side example could highlight differences in their syntax. The previous LINQ example done in XPath could look like:

```
\\numbers[n < 5]
```

The comparison between LINQ and XPath can only really be used to observe the difference in syntax as there could be situations where one is more concise than the other.

## 2.17 Introspection

In this chapter we investigated background information relevant to the project. The areas we looked into showed a great element of overlap but can be mainly summarised in the following three categories: .NET & the world wide web, events models and event based rules.

Looking back at the chapter, it is worth noting that not all of the technologies investigated will be utilized, however it is important that readers understand what has been investigated if they were to ever extend or critique this research. The benefit in investigating many technologies with the intent of using one is also understated in the previous sentence. A chosen approach could involve a combination of many technologies or the creation of a new one based on functional aspects of, or learning from the mistake of, investigated materials.

In the next chapter we investigate the theory that links and invigorates the background technologies in this chapter so that we can bring them to fruition in chapter 4.

## Chapter 3

# Theory and Implementation Planning

## 3.1 Introduction

In the previous chapters concepts and technologies that support this research were introduced. This was a starting point to build from. This chapter addresses the progression of building towards a resulting outcome. In order to provide a goal to work towards a series of use cases that we would like to achieve have been defined. After the use cases have been defined, abstract requirements are defined. These requirements outline desired functionality to be taken into consideration by the implementation design.

### 3.1.1 Use cases

In this section key use cases are identified in an effort to define what is trying to be achieved. The use cases help to clarify scope and the intent behind functionality that is required. The use cases are listed in the following table.

Table 2 - Use Case List

| # | Use Case Name | Functionality |
|---|---|---|
| 1 | Click button, disable other UI component | Single action target |
| 2 | Click button, disable other UI components | Multiple action targets |
| 3 | Different events from different sources trigger the same action | OR type composite events |
| 4 | Use Case for composite events | Sequence composite events |
| 5 | Default selection mechanisms with timer | Timer events |

The use cases listed in Table 2 are provided in more detail as we continue through this section. URML diagrams have been made to illustrate the use cases; they can be located in Appendix A.

### 3.1.2 Use Case 1: Click button, enable other UI component

**Table 3 - Use Case 1**

| | | |
|---|---|---|
| USE CASE # | 1. Click Button1, disable Textbox1 | |
| Goal in Context | User clicks Button1 and the application disables textbox1 | |
| Scope & Level | Single Event ---fires---> Single Action<br>Primary Task | |
| Preconditions | User clicks button1. | |
| Success End Condition | Textbox1 is disabled | |
| Failed End Condition | Exception Thrown. | |
| Primary,<br>Secondary Actors | Application User<br>Application Object Model (for accessing objects by name & not pointer/reference). | |
| Trigger | Button1 Click Event | |
| DESCRIPTION | Step | Action |
| | 1 | Notify all subscribed event handlers to the Button1.click event |
| | 2 | Event caught by form action handler for Button1 click. |
| | 3 | Form action handler disables textbox1 |

### 3.1.3 Use Case 2: Click button, disable other UI components

Table 4 - Use Case 2

| | | |
|---|---|---|
| USE CASE # | 2. Click Button1, disable all the form text controls that are not read-only. | |
| Goal in Context | User clicks Button1 and the application disables all form textbox controls that are not read-only. | |
| Scope & Level | Single Event ---fires---> Many Actions<br>Primary Task | |
| Preconditions | --------------- | |
| Success End Condition | All text box controls that are not read-only are disabled. | |
| Failed End Condition | If exception thrown, attempt to undo the action. | |
| Primary,<br>Secondary Actors | Application User<br>Application Object Model (for accessing objects). | |
| Trigger | Button1 Click Event | |
| DESCRIPTION | Step | Action |
| | 1 | Notify all subscribed event handlers to the Button1.click event |
| | 2 | Event Caught by form event loop. |
| | 3 | Form action handler disables all textbox's that satisfy the condition. It does that by cycling through all textboxes on the form (via the Application Object Model) and checking the read-only attribute. |

### 3.1.4 Use Case 3: Printing is triggered, disabling printing triggers

**Table 5 - Use Case 3**

| | | |
|---|---|---|
| USE CASE # | Either of the following are triggered:<br>PrintButton.clicked, Ctrl-P keystroke, menu>print | |
| Goal in Context | While printing the printing options are disabled. When printing is finished the printing options are enabled again. | |
| Scope & Level | Single Event \|\| Single Event \|\| SingleEvent ---fires---> Single Action<br>Primary Task | |
| Success End Condition | Printing buttons are disabled. | |
| Failed End Condition | Exception Thrown. | |
| Primary,<br>Secondary Actors | Application User<br>Application Object Model (for accessing objects by name & not pointer/reference).<br>Printer | |
| Trigger | PrintButton.clicked \|\| Ctrl-P keystroke \|\| menu>print | |
| DESCRIPTION | Step | Action |
| | 1 | Subscribe form_print() method to the delegates PrintButton.clicked & Ctrl-P keystroke & menu>print |
| | 2 | The form_print() prints the form and disables printing options |
| | 3 | After printing finishes the buttons are enabled. |

### 3.1.5 Use Case 4: Composite events

Composite events are complex events that have been made from a combination of other events. For this use case we considered both sequential and AND events. The justification for chosing to use sequential events is provided in the following subsections.

#### 3.1.5.1 Sequential events

Possible examples of a requirement for a sequential composite event:

- Click then hover mouse near the click. This is used in Visual Studio to display a contextual code completion helper panel.
- Right double click
- A triple left click
- Press the same key many times. This is used in the sticky keys program in Windows Vista to check that a key has not been jammed down.

This use case highlights the requirement of an event queue (if we support sequence's of events) to see what events have preceded any event.

#### 3.1.5.2 AND events

The number of use cases for this scenario are rare, however they should not be overlooked as they would be difficult to implement if they were required. A use case for this scenario could be:

- Complex mouse gestures such as in the Opera Web Browser (Opera, 1996) could be used in combination with the keyboard.
- Hold down H and left click the mouse and the word that was clicked in the document is highlighted.
- Holding down multiple keyboard keys at once.

While requirements for AND events have been provided, it is worth stating that AND events could be translated to sequence events instead. This would mean that AND events would not be required for functional completion.

#### 3.1.5.3 Use Case

Table 6 – Use Case 4

| | | |
|---|---|---|
| USE CASE # | Right Mouse button Clicks twice on button1. Contextual help is provided for the button. | |
| Goal in Context | The user could click on the left mouse button on button1 once to perform its normal action, or in this usecase the user could perform an unusual event such as right mouse double click to get context based help. | |
| Scope & Level | Single Event then Single Event -> Single Complex Event<br>Primary Task | |
| Success End Condition | Context based help is shown. | |
| Failed End Condition | Exception logged but not thrown to the user interface. | |
| Primary,<br>Secondary Actors | Application User<br>Rule Engine (for processing primitive events and translating them to a combination of complex and primitive events.). | |
| Trigger | Button1 double right clicked | |
| DESCRIPTION | Step | Action |
| | 1 | Define a complex event type of double right click. |
| | 2 | Subscribe an action handler to the invocation of double right click on button1. |
| | 3 | Button1 is double right clicked, firing the event handler attached in step 2. |

### 3.1.6 Use Case 5: Operating System Boot Loader

Table 7 - Use Case 5

| | | |
|---|---|---|
| USE CASE # | A Boot loader (like the Linux boot manager that loads Linux or another operating system). | |
| Goal in Context | The user either selects an option to load an operating system or the system does after a certain time limit. However it the user presses a key the timer is stopped from counting down. | |
| Scope & Level | Single event \|\| Single event \|\| Event sequence ---fires---> Single Action<br>Primary Task | |
| Success End Condition | An option is executed (i.e. an operating system is booted). | |
| Failed End Condition | User interrupts timer but never selects an operating system. (this is still not entirely a failure situation as the user could at anytime in the future select an option). | |
| Primary,<br>Secondary Actors | Application user<br>Timer<br>Application Object Model (for accessing objects by name & not pointer/reference). | |
| Trigger | Timer counts down to 0 or user interrupts timer and selections and option | |
| DESCRIPTION | Step | Action |
| | 1 | If the user interrupts timer go to 2 otherwise go to 3 |
| | 2 | User selects an option. Go to 4 |
| | 3 | Timer counts down to 0 and selects default operating system. |
| | 4 | Boot selected operating system |

## 3.2 Abstract Requirements

In the previous section use cases were identified in an effort to realise required functionality. This realization allows for the comprehension of what must be provided in order to have a functional resulting implementation. The functional ability of our resulting research does not provide insight into how we might prefer to obtain that result. How we would like to obtain our resulting implementation is summarized in the following abstract requirements:

- A programming language independent solution
- Support for round tripping of rules. i.e. rules are not compiled away.
- Support for composite events
- Any code generation or user input can be done in the IDE
- Preferably a way of partial migration to our API could be facilitated, so that an all or nothing approach doesn't inconvenience new users.

As shown in the previous list, we would like our solution to not only deliver core functionality but to do it in a way that is deemed to best mitigate risks for future usability and functionality development.

## 3.3 Rule Persistency

Planning an approach to an implementation of a user interface based rule processing library requires us to first look at how we intend to store the rules. The way we store rules could be subtly added to code or could be as bold as adding an explicit file of rules.

From investigation into NetBeans, Visual Age for Java and Visual Studio the following techniques have been identified.

1. Rules stored externally (XML)
2. Rules put in generated code as attributes/annotations
3. Rules generated as code but without attributes/annotations.

### 3.3.1 Scenario 1: Rules are interpreted at runtime from an XML List.

The XML stored rules are interpreted at runtime. There is a reference to an API that chooses rules from the XML Rule List. The advantages and disadvantages are outlined in Table 8.

Table 8 - Architectural Arguments I

| Pros | Cons |
|---|---|
| XML allows the use of a potential standardized storage format. Standardization allows for better chance of third party tool support. | Potentially difficult to debug due to runtime rule construction. |
| View layer doesn't handle events. View layer doesn’t subscribe to events which reduces View code. | Aspects of the controller are in our API which is a third Party Library in developers eyes |
| With development of XAML UI's we can potentially eliminate code behind XAML. | The controller is split between the user specified code and third party code. |
| It extends what would appear to be future practices. For example the hybrid MVP patterns. | |
| Low coupling or no coupling (with aspects) to API | |
| Unit testing easier due to more independent tiers. | |
| Easier to test the API the research provides, because all ECA rules are processed in the exact same way. (at an abstract level) . | |

#### 3.3.1.1 Addressing concerns

The argument that generated rules could be difficult to debug is probably true, however there is the possibility of not generating the rules at runtime and using design time generation.

The other two arguments against this approach are stating that the controller in a model view controller based design is going to have part of the controlling logic in third party code (i.e. our API). This could be considered a disadvantage if our API is not flexible for developers. As we are planning on releasing the end product open source this is not a major concern.

### Scenario 2: Rules Stored as Annotations in Code

This scenario would likely involve a form with normal action handler methods written in code. Those action handler methods are extended with annotations. Annotations are known as attributes in the .NET community.

**Table 9 - Architectural Arguments II**

| Pros | Cons |
|---|---|
| It extends current architectural practices | Current architectural practices are changing to a MVP approach. |
| User can see most of the code used for handling events or see where it is extended. | High dependency on the API (each action that has a custom annotation depends on it). |
| It uses concepts that have been demonstrated in NetBeans. NetBeans has used comments in source code that are like early annotations. | There appears to be a massive trend towards using annotations on everything. This could lead to annotation bloat. |
| | The method has to be divided into two blocks of code. One for before the annotation is executed and one for execution after it is executed. |
| | More code generation required than the first scenario that largely doesn't inject code. This will be slower as it will have a higher dependency on reflection |
| | Less abstract than first method. This leads to a design that is more closely based on the lower level operations offered in the language (C#). |

### 3.3.2 Scenario 3: Rules Generate Code

The rules are used to generate code at design time. This is the way the user would have done it without using the outcome API of this research. It could be possible to implement this method, however reverse engineering could be difficult as it would require code parsing. This would disillusion the user by offering code generation but not supporting reverse code generation. In not supporting reverse generation of rules from code would quickly break the tool. Code parsing has been identified as outside the scope of this research however we are investigating it as idea for future development beyond this thesis.

Table 10 - Architectural Arguments III

| **Pros** | **Cons** |
|---|---|
| It extends current architectural practices | Assumes one style of method handling is always employed. Traditional VB style. |
| It can be learnt gradually as it doesn't change current method handling practices | High Dependency on the API (Each Action that has a custom annotation depends on it). |
| It allows 100% control as everything is done at design time. | More code generation required than the first scenario that largely doesn't inject code. This will be slower as it will have a higher dependency on reflection |
|  | Less Abstract than first method. This leads to a design that is more closely based on the lower level operations offered in the language (C#). |

## 3.4 Referencing

In the previous chapter the topics of: Namespacing, Unique Resource Identifiers and the Document Object Model were introduced as possible referencing approaches. While it is important to be mindful of the first two, the Document Object Model is of particular interest to this research as we require a tree like referencing system that allows us to provide an opportunity to reference any part of an application that a user interface interacts with. A User interface could potentially interact with any part of an application. The Document Object Model is conceptually appropriate for what we require however we are not dealing with a Document in our research so from now on we refer to the model adopted as the Application Object Model. This also allows changes to be made where necessary as the DOM was designed with a different domain/scope in mind.

### 3.4.1 Application Object Model Concepts

An Application Object Model is parallel to the Document Object Model in design for most of its features, however it is also different. The Document Object Model standard (W3C, 2004) currently has three levels. The parts we are adopting that are of specific interest are:

- DOM level 2 Events
- DOM level 3 XPath

DOM Level 3 XPath is of interest later in this chapter when query based referencing is discussed. The DOM Level 2 provides an event model that allows for event bubbling.

In our Application Object Model the event bubbling is provided for a select part; it is not available all the way to the root of the tree. This is partly due to 'virtual nodes' being used, and due to event bubbling currently not being considered outside the user interface level of the tree. A point that must be made is that the 'EventManager' Class provided by the .NET framework does allow for handling bubbled events from anywhere in an application, leaving the ability for the Application node or the Presenter/Controller to eventually handle bubbled events. To illustrate an example, Application object model, Figure 16 shows a simple MVP design and how it could map to an Application Object Model tree.

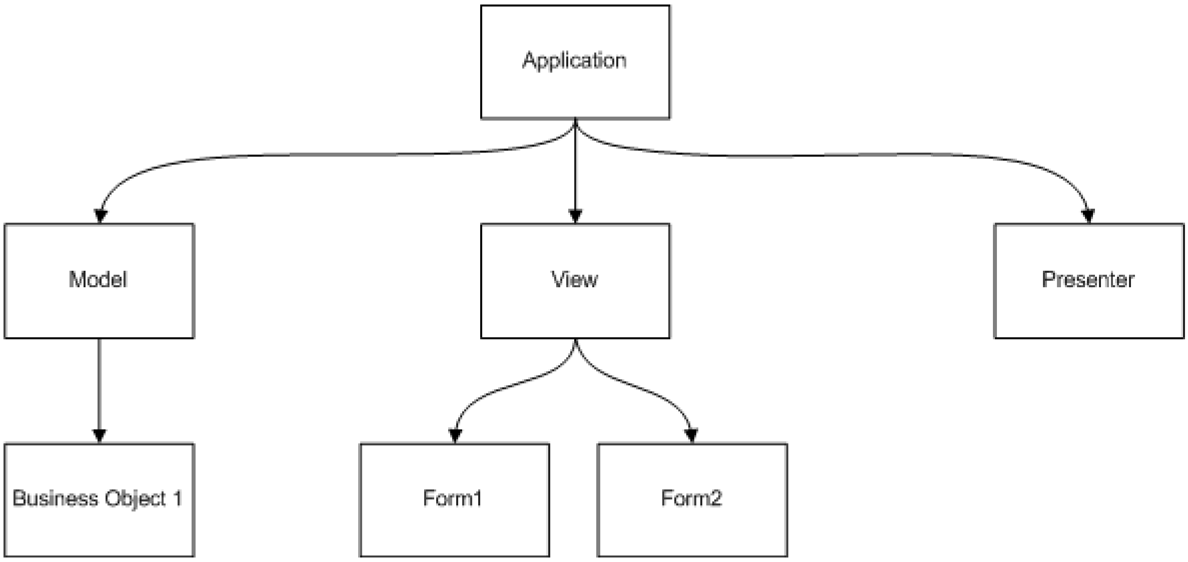


Figure 16 - Example Instance of an Application Object Model

The Application Object Model is organised in to a number of tiers which are shown in Figure 17.

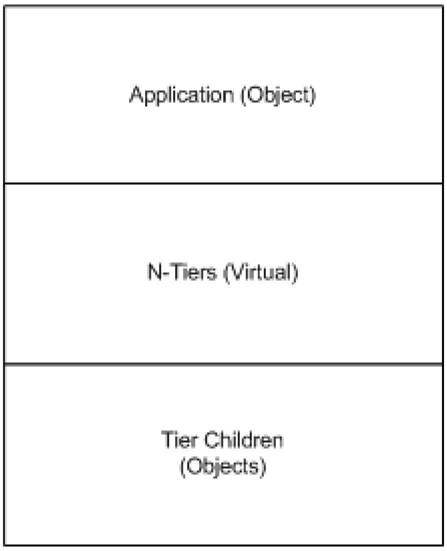


Figure 17 - Application Object Model Hierarchy

Figure 17 has virtual layers marked and non virtual layers (object layers) marked. Virtual layers are layers that are added to the application object model so that the model is logically divided into sections. It would be possible to have no n-tier layer and remove virtual nodes from the application object model tree, however the number of children from the application object would likely get very large. Virtual Nodes have been added to provide separation of concerns and provide parallel terminology in referencing to the model adopted by developers. Virtual nodes don't reference object(s).

As mentioned earlier in this section Event bubbling is a bit different when applied to the Application Object Model in comparison to the Document Object Model. A sample use of a document Object Model is show in Figure 18.

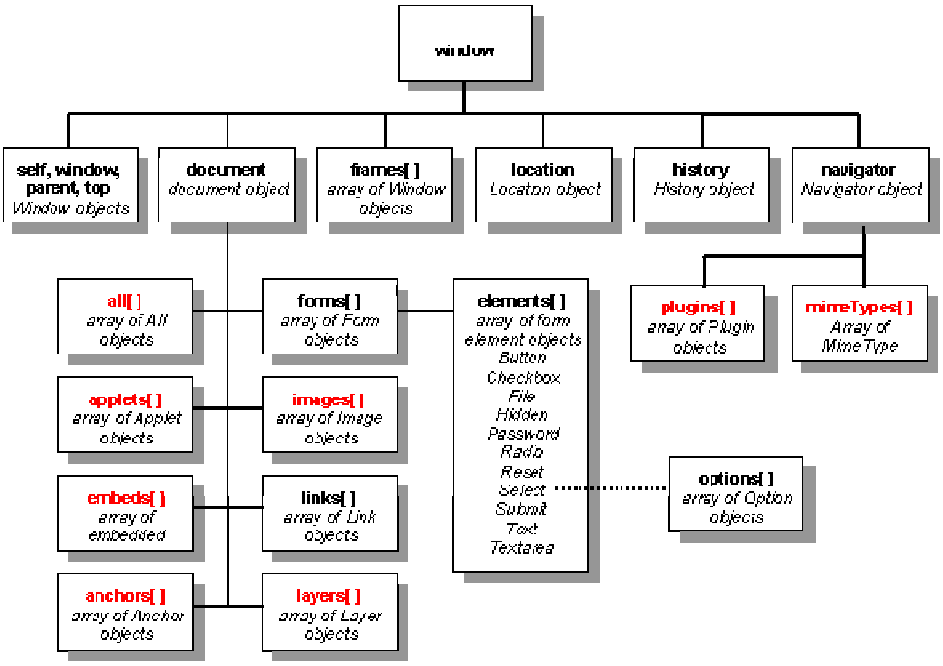


Figure 18 – Sample use of a Document Object Model (Martin Webb, 1999)

If an event occurs on an image in Figure 18 the event will bubble (assuming it's not consumed) until it gets to the root object of window. As the Application Object Model is only a referencing structure and is not the real structure of how objects interact event bubbling is restricted to Tier children mentioned in Figure 17. That means that event bubbling is restricted to bubbling to the form or window object. This is the default way that .NET handles events.

### 3.4.2 Query based referencing

When first designing the Application Object Model querying wasn't considered, possibly due to the heavy use of URI types in Reaction Rule technologies. An example of URI type usage in Reaction Rule technology is evident in the R2ML schema. It is clear from the R2ML schema that rules are assumed to come from a single source. This soon appeared inefficient in that a URI scheme provides a single source, causing many rules to be required where one could be used.

Minimizing the number of rules and maximizing the effect of each rule is preferable in design when using a high volume of rules. It is harder to quickly find a specific rule; Visual Age for Java was criticized for similar concerns (L. Chamberland IBM, 1998).

Another consideration was that using the application object model provides a standard way of considering references to the data, not unlike a relational database. This is not unlike a database in that data will be organised in a certain way and because of that we can perform standardized querying across datasets. While our Application object model is not configured like a relational database, it is configured as a tree data type. The tree Abstract Data Type is well known in Computer Science circles and because of this there has been a number of querying languages developed for querying trees. While there are many querying languages that are possible it was preferable to pick one that was reasonably popular as it is preferable that people have a minimal learning curve when using the resulting tools of this research. The main ones considered were: XPath, LINQ.

### 3.4.3 Querying by Meta Data

Referencing objects by name was also realized to be bad practice in that developers should be referencing user interface objects by responsibility. A scenario for why referencing objects by responsibility could be that we want to disable all print buttons. If a user interface control is a button and is responsible for printing then disable it. That would allow for the disabling of none, one or many buttons. If we reference by name then the approach is not extensible to runtime discovery.

While there could be a small use case for referencing objects by name, it seemed clumsy to do so when the programmer was trying to achieve an action like:

| When a print trigger is activated, disable user interface print controls and start printing. |
|---|

Pseudo code for the previous rule could look like:

```
On_Print()
{
    //iterate through all items.
    foreach(UIElement element in Form1.Controls)
    {
       // if the current item(control) is a print item.
       if (element.tag.contains(“Print”))
       {
           //disable the item.
           element.disable();
        }
    }
}
```

Currently in C# we would probably have one event to action mapping for every print item and there could be many buttons or menu items. If we allow rules to query user interface items many mappings could become one allowing for a reduction in code.

C# provides a property to all ‘FrameworkElements’ called Tag that allows for metadata to be stored within a user interface object. A framework element is a WPF element that is one of the base classes that either all or almost all WPF UI components inherit from. If all user interface items were tagged with meta-data to say what their responsibility groups were, then event rules can be written at a level that is not only higher but close to how a use case diagram might describe user actions.

If a predefined group of meta-data tags are decided upon it could be possible to code both the user interface and the rules which make it dynamic in parallel. It could be argued that this was previously possible, and to a limited extent both actions and a user interface could be implemented in parallel but there would possibly be time spent wiring events afterwards. There would be potentially no time or much less time spent on joining the user interface and its actions with the resulting API of this research, assuming that a list of meta-data tags was agreed upon.

### 3.4.4 Potential implications of non native querying

With the advent of Language Integrated Query (LINQ) for .NET there has been the ability to ‘natively’ query objects that belong to the collections library which include set data types like lists. If a non native type of querying is used, what is the overhead or implications of this? To investigate this question XPath was used. XPath although supported by .NET libraries is a non native way of querying in that objects must inherit XPathNavigator in order to support XPath querying. Another alternative would be to write a mechanism that emulates XPath queries by converting them to a native alternative such as LINQ. It is worth noting that XPath appears to not be natively supported as a mechanism of querying user interfaces in .NET.

Assuming we were to inherit the abstract class XPathNavigator then each object in our application object model must inherit this class. If that is done, there is a concern that classes in .NET can only inherit from one super class. This could provide problems in the future.

## 3.5 Events

### 3.5.1 Event Algebra thoughts

From the event algebras introduction in chapter 2 it has become clear that in requiring an implementation using event algebra there is a requirement for a high degree of functional completion if not full completion. When considering the functions available in event algebra there is a degree of parallelism between gates used in electronic circuitry. The point of comparing event algebra to gates is that by using only NAND gates (not & and) it is possible to make all other gates from that combination. It is possible that a functional implementation could be reduced in size for our proof of concept as an event sequence such as 'and' could be represented by multiple sequence operations.

### 3.5.2 Event handling approaches

In the previous chapter it was mentioned that there are three ways of handling events provided by the .NET framework as of version 3.0 . The methods of event bubbling, tunnelling and direct handling provide many possibilities of how we can trigger events by creating event to action handler mappings in code form.  A number of options were considered and are mentioned in the following subsections.

#### 3.5.2.1 Form Event Loop.

When looking at event handling approaches event loops or message pumps are options worth considering. A message pump is an approach to handling events by notifying a method (the message pump method) that an event has occurred. That method then tries to work out what type of event occurred. After which the loop works out the object that raised the event. The next step is to call the action handler of the appropriate action to handle the event.

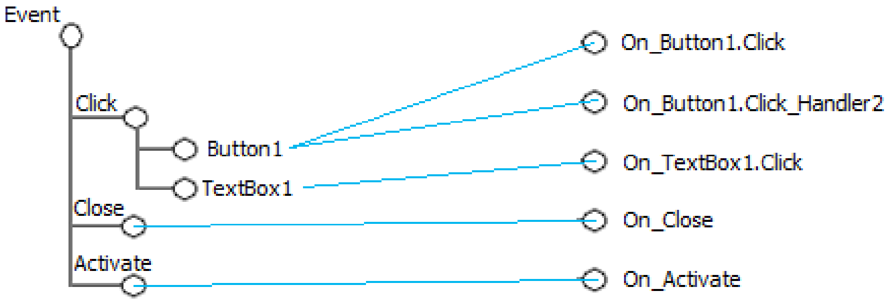


Figure 19 – Event Loop

Figure 19 demonstrates a simple scenario that uses a message pump. As the diagram shows there is a hierarchical approach to working out what event has occurred. Source code for a simple event loop like that shown in Figure 19 is provided in Appendix B Example 1. From observation, message pumps have been widely used in languages such as C++.

### 3.5.2.2 Flat Level Architecture

This would look the same as Figure 19 however it would have no hierarchy on the left. This would create event code that is harder to read because there is no hierarchical layout provided in the event loop. There is also a decentralized event architecture making events harder to trace. This is the common way that C# and Visual Basic.NET encourage developers to use. This is possibly the fastest to get started with as it appears the easiest to understand and implement in a small scale application. This method involves an action handler subscribing to an objects event as a listener or observer. The object then notifies its observers when the event that is being observed occurs.

### 3.5.2.3 Action Chooser

The way this approach works is demonstrated in Figure 20: An event occurs, then a chooser module will read the event rule list to work out what rule to execute. That rule is then handed to an execute action module which will execute the action defined in that rule.

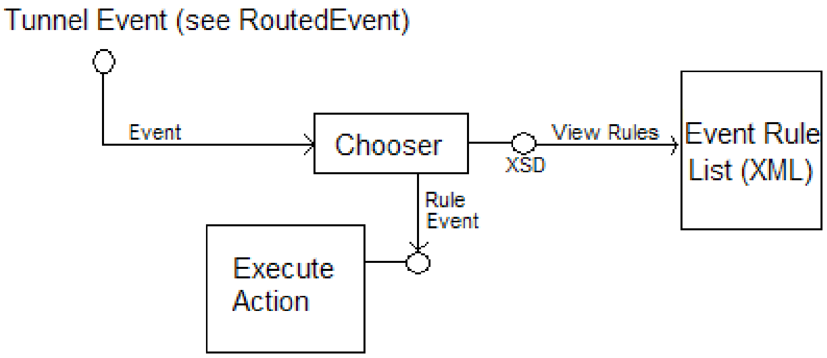


Figure 20 – Tunnel Event Chooser

This approach has the most reusable code, however it pushes a lot of logic out of code into XML. The main problem with pushing information out of code is that debugging becomes a problem. If this is unlikely to be a source of errors then it might not be an area of large concern.

### 3.5.3 Complex event processing

In order to process primitive events and transform them into complex events, we need to consider the technologies that are available. To work out what could work well it is important to consider what we are trying to achieve.

The ability to transform a list of primitive events to complex events is not as easy as initially thought in that complex events can be made from other complex events. This type of problem can be solved by using recursion. Another question that is important is how will events be removed from the event queue?

That question is not as obvious to answer as one would think. To provide an example situation to which readers might identify with, consider the composite event ‘double click’. A double click could be a double click if the frequency of clicks is fast enough. If it is not then the user has merely entered two single clicks. This may appear simple, but what if there was such thing as a triple click? We would then require a way of giving expiry times to events so that they don’t expire too early or stay in the event queue for too long. If a user clicked three times the first click could register as a ‘single click’ or contribute to a ‘double click’ then it could be possible for those clicks to expire (if the expiry time was small) before third click meant to be a triple click. If the opposite happened and clicks didn’t expire then all clicks would eventually become a triple click which is not a desired outcome.

As we are going to require an implementation we will have a look at how a rule engine might process a triple click assuming that a double click is a defined complex event. To visualize this process of realizing complex events from an event queue we have chosen to use a notation based on the event algebra by (Zimmer & Unland, 1999).

If no events other than clicks occur in the application then the first event added to the event queue would be the first click (potentially of a sequence). This is demonstrated in the first event queue. Assuming the first click doesn’t expire because the user clicks before it expires, we will progress on to the second event queue.

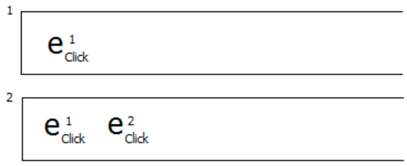


Figure 21 - Event Queue 1

There are now two click events in the queue. The rule engine will process these clicks and will translate the two click events into a double click. As shown in the third event queue in the following diagram.

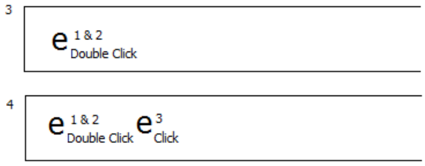


Figure 22 - Event Queue 2

The previous diagram shows in the fourth event queue the events that would make up a triple click if a double click is defined as a complex event. A triple click without a double click defined could be a sequence of three clicks, however as a double click is defined this will never be the case. If the double click shown in event queue 4 never received a click afterwards before the double click expired, then the double click would have been invoked instead of a triple click. From looking at the event queues it is worth further discussing event expiry. In an implementation of a complex rule processor we will require a timer mechanism otherwise events could sit on the queue potentially forever. It would also affect the outcome of event processing as the rule engine likely would not consider the time restrictions over events as this would be outsourced to an expiry device.

As mentioned previously we are going to require an implementation of a complex rule processor. To facilitate this implementation we investigated R2ML and found that the language offers an item called a production rule. Production rules can be used for performing actions that happen on data changes. Production Rules do not react to an event so it is largely up to the person implementing the rule engine to decide when to execute them. Production Rules are similar in concept to what many people refer to as business rules. Microsoft provides a business rule engine which is provided by the Windows Workflow Foundation (WWF) division of .NET version 3. WWF is used for the purpose of implementing rule chaining which will help identify complex events within other complex events. The implementation of this is investigated in the following chapter. JBoss are a company that provides a competing rule engine called Drools.net (JBoss, 2007). This was identified as a possibility however unfortunately it was poorly documented for .NET users so it was not worth using in comparison to the Microsoft Library.

## 3.6 Rule Entry Approach

The aim of this project is to investigate making a rule based user interface designer. It has become apparent that an extension to a traditional User interface designer is not the best approach as was initially thought. If we consider Visual age for Java which allowed for event/action mappings to be visually displayed on a user interface designer, it worked on the assumption of linking a component to an action. From investigation into using querying rather than URI techniques; we find that extending a GUI builder concept is not necessarily the best way to approach the problem. This is due to the possibility that there could be components that do not exist yet or only become subject to a query at runtime. With Visual Age for Java there are explicit start and end points in code which allow for the explicit reference of objects.

As previously mentioned in chapter 1 Visual Age was criticized for too many event mapping links shown at once. The links we would provide could be many to many relationships on a GUI builder, so the problems suffered by Visual Age would likely be amplified by the approach taken by using an indirect referencing technique such as querying.

If a visual display over a GUI builder was made for this project it would merely be able to provide an overlay of a user interface showing the rules that currently apply to the user interface. That ability is still potentially useful, however it appears to be better suited to a visualization of existing rules rather than creating new rules. If we provide code completion via reflection for a text based rule editor it should provide a clearer way to deal with event to action mapping rules. A text based rule editor is a more logical choice for now as it is likely going to not suffer from the problems VAJ did with large projects. The visualization of rules over a GUI builder has been deemed as a non crucial aspect of this research and therefore has been excluded from the scope.

## 3.7 A possible approach

From the information gathered earlier in this chapter the diagram in Figure 23 has been produced. The diagram shows an Actor, who is a user of an application that uses our Application Programming Interface (API) that is produced as a result of our research. Our API is shown on the right hand side of the diagram and a developer's application which uses our API is on the left.

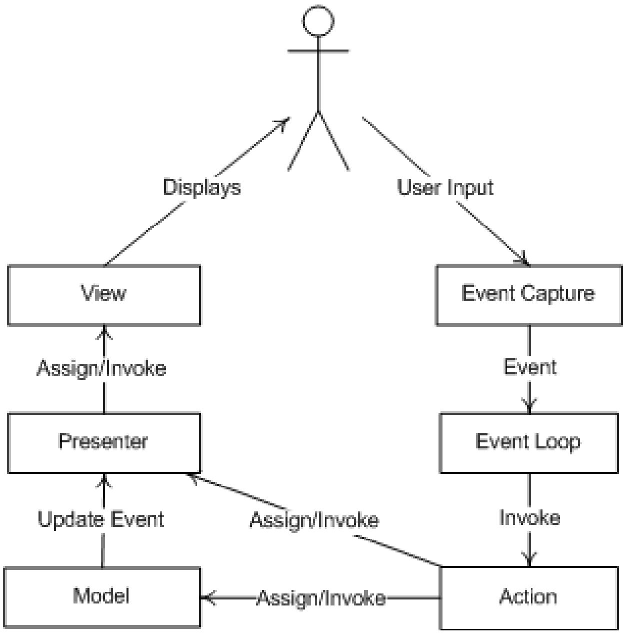


Figure 23 – A possible implementation approach

When developing our solution we will provide a way of capturing events, establishing which rule to invoke in an event loop, then invoking action code. The action code will then interact with the existing code provided by the rest of the application.

## Chapter 4

# Implementation

## 4.1 Alternative software

In this chapter the implementation of the proof of concept is discussed. To discuss the implementation, introduction of third Party software that was used in the project is required. This provides insight into problems discussed later in this chapter. Products that were considered but not used are also discussed.

In the development of a solution to show a proof of concept, it was important to try and use existing products either off the shelf or open source that could enable us to get results faster than coding everything from scratch or from the .NET base libraries.

The .NET base libraries are standardised and are supported by many people, for example the Mono project (Novell, 2004) provides an implementation of the Common Language Runtime (CLR) that can run on the *nix platforms, Macintosh and also Windows. Mono also supports Java and Python. C# like Java can be programmed in a way that is able to be used across many platforms or can be restricted to a certain platform by either bad practices or its choice of libraries. As we investigated third party libraries and tools to use, it was important to keep these concerns in mind.

### 4.1.1 Visual Studio SDK

As we were developing with C# the logical choice was to develop a plug-in for Visual Studio which accommodates making plug-ins in C#. The other main alternative would have been the Eclipse project by IBM as it can also be used as an IDE for any language due to its design for plug-ability. Eclipse offers a JDT which is a Java Development Tool Kit. This is less preferable as it is aimed towards Java, and it is easier to keep development in one language. The Visual Studio SDK provides a number of high quality samples that have aided in developing the plug-ins parts of this project. One thing that was unusual about the Visual Studio plug-in model was that there were three types of plug-ins with different names and different life cycles. The first were Macros which most readers will identify as being something quite standard among Microsoft products like Office. The second were called Add-ins. Add-ins most commonly start when Visual Studio starts and end when Visual Studio closes. The third option was to use a VSPackage. VSPackages allow the addition of new languages and designers to Visual Studio. These are not necessarily started when Visual Studio starts. Even with the different plug-in types there was a relatively consistent model.

A tool called Visual Studio SDK assist (Galiano, 2007) proved helpful to providing faster ways of developing with the Visual Studio SDK but it didn’t ease the initial learning curve. Visual Studio provides an experimental registry hive for development of plug-ins. This proved helpful because whenever a plug-in ‘broke’ the IDE in some way we could reset the hive.

### 4.1.2 Template Engine

From investigation in the previous chapters it has become apparent that code generation would be how we are going to approach rule generation. Due to our selected code generation approach it was necessary to find a way to output code through a template engine. A template engine offers the ability to have a static text document with a number of variables in it. The template acts like a form. It is sent the variables and the template engine outputs the results as text.

Our investigation first led us to a template engine called ‘String Template’. This template engine was chosen initially over NVelocity as the latter seemed poorly documented. Choosing String Template for this project, we believe, was a mistake due to the agile nature of development that was used. During development our goal was functional output, as soon as possible to prove a concept, with the intent of later refactoring. When investigating String Template it was decided to be very strict compared to NVelocity which was more like programming C#. We also found the String Template documentation difficult to comprehend quickly.

Due to NVelocity’s C# like nature it was quick to implement. NVelocity is a project taken over by the Castle Project (Castle Stronghold, 2004) after it was discontinued by the original developers.

### 4.1.3 Rule Engine

In the previous chapter we discussed the requirement for a way of translating a list of events into complex events. To do this we first looked into using a JBoss (JBoss, 2007) solution. JBoss are popular in the Java community for their libraries such as JBoss Rules and Hibernate. JBoss Rules has a .NET implementation called Drools.NET. We found the .NET version of the product (at the time) to be poorly documented in comparison to a Microsoft alternative called Windows Workflow Foundation (WWF). Windows Workflow Foundation supports XML rules, rule chaining and the ability to update rules while the engine is running. The documentation and examples of how to use it helped a lot. Windows Workflow is available as part of the .NET framework version 3 Software Developer Kit.

### 4.1.1 Schema Based Code Generation

As discussed earlier XML is a standard way of storing data in a tree like structure. As this tree like structure can be similar to source code, the XML community has devised a number of inferencing tools. These 'inferencing' tools work by interchanging XML instance documents, XSD and source code. In this section we consider the two main tools investigated and their limitations.

#### 4.1.1.1 XSD.exe

This tool is provided with the installation of Visual Studio. It is a command line tool and supports Visual Basic and C# as the languages to generate source code for. If this tool is used to generate code from an XML schema the code heavily uses arrays and it does not generate classes into separate files.

#### 4.1.1.2 Dingo project

This tool uses the command line also. This project is open source and is available at Sourceforge (Lin, 2004) and can generate C# and Java code. This project separates classes generated into separate files, however it tends to generate double the amount of classes than are necessary under its default configuration. Unfortunately this tool often overlooks certain aspects of the schema resulting often in code loss in the generated code. The code that it does generate appears to be closer to a level that a developer would be happy with. We found this application better to use than 'XSD.exe' for this project as half generated tidy code is easier to read and work with.

#### 4.1.1.3 Current problems with Schema Inferencing

Currently with the Dingo project and the 'XSD.exe' tool, there are issues in not supporting features in XML schemas such as Import tags. This is extremely frustrating, as schemas are currently often made to be modular so that they can be reused in many other schemas. Unfortunately from investigation this feature appears to be difficult to put into schema inferencing due to a graph like structure of 'include' and 'schemalocation' tags.

### 4.1.2 Unit testing

When developing this research certain areas were difficult to ‘keep working’ such as the Rule Engine project which ‘realised’ complex events from a list of events. While it is true that we would want to eventually have unit tests for the entire project, due to time restrictions, we chose to unit test critical areas such as the rule engine. The reason for this is due to the dependency that the rest of the research has upon that project. It is also important to note that testing the rule engine is difficult without unit testing as there are other projects that potentially change or view data before the rule engine receives it. By ensuring that the rule engine meets standards that we class as it working, it allows us to focus our debugging efforts elsewhere. It also allows us to update the rule engine and quickly test if it behaves as expected.

To perform unit testing this research investigated two avenues: Unit testing and Mock Objects

#### 4.1.2.1 Unit Testing Frameworks

NUnit

NUnit is an open source testing framework for .NET. It has a GUI editor for viewing tests and version 1 of NUnit was based on and was very similar to JUnit which is a Java unit testing framework. NUnit version 2 introduced the idea of using attributes (also known as annotations to Java readers).

XUnit.NET

This is a project started by two of the developers of NUnit who felt that testing frameworks were more complex than they needed to be. In the first release of XUnit.NET they removed the setup and teardown attributes/methods and replaced them with the use of an ordinary Constructor and Destructor. This project is open source and is available on Codeplex (XUnit, 2007). This is the testing framework that this research uses. This project has been used as the potential of reducing code and increased readability is likely to help with development. While XUnit.NET doesn’t provide a GUI tester, this is not considered a necessary requirement.

#### 4.1.2.2 Mock Object Frameworks

Unfortunately unit testing currently has the inconvenience of not being able to easily put complex objects into tests without initializing the actual object. In a Model View Presenter design approach it is important to consider that the presenter is passed objects that are potentially large in complexity and are initiated in another layer. Mock objects allow us to pretend passed objects exist by pointing calls to an expected result from that object.

##### NMocks

This framework appeared to provide everything necessary to our testing need. Pseudo code is given in the following example:

```
Expect.Once.On(time).GetMethod(\"GetHour\").Will(Return.Value(50));
```

While looking for Mock Object Frameworks, there were a number of recommendations to try Rhino Mocks instead. We next look at Rhino Mocks.

##### Rhino Mocks

This framework offered the ability to create a Mock Object and to then just treat the object as if it were the actual object. This led to more concise code as demonstrated in the following example:

```
Expect.Call(time.GetHour()).Return(15);
```

## 4.2 Visual Studio plug-in model.

Earlier in the chapter we discussed the Visual Studio SDK which introduced the idea of add-ins and VSPackages. These are of particular interest in the development of a proof of concept in that they will enhance usability of the tool we are aiming to provide. Figure 24 shows us how we enhance user experience by developing two plug-in projects that facilitate a Visual Studio integrated solution for code generation and data entry.

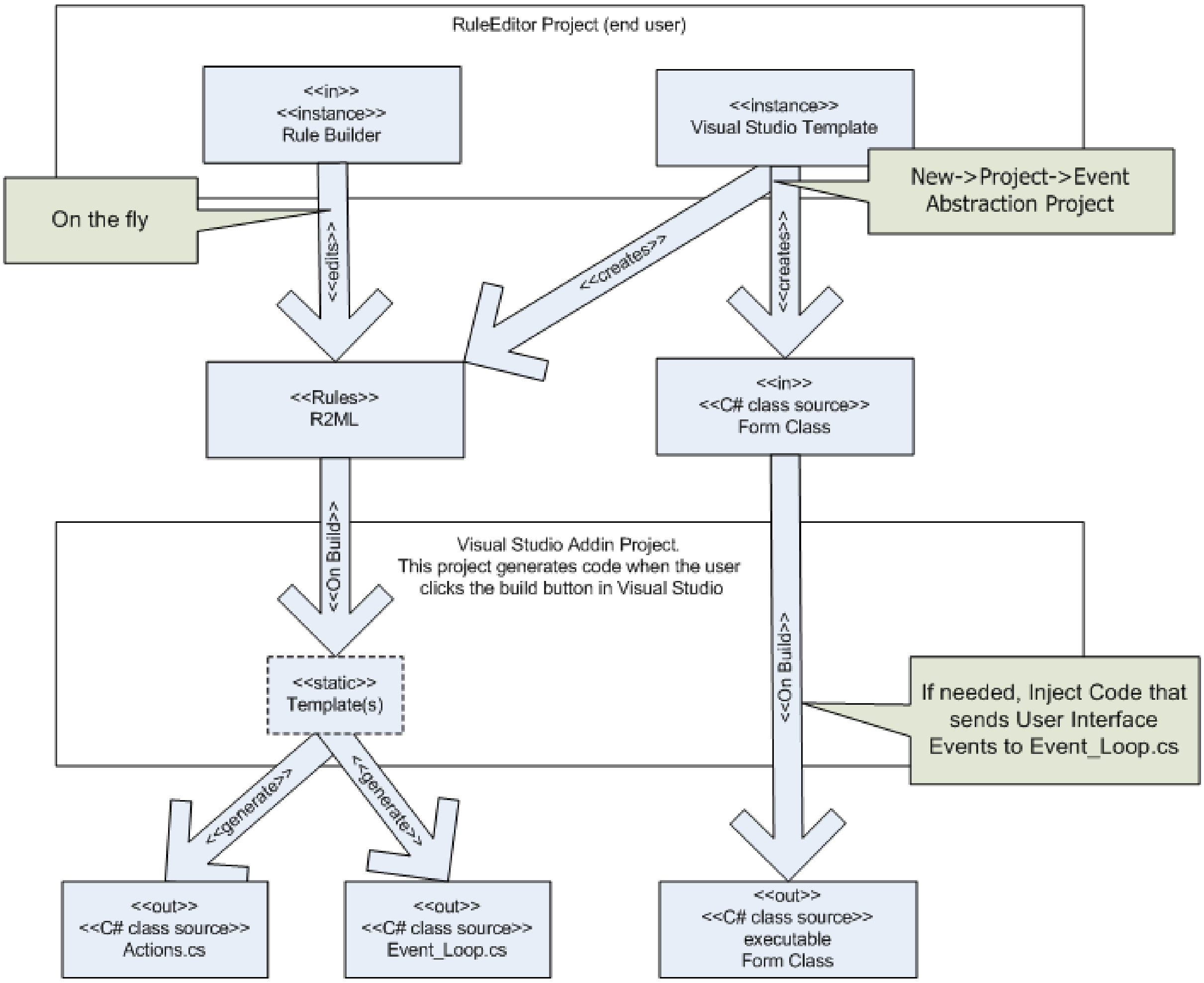
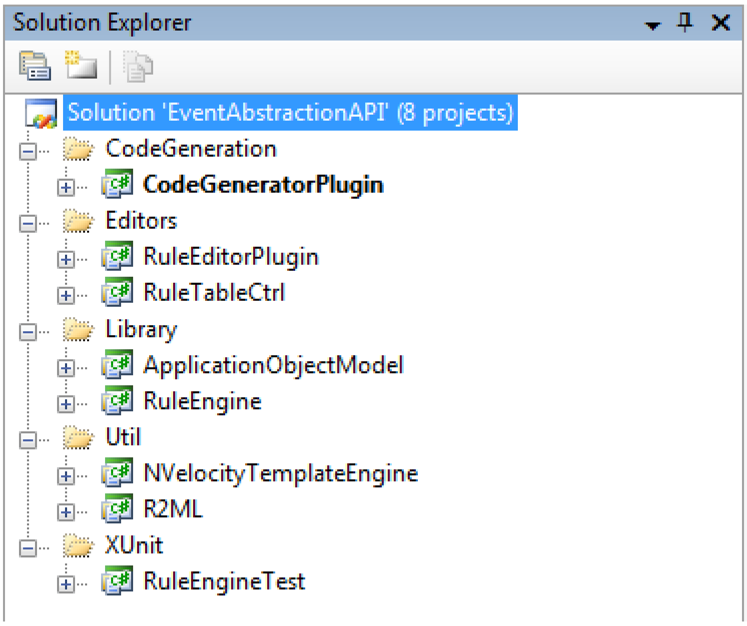


Figure 24 – Visual Studio Plug-in Interaction showing code generation and user interaction

Figure 24 shows two white boxes which represent the two plug-in projects and a number of coloured boxes with UML sterotypes denoted by ‘<<’ and ‘>>’. The top project labelled ‘RuleEditor’ is a VSPackage and provides a user interface for the rule files (.r2ml) and provides Visual Studio templates so that a user can create a project that uses the rule builder. The visual studio templates have sterotypes indicating the creation of an R2ML file and the form class. The template can generate the files necessary to use the rule editor/builder. The Rule editor can edit the rules file, which is shown in the diagram with an edits stereotype. After the user is satisfied they can click the Visual Studio build or run buttons. Arrows with the ‘On Build’ stereotype on the diagram show that data is taken from the R2ML and Form class files when the build event is triggered and passed to the Visual Studio Addin Project. The arrows also show where the information is used for final output which is denoted by the ‘out’ sterotype. Information from the R2ML file is shown to be fed into static templates. These templates are used to generate two outputs as shown in the diagram. The output items are shown on the bottom of the diagram: Action.cs, Event_Loop.cs and the Form Class. The inputs where the resulting outputs came from are denoted by the ‘in’ sterotype. Although the Visual Studio templates contribute source code, the source code from the templates have no user input before creation of the Form class and the R2ML file.

## 4.3 Solution Projects

To maintain understanding over the code being developed in this research, constant refactoring was required. The implementation was programmed in an agile way which was also a reason for constant refactoring. An agile approach was taken due to being unfamiliar with a practical understanding of the Visual Studio plugin model.Refactoring is a necessary evil in that it allows for the consolidation of progress. If developing a project was considered to be the same as rock climbing, see (Y. Yongqing EuropeLoan Bank, 2002), refactoring could be considered a side step. Side stepping can help to get into a better position for the next stage. With the gain of experience during the process we eventually worked out a logical division of the project which is shown in Figure 25.



Solution 'EventAbstractionAPI' (8 projects)
- CodeGeneration
  - **CodeGeneratorPlugin**
- Editors
  - RuleEditorPlugin
  - RuleTableCtrl
- Library
  - ApplicationObjectModel
  - RuleEngine
- Util
  - NVelocityTemplateEngine
  - R2ML
- XUnit
  - RuleEngineTest

Figure 25 - The Solution Explorer

Figure 25 shows five folders which are solution folders. These folders are a way of separating concerns. In an abstract sense the folders outline what the implementation provides. How those concerns are addressed is provided by the projects contained within the folders. The next part of this section investigates the concerns each project addresses and how the implementation works.

### 4.3.1 Code generation Plug-in

This project can be considered in achieving two concerns. These concerns are: Subscribing to the Visual Studio build event and generating source code.

#### 4.3.1.1 Subscribing to the Visual Studio Build event

To subscribe to the build event in Visual Studio the best option appeared to be to create a Visual Studio add-in. The appeal of an add-in was that add-ins can start when Visual Studio starts, this means that we can listen for events from when Visual Studio starts, where as a VSPackage starts when it is first accessed, which is usually delayed.

By default an add-in implements the interface 'IDTExtensibility2' which provides a method called: 'OnConnection'. The 'OnConnection' method then subscribes to the 'OnSolutionOpened' event to be notified when a solution is opened. When a solution is opened the add-in checks to see if the project contains an R2ML file. If it does then we assume that this project is using our API and we then subscribe to the 'OnBuildBegin' event. That subscription to 'OnBuildBegin' allows for code generation before the build process has begun.

#### 4.3.1.2 Generating the source code

To generate source code we have taken the approach of using NVelocity templates which were introduced earlier this chapter. NVelocity provides benefits to this project as the code that is generated is largely static with a few minor areas that require dynamic generation. Figure 26 shows the process of code generation from the user's point of view in the IDE. What it does not show is the NVelocity Templates. Due to the already large size of the diagram they were omitted. EventLoopBuilder and ActionBuilder call velocity templates that generate code based on variables they submit to the NVelocity Library. 'EventLoopBuilder' and 'ActionBuilder' expose the method of 'GenerateCode' which provides an abstraction from NVelocity. This is important as we could choose in the future to replace NVelocity or upgrade it. Figure 26 starts at the left hand side by showing a user create a new instance of Visual Studio. The user later initiates events on the IDE.

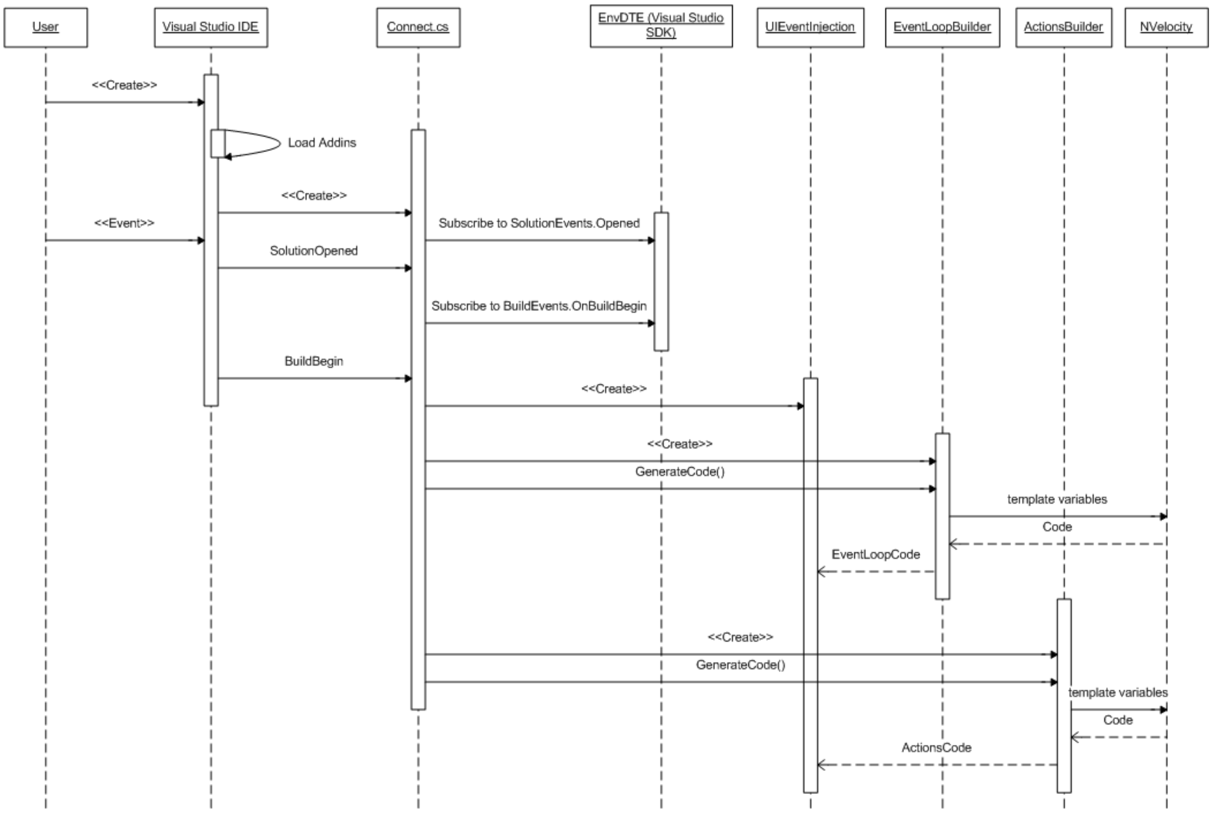


Figure 26 - Accessing the code Generation Classes

While Figure 26 shows how the code is generated it doesn't show how the users code will interact with the generated code; this is shown in Figure 27. Figure 27 shows a user and a 'Window1' object which represent the user interaction and the communication from 'Window1' (a window in the users application) to the generated code 'EventLoop'. Following the diagram further shows communication within the generated code. The generated code then passes the appropriate calls to pre compiled libraries: 'R2ML' and 'ApplicationObjectModel'. 'EventManager' is a feature of the .NET framework but is included in the diagram as it is a key feature of the event stealing technique we used to re-route events. The object referred to as 'Actions' represents an instance of 'Actions.cs' which is generated code like 'EventLoop'. Examples of output code for 'EventLoop' and the Actions classes are available in Appendix B.

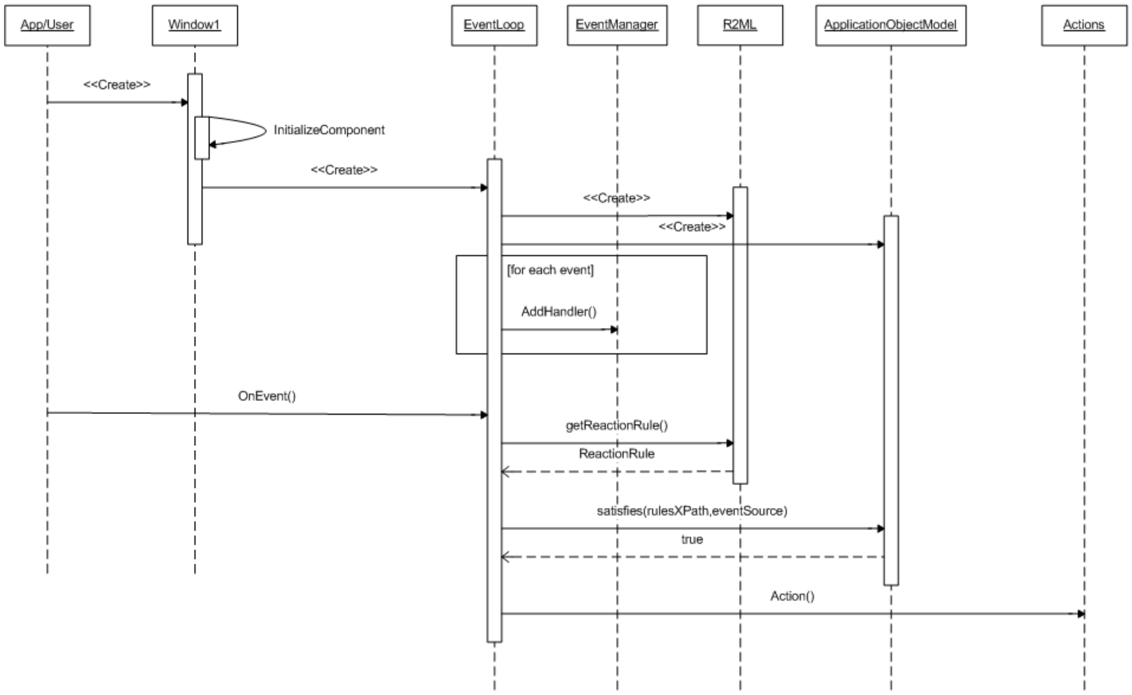


Figure 27 - Wiring up generated code to User Code

#### 4.3.1.3 Event Stealing

Unfortunately in the effort to re-route all events that the application has received, the user has to deal with ‘event spam’. Subscribing to all events was inevitably going to give us unwanted events, but unfortunately as we are looking at the order of events there are some events that interrupt event sequences. For example if we want to detect a double click (using a sequence of 2 single clicks with the research API, i.e. not the inbuilt double click handler in .NET) we might expect to mainly see mouse button events. The actual event log of a double click starting and ending at both click events is:

```
Click
MouseUp
previewMouseMove
MouseMove
PreviewMouseDoubleClick
Preview Mouse Down
Got Mouse Capture
PreviewMouseMove
MouseMove
MouseDoubleClick
MouseDown
PreviewMouseUpOutsideCapturedElement
PreviewMouseUp
LostMouseCapture
PreviewMouseMovew
```

MouseMove
Click

The list of events shows that the .NET frameworks ‘MouseDoubleClick’ event precedes the ‘MouseDown’ event, which happens before the ‘MouseClick’, however the point of the list is to show that a mere sequence of $E_{DoubleClick} = ;(E_{Click},E_{Click})$ is not enough to specify a double click.

To rectify the high volume of events, either an event filter or a wild card operator could be introduced. A filter would only allow events of interest to affect the complex events. A wildcard operator would likely change a double click event to be declared like: $E_{DoubleClick} = ;(E_{Click},*,E_{Click})$

It is worth observing that there is likely to be an increased processing penalty if the user wants to inspect mouse movements. Mouse movements could offer a lot more data to process, as they could be frequent if the user uses a mouse.

### 4.3.2 Rule Editor Plug-in

The purpose or concern that this project addresses is to provide a graphical editor so that the user can enter rules. A further goal was to provide the ability to show the underlying data behind the editor much like a GUI builder can show the code it produces. The purpose of this project was also outlined in Figure 24 which showed the overview of the Visual Studio plug-ins. To identify what was expected from this project we came up with Figure 28.

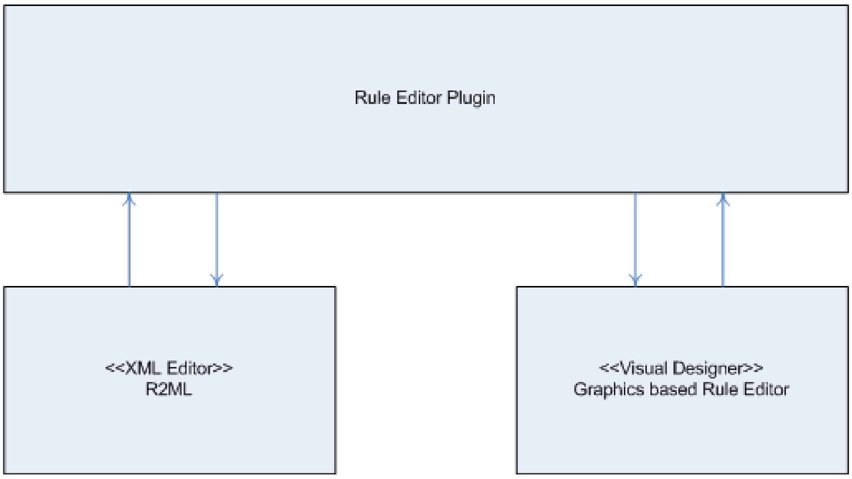
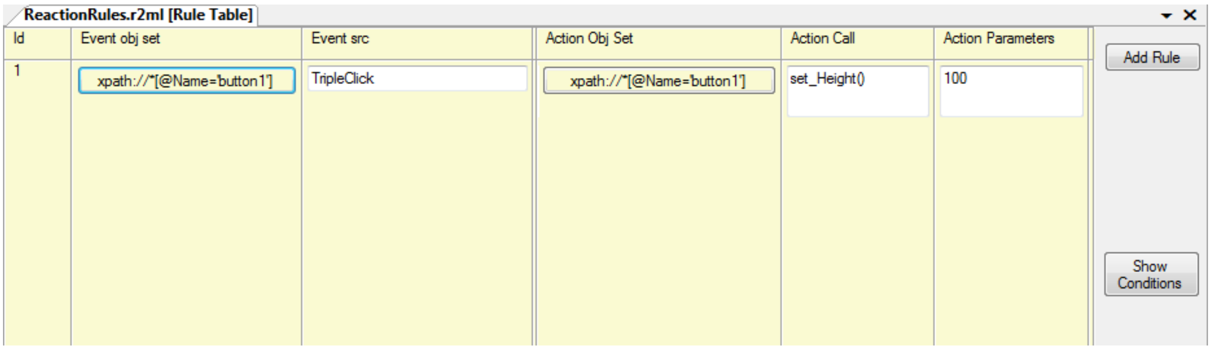


**Figure 28 - Rule Editor Plug-in Overview**

Figure 28 allows the illustration of three important concerns that were addressed specific to the Rule Editor project. The first goal was to get an XML editor working so that the R2ML could be shown and edited. The second goal was to implement the communication between the XML editor and the main code of the Rule Editor plug-in. As that concern became very large it became the R2ML project that is discussed later in this chapter. XML is not directly of any use for machine understanding so it requires interpreting via a form of parsing to retrieve elements of interest. The third goal was to implement the visual designer. This was also separated into another project as it not only had the potential of getting large, but it had coupling to Visual Studio because it was a part of a VSPackage project. Moving it to its own project removed that dependency on Visual Studio allowing it to be reused if required in the future. In order to make an XML editor, we just create instances of the inbuilt XML editor in Visual Studio. To make our Visual Rule editor, we created the RuleTableCtrl project.

### 4.3.2.1 RuleTableCtrl

The RuleTableCtrl project stands for rule table control. This user interface control provides a reusable control for entering rules. The control is demonstrated in Figure 29.



| Id | Event obj set | Event src | Action Obj Set | Action Call | Action Parameters |
|---|---|---|---|---|---|
| 1 | xpath://*[@Name='button1'] | TripleClick | xpath://*[@Name='button1'] | set_Height() | 100 |

Figure 29 - RuleTableCtrl

While this project is constantly changing to accommodate better usability the concepts of input are likely to remain similar as they are based on the concept introduced earlier as ECA-P rules. There is a ‘Show Conditions’ button the right side which can show two extra columns to enter in the condition and post condition fields. Additional rules are accommodated with the ‘Add Rule’ button on the right side. To enter data into the fields the user can click on the control and a drop down box appears for most of the fields. The drop down boxes are different for each field but are loosely based around the concept of Visual Studio’s intellisense which helps guide the user with input as they type.

### 4.3.2.2 Potential for future development

It was stated earlier in this thesis that the initial goal was to provide a way of drawing the rules, and our initial assumption on this was to incorporate it into the GUI builder offered by .NET. While we found this to not be as practical as initially thought, it could be implemented even if only in a read only fashion for visualization of rules. In further research it could be possible to implement such a solution however there appears to be many issues that would require investigation. The main issue that caused us to avoid this approach is that some rules may not apply until runtime and hence wouldn’t make sense drawn on a GUI builder.

### 4.3.3 Application Object Model

In the previous chapter the Application Object Model and the theory behind it was introduced. In this section the implementation and the challenges that undertaking provided are discussed. In the implementation of this project it was our goal to leave the ability to implement two query languages: XPath and LINQ with the intent of only implementing one during the course of this research. As LINQ is providing functionality by default upon .NET 3.5 collections it is assumed that we can implement LINQ with minimal effort. Implementing XPath is likely to be more difficult as we would either have to provide a means of translating an XPath query to LINQ or inherit the ‘XPathNavigator’ class provide by the .NET framework. As LINQ was experimental at the time of this project it was decided to use the ‘XPathNaviagator’ approach. Another reason was that writing an XPath to LINQ converter would be difficult to write to encapsulate all XPath due to the size of the language. Also the conversion is possibly not a simple one to one mapping. Unfortunately C# only supports single inheritance. This means that any other query types added, other than XPath or LINQ would not be able to use direct inheritance and would have to be translated or use a work around technique for multiple inheritance. The projects code is divided into two areas: The object model definition and the construction/interaction with the model.

#### 4.3.3.1 The Object Model Definition

The model definition consists of a node definition that subclasses ‘XPathNavigator’. ‘XPathNavigator’ is an abstract class that is provided under the ‘System.Xml.XPath’ namespace. The node definition allows for bidirectional references by storing references to the parent and children nodes. The node definition also allows for the storing of a name and an object. The names are used for the querying and the objects are extracted for the result of a query (like a dictionary abstract data type).

#### 4.3.3.2 Interaction with the Model

The Interaction with the Model is divided into four Code Regions: Initial Construction, Runtime Construction, Query, and Execute.

The construction regions are to do with building the object model. The initial construction is the constructor which makes the initial object model under a default configuration. The runtime construction provides additional functionality to modify the object model for a higher level of customization.

The query region provides methods for querying the object model.

The execute region performs actions over result sets from queries. This functionality was included to reduce dependencies on the assembly. Without providing this functionality the user would have to reference the node type and perform the action themselves. While the user can still do this, it is not a preferred interaction due to an increase in coupling.

### 4.3.4 Rule Engine

The rule engine uses Windows Workflow Foundation to provide a basis for dynamic rule injection and to better enable rule chaining. Windows Workflow Foundation uses XML to store its rule definitions. This was used to our advantage in development of the dynamic rule injection. As rules are created in Visual Studio our rule engine needs to be updated. Due to XML's tree like nature we can inject new rules, as XML, easily into the Workflow rule list. The next step was to figure out a way of writing a rule to inject into the rule list. In order to provide that functionality we decided to, again, use NVelocity. Due to the reasonably static nature of the rules we were inserting this worked well.

However development was difficult in an unexpected way. During development of the rule engine the smallest change would have massive consequence for the rest of the projects. As all projects needed to be run together it was decided that unit testing was required to consolidate efforts. Writing test cases for this part of the project was initially difficult due to the use of complex object inputs, however the tests aided progress of the development later on.

Windows Workflow was initially employed to allow the easy development of advanced rule engine features such as rule chaining. While it provided that functionality it also provided the ability to treat the engine as a task that can be interrupted. This provided a convenient way of interrupting the rule engine if required. This functionality was not initially considered, but is important as the engine could take too long and might need to be interrupted so that it doesn't act unresponsively to users if it is processing a lot of data. This concept can be considered the same as context switching, which is employed by many modern operating systems to simulate parallel processing by swapping tasks on and off the CPU so that all tasks get their turn. If a task takes too long then it could make other processes appear unresponsive. Also if the task took too long itself without reporting back then it could be considered unresponsive.

In order to make the rule engine more responsive a system of 'popping events' has been set up so that events can expire. The expiry is done by using a time stamp, adding a variable amount of time and checking to see if that time is greater than the current system time. This is important because otherwise events could stay in the engine for a long time being processed over and over again. It is also possible that if an event didn't expire after a certain time that two single clicks for example could be seen as a double click. A double click is two clicks, but it has a time restriction stating that the two clicks must happen within a certain time frame.

As the rule engine project is a dependency project in that the user does not enter data into it but rather data is sent to its instance from another project, it has a class that provides an interface for interaction. The interface is called ‘EventAlgebra’ and has two inputs and one output. The Inputs are Register and Receive. Register is a method that declares a rule. The rule is a declaration of a complex event. Receive is the method that accepts primitive events which are fed to it from the event re-routing done by the Code Generation project. The output is an event called ‘RaiseEvent’. As this is a dependency project it is the project that will be referenced. A project that references it will not be referenced by this project as circular dependencies are considered bad practice. Circular dependencies also provide unnecessarily high coupling which is negative if this project was to be reused. To get around that problem the output was made an event. In .NET events are a good way to provide weak referencing. An event in .NET provides a mechanism so that an object can subscribe to the event. The event maintains a list of subscribers and notifies them as in the observer pattern (Gamma, 1995). This is considered weak referencing in that it is still a way of referencing the calling project, but the reference is in the project hosting the instance of this project, keeping compile time references unidirectional. To visualise this referencing structure in relation to this project Figure 30 has been provided. That diagram shows normal method calls as solid arrows, and shows events as dashed lines. Dashed horizontal lines can be thought of as weak references and solid lines can be thought of as normal references.

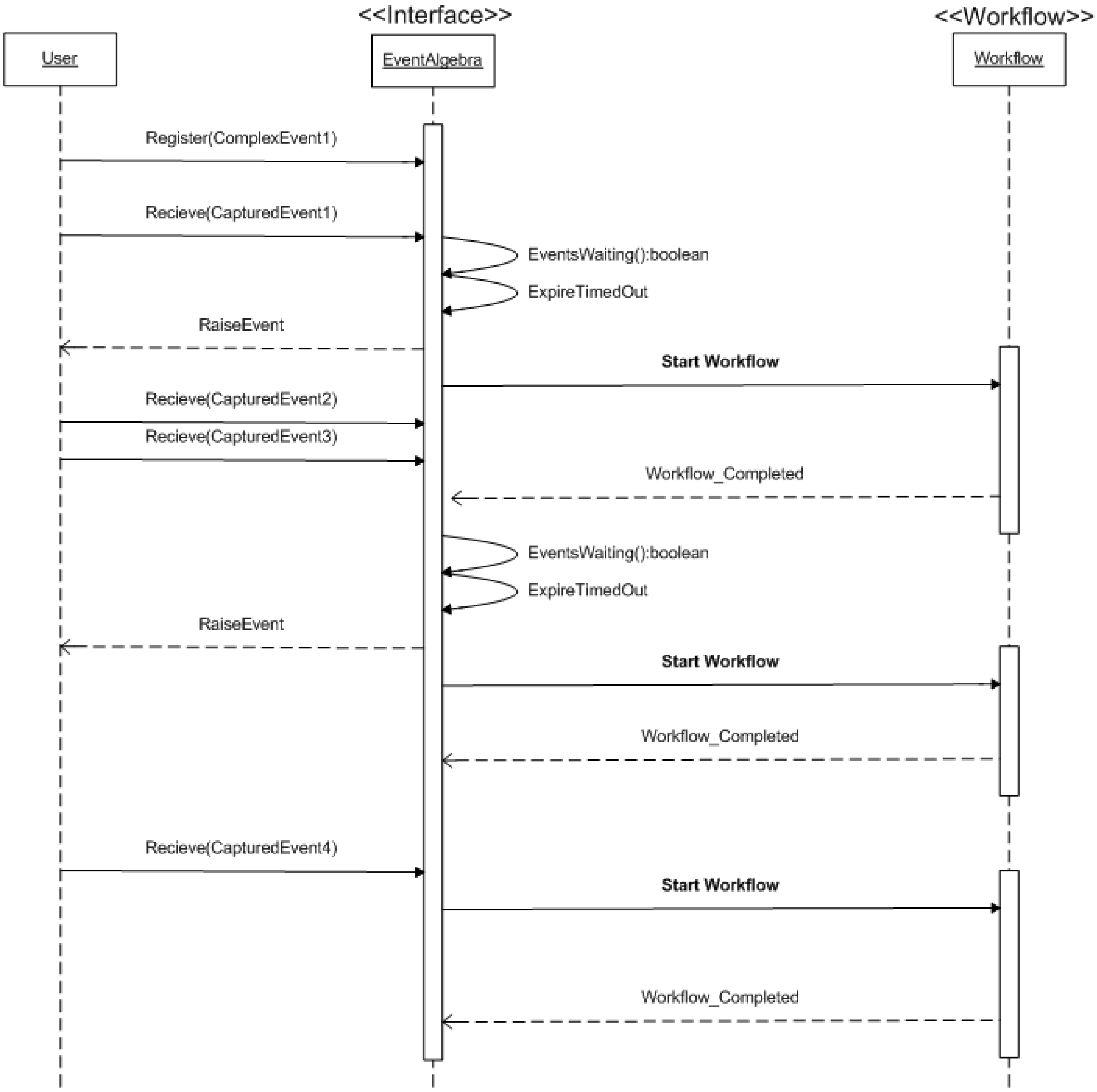


Figure 30 – Event Algebra Interface

Currently this project works but has room for improvement. An event is expired when a workflow is created or when we get a new event via the receive method. It is possible for the last event to stay on the queue until another event arrives. This could be solved by many approaches, such as polling the last event until it expires or noting the expiry then restarting the rule engine. This was not deemed a high priority because we were working on proving the concept. Another point for not correcting it is that if the rule engine is running when the last event is added then the rule engine will restart and will process it, so there is not a large use case for that problem as of yet. In the future however this will need to be addressed.

### 4.3.5 NVelocity Template Engine

This project is adapted from third Party software. It was included as it was an easier way to start NVelocity. This project is a factory that wraps NVelocity to provide use case like ways of using NVelocity. It allows for:

- String output
- Outputting to a memory stream
- Writing to a file

NVelocity by itself has one constructor rather than three and was less obvious to use while getting started. As development progressed this library was customized to meet our requirements better. Change was minimal however.

### 4.3.6 R2ML

This project is not dependent on other projects however it has a lot of projects that depend on it. The R2ML project allows for the interchange of R2ML valid XML to Variables. This allows for the ease of extracting information stored in the R2ML. At the time of implementing this project version 0.4 of R2ML was released. Since then version 0.5 has been released. We have not updated the library to 0.5 at the time of writing due to the non critical nature of the update. The update is considered non critical as it appeared to have little or no impact on what we were doing.

The implementation of this project was anticipated to be very easy, however this turned out to not be the case. The problem with the implementation was briefly mentioned in the introduction of XSD.exe and Dingo. The specific issue was that the schema was spread over many files, and that schema imports are not well supported in schema code generation tools. Another minor problem that arose was that .NET namespaces don’t support a namespace name starting with a number. Dingo by default attempted to put the code it generated into the XML namespace defined in the R2ML schema which contained the year of 2006 as one of the URL folder names. Schema code generation when it works saves a lot of time, but when it doesn’t work it can equally be a waste of time. The code generation tools also tended to use arrays rather than lists, so when creating new rules with the R2ML project, we had to be careful to initiate arrays. Currently the project still uses arrays but if possible should be changed so that it uses lists.

R2ML also unfortunately makes an assumption that a rule is performed upon one element. It was previously discussed last chapter, querying offers better design by making rules higher level. To accommodate queries in the R2ML, we just replaced the URI with a query. That worked well, however Visual Studio gave a warning in its XML editor that the URI is invalid.

This issue was that the URI being used in our XPath queries was said to be non valid to the URI specification of the http://www.w3.org/2001/XMLSchema:anyURI schema/element. The official URI specification (RFC 2396) was investigated to obtain what symbols were valid. Both .NET and the W3C use RFC 2396 as their URI definition.

The following was the invalid URI:

```
xpath://*[name='test1']
```

From investigation the characters {*} {'} are valid. The xpath:// is also valid. The = symbol and all alpha numeric (both upper and lower case) are valid.

The specification shows that the square brackets are not valid.

```
Berners-Lee, et. al.        Standards Track                [Page 10]
RFC 2396                   URI Generic Syntax             August
1998
   Other characters are excluded because gateways and other transport
   agents are known to sometimes modify such characters, or they are
   used as delimiters.
   unwise      = "{" | "}" | "|" | "\" | "^" | "[" | "]" | "`"
   Data corresponding to excluded characters must be escaped in order
to
   be properly represented within a URI.
```

To accommodate for the limitations of the URI scheme chosen by R2ML, it was decided to use an escape sequence and use the square brackets octet number. i.e. like space is converted to: %20 (where 2 and 0 are hex digits). The one for '[' is %5B and ']' is %5D

The query would become:

```
xpath://*%5Bname='test1'%5D
```

While an easier way out would be to just change the R2ML schema, it is not conducive to maintaining a common standard. This problem also, has been proven not to be insurmountable with the existing schema.

### 4.3.7 Visual Studio Templates

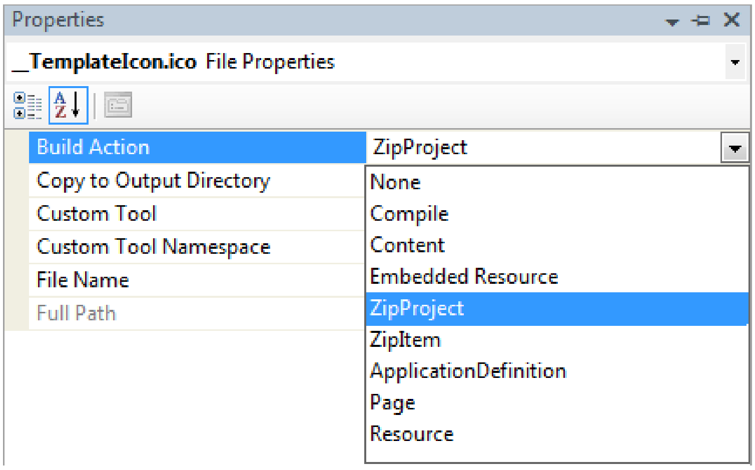
During the initial stages of development it was envisioned that a resulting implementation would be built in a start to end way. What that means is that with a normal Visual Studio project it is possible to create a new project, then hit run and it will compile. While this was always a usability goal it took a 'back seat' in initial development as it was not a core aspect of a proof of concept.

After the initial stages had been linked together an additional development iteration was undertaken to go back over the project with start to end usability in mind. Defining a Visual Studio template was identified as a starting point, because creating a new project is likely the way most projects start. To install templates into Visual Studio it was necessary to have a

VSPackage project and to elevate Visual Studio's permission levels to Administrator (this was the case with a default Windows Vista Ultimate setup).

The reason that we mentioned a VSPackage being necessary was because of an attempt that was made to use VSTemplates with an add-in project. While the template files could be made in the add-in project, we could not manage to get the build scripts to zip up the templates and copy them to the Visual Studio directory using recommended existing facilities. As we did not want to waste time on getting distracted with something that should work, we moved the VSTemplate files to the VSPackage project.

Figure 31 shows the options available to a VSPackage. The importance of the diagram for those familiar to Visual Studio build actions is the addition of a 'ZipProject' and 'ZipItem'. These are options that allow for automated zipping of a template project or of a template item. The difference between ZipProject and ZipItem is that a ZipItem is a VSTemplate for adding a new item to an existing project where as ZipProject is a VSTemplate for a creating a new project.



**_TemplateIcon.ico** File Properties

Build Action: ZipProject

Figure 31 - VSPackage Build Actions

Figure 31 also will raise the question with many readers as to what the other build actions do. Build actions are a way of telling the compiler what a file in a project means, for example is it compiled? Do we embed a resource in the assembly (note: resources such as images can be stored in an assembly)? Or do we just access it from the file system? Also for items that are compiled, there are different compilation options for different items, especially with the introduction of XAML user interfaces which are compiled to a Binary application Mark-up Language (BAML) and are linked to the code behind. It is possible to add custom build actions by manipulating the .csproj file which if opened by a text editor is an XML build script using the MSBuild syntax.

### 4.3.8 Build Scripts & Registry Modifications for Visual Studio

To install the Visual Studio plug-ins and put the assemblies in a common place that can be referenced, it was decided to use a build script. We decided to use MSBuild which is the default build script provided with Visual Studio. MSBuild uses an XML syntax to provide information to the IDE and to perform Actions during different parts of the build cycle. The reason a build script was chosen is because the source code is changed often. Also chances are that most people using our API will want the source code so that they can debug the API. A sample part of a build script is given in the following text:

```
<CommonRegAttributes>C:\Program Files (x86)\Visual Studio 2005
SDK\2007.02\VisualStudioIntegration\Common\Source\C#\RegistrationAttri
butes\</CommonRegAttributes>
```

The build script was mainly used to:

- Copy the plug-ins into the Visual Studio plug-in folder
- To register the VSPackage project with the registry.
- To zip Visual Studio templates.
- To copy the API to a directory in ‘Program Files’ from where the assemblies can be referenced.

## 4.4 Further Implementation possibilities

As this research has a limit to the amount of development required, and development could be considered a continuous process with no end, there is no shortage of features left to implement.

Currently Microsoft are working on parallel extensions for .NET (Microsoft Corporation, 2007).This extension could provide the potential to easily develop a way to process workflows in a parallel way or alternatively the workflow could be reconfigured by use of a factory class so that it is actually many workflows operating in parallel. This could help speed up processing of complex events that are currently likely to be a bottle neck if a large number of frequent events are subscribed to.

Implementing the functionality for a tag Interface could be beneficial to design because if an interface was specified before a project started, the user interface and the rest of the application can interact assuming the user interface will have the appropriate tags implemented. An example could be if we wanted to disable all print buttons. In that example ‘print’ would be a word that could be declared in a tag interface. An interface would be like a dictionary containing the terms that are considered to be known. An interface if written in an extensible way and updated as required, could potentially be used for every project a developer makes. As some projects could be very specialist, developers may want to use many interfaces, some being general and reused and some that are unlikely to be reused.

In this section a direction that this project could be taken in terms of development has been outlined.

Chapter 5

# Validation

In the previous chapter the implementation was investigated. To verify if the implementation can be considered a success a way of providing metrics must be found. To provide these metrics it was decided that two example applications should be compared. The first of the two applications uses our API and the second application does not use our API. The second application has been designed so that it achieves the same functionality that the first example application is attempting to show. After the comparison we also mention some architectural benefits of using the previously discussed implementation as a developer API. For access to the source code see (Wilburn, 2007).

## 5.1 Comparison

The example scenario that is used to compare our API with a 'normal' C# alternative, involves a print menu item and a print button. If either are triggered both are disabled. The Button and menu item will have a tag that indicates that they are a 'PrintItem'. Referencing components by name will not be allowed. The example's source code is provided with the source code of the API.

To provide the following statistics NDepend (NDepend, 2005) was employed. We start by comparing overall statistics of interest in the following table.

| | With Out API | With API |
|---|---|---|
| IL Instructions | 198 | 410 |
| Cyclomatic Complexity | 7 | 9 |
| Assembly References | 4 | 7 |

The previous table shows that there is over twice as many Intermediate Language (IL) instructions in our API. IL instructions are the product of compiling source code to assembly. There is also a slightly higher Cyclomatic Complexity (CC), and a higher number of external assemblies used with our API. Cyclomatic Complexity shows the number of decisions (i.e. 'if', 'while' and 'for' statements) that can be made in a procedure. The NDepend application recommends that a CC higher than 15 is getting hard to maintain, and above 30 requires a procedure to be split into smaller methods. While that is probably a matter of opinion, it does provide a comparison to judge the average CC values provided previously. In the following tables we delve deeper into individual files to compare differences at a class level.

| | With Out API | With API |
|---|---|---|
| Window1.cs IL Instructions | 177 | 55 |
| Window1.cs Lines of Code | 27 | 8 |
| App.cs IL Instructions | 21 | 21 |
| App.cs Lines of Code | 1 | 1 |

It is worth observing from the previous table that 'App.cs' has the same lines of code and same number of IL instructions with and without the API. This is expected as the files are both generated by Visual Studio. This statistic has been included as it shows the reader that we do not modify the 'App.cs' file in our API.

The API generated code does not apply to the non API demonstration. The statistics on code specific to the API is in the following table.

| | With API |
|---|---|
| Event_Loop.cs IL instructions | 298 |
| Event_Loop.cs Lines of Code | 41 |
| Actions.cs IL instructions | 26 |
| Actions.cs Lines of Code | 2 |

With the addition of the lines of code in the previous table, it is evident that the API example uses more code/IL code than the non API example. The thing worth noting is that 'Event_Loop.cs' and 'Actions.cs' are not editable, they are regenerated at build time. The code in those classes are meant to be like a library in that they are not touched, however they are provided in the users program for debug-ability, test-ability and understanding of the API. Those classes are generated from one or more R2ML reaction rules.

If we choose not to consider the generated code for a moment, the difference in what the user has to code and what they do not can be observed in the following table.

| | Without API | With API |
|---|---|---|
| Window1.cs IL Instructions | 177 | 54 |
| Window1.cs Lines of Code | 27 | 8 |

'Window1.cs' has 123 more IL instructions and 19 more Lines of code in the example not using our API. When coding the example without the API those 19 lines of code that were required were difficult to write because the example required recursion. The code that was added to 'Window1.cs' in the 'without API' example could be reused in a future program, however it is not as flexible as our API as our API uses reflection to interact with methods and properties. In this comparison example reflection was not necessary as disabling a UI control is provided by the 'FrameworkElement' class which all UI components should inherit from. Reflection would add to the lines of code of the 'Without API' example however it would not add lines of code to the 'With API' example as it is using reflection.

### 5.1.1 Performance

If we want this API to be adopted by users it is important that the API does not hinder performance to a level that is obvious to users. The performance of our API initially raised concern because the API appeared to have a delay of about two seconds between a user triggering an event and an action occurring. This was due to the fact that we were outputting information to the developer console which caused performance issues. To get a decent comparison we compiled both samples with the 'no debugging' option enabled. This made the two examples very similar, with the API version running at similar speeds apart from the occasional delay of up to one second. The reason for a delay sometimes with the API version is that an item has to be parsed twice by the rule engine to be expired. If an event is expired on first parse there is only a 100 millisecond delay but if it is expired on the second parse then it has to wait for the 100 milliseconds and then a parse after that to occur. Note that the 100 milliseconds is a chosen default expiry time to give to primitive events. To replicate that behaviour compile the API version of the comparison projects with the 'no debug' option enabled. To simulate a slow (up to 1 second response) click on the print button (making sure this is the only thing clicked). To simulate a quick response click on the textbox then click on the print button. The computer we used for this observation was running 'Windows Vista Ultimate x64' with two gigabytes of 'DDR ram'. The processor was an 'AMD X2 4400+'.

When building the API we chose to implement the functionality of the rule engine, which processes and expires events, in a way that was functionally complete but not optimized for event expiry. This was due to time considerations defined in the projects scope. We are confident that it would be possible to better thread a solution so that when events expire they are expired immediately rather than waiting for a parse of the rule engine. This would enable the currently slow event handling to be done at the same speed that the fast event handling is done.

From the performance tests it is clear that we need better handling of event expiry, however we do believe that the API shows signs that it can be implemented in a way that addresses performance concerns.

### 5.1.2 Comparison reflection

It is also worth considering that our API offers more functionality than is being tested in this example in that it can handle composite events and more advanced querying. Unfortunately those are not as easy to compare to a standard C# application.

Looking back at the comparison it showed that the code generation provided by our API helps to reduce the code that the developer needs to create. This is almost certainly advantageous to developers providing that the creation of a reaction rule in our API is simpler than writing the recursive and potentially reflective code. When writing the examples we found it quicker and simpler to use our API. We also found that it was challenging to write the additional code in the 'without API' example in a way that would likely be easily reconfigured if a change in requirements occurred. With our API we believe this to be provided.

## 5.2 Design Validation

To continue with the validation of our Research it is not only important to compare our research to alternatives but also to compare it against our initial architectural goals. This enables us to realise what areas require additional research, what areas were impractical to implement and what we have managed to implement.

At the beginning of this thesis it was mentioned that we wanted to:

- Not impede existing Visual Studio debugging ability
- Raise event abstraction to provide easier ways to implement Composite Events.
- Lower coupling between user interface and UI events
- Facilitate tidy coding practices
- Facilitate an increase in parallel development of UI
- Close the development gap between user requirements and UI programming.
- Achieve standardization and language Independence
- Allow users to bypass the Event Abstraction API

In the following sections we investigate how we have addressed the fore-mentioned design parameters.

## 5.3 Debugging

Retaining debug-ability is important because users learning to use our library need to be able to see what is causing exceptions in their code.

To retain debug-ability we have provided the debugging symbols with all of the API assemblies. This enables debugging for API code.

The code that is generated is also debug-able however there is an inconvenience of having debug points cleared on generated code when the generation takes place. It is possible however to place debug points in that code once the application is running. While this is an inconvenience there should be no reason to debug it if the library code is running correctly. It will also be possible to rectify this inconvenience in a future release; however it was not deemed as a core feature and hence was regarded as outside the scope of a functional prototype.

## 5.4 Composite Events

Composite events were introduced as a requirement of this research so that users of our resulting API could have an extensible way of declaring high level events. That functionality has been delivered and proven in the tutorial examples provided with the source code of the API (Wilburn, 2007).

While we have managed to implement composite events in a way that we consider to be extensible there is an interesting design feature that we consider to be a drawback to the API's extensibility. This drawback is demonstrated in the example following:

If a triple click is a composite event then a quadruple click can't be four 'clicks' or two 'double clicks' it must be a 'triple click' then a 'single click'. If 'triple click' was not a registered composite event, but 'double click' was then a quadruple click would be made up of two 'double clicks'. That scenario demonstrates how if a definition of quadruple click existed and triple click was suddenly defined, breaking changes would have taken place.

While the drawback of having inter rule dependency can hinder rule independence and hence extensibility, this problem is not considered a major concern because the rule editor could scan for breaking changes in rules. This could be addressed in a future release, however it was deemed functionality that is outside of the scope of this thesis as the developer has the ability to avoid this type of mistake through their own checking.

## 5.5 Coupling Improvements

When considering coupling, the main design concern for this thesis is the GUI. The GUI is most important in that it is the most likely to require change. Often GUI's suffer from a process of wiring their event logic into the presenter. We have managed to create a design that moves coupling away from a user's GUI by exploiting event bubbling offered by the .NET framework. This exploitation allows for no declared event handling in the GUI. The following diagrams compare a possible top down MVP implementation with an MVP like design using our API.

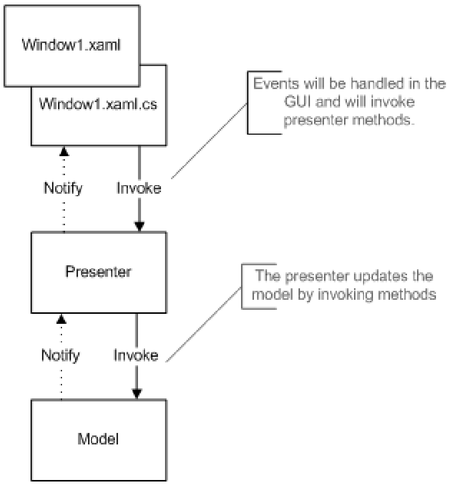

Window1.xaml
Window1.xaml.cs
Events will be handled in the GUI and will invoke presenter methods.
Notify
Invoke
Presenter
The presenter updates the model by invoking methods
Notify
Invoke
Model

Figure 32 – Tiered MVP based design

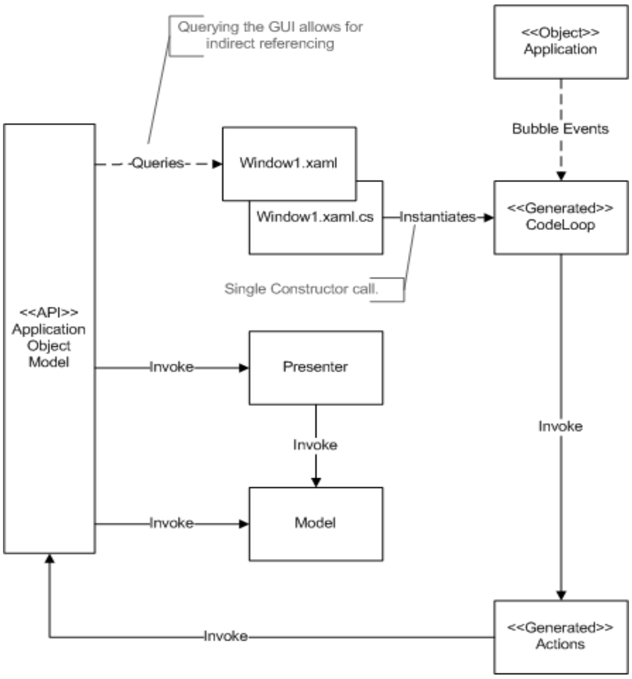


Figure 33 - MVP based API approach

In Figure 32 we can see a tiered MVP approach. That is one possible way of implementing MVP. The ‘Notify’ arrows signify weak references which are in this case events. The ‘Invoke’ arrows represent method calls. In Figure 33 we can see what would appear to be a more elaborate diagram. This is due to the inclusion of the Application Object which is a framework feature for executable applications. There has also been the inclusion of the Application Object Model part of the API to show a full interaction of how invocation and event parsing is facilitated. It is worth pointing out that there are less notify arrows on Figure 33 because a combination of the .NET frameworks event bubbling strategies and reflection allows for the generated code to subscribe to events. It is also worth pointing out that the diagrams do not highlight the fact that there is almost certainly more code in the ‘window1.xaml.cs’ in the first diagram than the second. This is due to the removal of code from ‘window1.xaml.cs’ (apart from a single constructor call). Code instead has been moved to ‘Actions.cs’ shown in the bottom right corner of the second diagram. Actions normally would be in the ‘code behind’ of the GUI and probably directly reference GUI elements. The code in ‘Actions.cs’ doesn’t directly reference any GUI elements because it attempts a query over the Application Object Model and then attempts to invoke a method over the resulting set.

The use cases are a good demonstration of what sample application code looks like when using our API. Our API has provided a slave GUI. The GUI is comprised of XAML which is used for display only. The GUI has an instance of ‘Event Loop’ that is the only non display line of code in the GUI (which is injected by the plug-in). Rules are translated into code automatically. Rules are applied to the GUI by applying functions to resulting queries over the GUI. As queries are used there are only weak references to GUI elements. Example B.3 available in Appendix B shows the code that updates the user interface via a query.

## 5.6 Facilitate tidy coding practices

At the start of this project it was mentioned that there was a preference to encourage tidy coding practices where possible. We have facilitated this possibility by:

- Reducing additional code to one extra line in the GUI.
- Generated code files are put into a separate folder. This enables a clear separation of generated code and non generated code.
- Code in GUI is likely to be less. This is because the GUI provides less functionality due to event handling being relocated to a separate class. In the comparison provided at the start of this chapter we reduced/relocated the GUI code leaving 33% of the code previously in the GUI.

## 5.7 Parallel development of UI

A developer can create a GUI and rule for event to action mapping in parallel. They can then put the two together in the same project and our plug-in will generate all the code required to wire the two together. The only dependency between event to action mappings and the user interface is the search terms used by queries. It is recommended that a list of tags is compiled. This list will in theory enable full parallel development.

## 5.8 User Requirements and Code

Requirements analysis and implementation can be considered as two sides of a ravine where the resulting software solution is a bridge between a user’s goals and practicality. In this thesis it was deemed advantageous to attempt to provide a higher level of abstraction for events than what the machine offers so that it is at a level that is more comparable with user requirements. If software truly is the bridge between user requirements and how machines operate then it only makes sense to attempt to make these two areas easier to connect by making them work at a similar level of abstraction. We propose composite events provide this abstraction.

## 5.9 Standardization and Language Independence

While our solution does have a large dependency on the Visual Studio plug-in model, it does not have a large dependency on C#. As the code from our resulting API is compiled into the Common Language Runtime, it would be possible to allow the execution of our C# code from Visual Basic .NET or any other .NET language. The only extension our solution would require is that the one line of code that is injected into the GUI is translated to the .NET language of choice. Some popular languages that have third party ports to .NET are: Ruby and Python. While our solution can support many languages it would require a lot of work to translate the Visual Studio plug-ins to another IDE and language base. .NET provided a standard for generating code in a language that can easily be linked with other languages. Moving our solution from .NET would be challenging in that our solution was designed to exploit .NET libraries and abilities. It would still be possible to translate our API to another language like Java. The main issue would be finding Java versions of those libraries or libraries of similar functionality.

In this research it was decided to use R2ML as the syntax to store rules. This language was chosen on purpose so that event to action mapping rules could be independent of programming language. This would mean that if a solution was implemented in Java, that rules could require minimal or no changing. Adopting a potential standard, R2ML could allow for the use of third party tools in the future. R2ML also provides a benefit in that it is an evolving language that is yet to get to version 1.0. This enables us to provide suggestions as to features that could be useful additions to the language.

## 5.10 Freedom of use

Users of our API have the ability to bypass the use of rules to handle events by explicitly handling themselves. While in most cases that would not be encouraged, it is important to not force users into an 'all or nothing' approach. Due to the nature of event bubbling in .NET a user could explicitly handle an event and still allow it to bubble. They could also take the opposite approach and not allow it to bubble.

# Chapter 6

# Conclusion

In this section we look at the conclusion, in which our results are discussed. We also recommend avenues for future investigation.

## 6.1 Conclusion

During the course of this research we delved into many areas, however there were three main areas our research can be divided into:

- Development of an event algebra with supporting infrastructure.
- Implementation of the execution of event to action rules
- Development of an application object model

The three main areas previously listed were implemented in a way that they were able to be independently implemented. This made the implementation easier, and it allows us to assess their status more easily.

Looking at the event algebra aspect of this thesis, we have managed to implement a proof of concept, however the overwhelming number of unwanted events makes an event filtering mechanism desirable. We implemented the ability to do ‘sequential’ and ‘or’ events. We believe that in the future more complex event types may require implementation; this option is available as there is a complex event class that can be sub-classed. From the previous chapter it has become apparent that our event processing engine for processing composite events could use more optimization.

The execution of event to action rules refers to the code that is generated from the R2ML rules. The main limitation the code generation project faces is that the implementation is currently confined to use in Visual Studio. The use of Visual Studio was provided as a convenience so that separate applications were not required to generate rules. It would be possible in the future to make the code generation not dependent on Visual Studio.

The development of an application object model helped with decoupling dependencies on the user interface for action handling code. It has been proved in the comparison in chapter 5 that we could have used alternate ways from an application object model, however querying provides better extensibility and is likely to be closer to being functionally complete.

In the next sections we look at areas in which this research could be extended beyond this thesis.

## 6.2 Future Work

Due to time constraints and scope not everything could be implemented or researched. This section outlines ideas for further extension on this thesis.

### 6.2.1 Research

Currently there is support for only one set of rules per project. While it is a good idea to have the same behaviour/rules across a whole project there could be a requirement in the future to override rules when applied to a specific user interface. Use cases for different sets of rules could be: accessibility; internationalization and languages; and different experience levels. Accessibility could enable extended help for the area the user requires help with. Internationalization could take cultural habits into account when dealing with users. Different users with different experience levels often provide a challenge for developers as they have very different expectations from software and require different styles of interaction. It could be worth considering object oriented techniques such as overriding when researching the idea of multiple ECA rule lists.

Extending this research by taking event filtering into account could be interesting. During this research it was discovered that the volume of framework events was at a level that it either required filtering or wildcard operators in the event algebra. In the implementation of this research the API did implement XPath which supports wildcard operators; however the rule engine that processes these events does not. The purpose of wildcards in the definition of complex events would provide a means of filtering unwanted events between event sequences. Wildcards in the event algebra could provide a challenge in implementation due to the events life time expiry. Scanning of events would happen in parallel, requiring threads. That would complicate the issue of scanning an event list looking for certain events. With the introduction of wild card operators, logic such as ‘NOT’ would appear to make sense to use to represent an event sequence not being interrupted by a certain event. It would however also making event expiry harder to work out because events currently are calculated based on the event algebra sets they belong to. Unfortunately this method would not work as any event could be in the middle of a wild carded complex event meaning that all events would possibly have a dynamic expiry depending on all events preceding it.

An interesting realization that was made in chapter 4 was that XPath is not valid to the official URI standard. We also found that R2ML requires the use of a URI for event sources. We propose that a URI could be a reference to a singleton set of resources rather than just a single resource. Some web URL's currently provide query based links on websites that are conformant to the URI standard. This would be an area of research that could be investigated further.

### 6.2.2 Software Improvements

Currently there is a requirement for the Standard, Professional or Team System version of Visual Studio to use the code generation features of the API. In a future edition we would like to make the code generation a stand-alone executable that could be used by another IDE that may not support a plug-in model.

The complex event processing engine should be built so that it can be more optimal. Avenues of investigation could be parallel processing and algorithm improvement.

Investigation into transformations between R2ML, RuleML and Windows Workflow Rules could be beneficial in providing standards access to a wider range of people.

### 6.2.3 Additional Features

An idea for extension could be to implement this research so that it facilitates website applications as well as desktop applications. We believe this to be entirely possible with existing technologies such as JavaScript, which could be used for the client side. The challenge that the web provides is that it will require an approach that takes into account the distributed nature of events on the web. When referring to the distributed nature of the web we are referring to the concept of client server applications where the user provides events on the client end causing updates on the server. Although it is difficult to speculate how this would work without research, we can state that technologies such as AJAX (Asynchronous JavaScript and XML) will make implementation possible by providing a means of updating a webpage without refreshing the page. This is due to the Asynchronous nature of AJAX. Asynchronous events are events that are non-blocking to the main execution of a program. Normal events in .NET are synchronous.

The Event Algebra could be extended to allow for the importing of R2ML production rules. These production rules could represent the complex event definitions. This would allow for more choice as users could code or script composite event rules.

The R2ML library is only using version 0.4. This library could use a 'tidy up' by researching how to better use schema-to-code tools. This library could also benefit from updating to version 0.5 or the latest version of R2ML.

A feature that was initially planned in this research but was discarded, was the idea of visually drawing rules. Due to our querying approach, drawing rules on a GUI builder would probably have almost no additional meaning for our rule editor. With that said it could be worth drawing existing rules and seeing how they would apply to a current GUI over the GUI designer window. There could possibly be no advantage to this visualization, but this would have to be investigated further.

# Bibliography

Avery, J. (2007, 12 07). *What is dot net*. Retrieved from OnDotNet: http://www.ondotnet.com/pub/a/dotnet/2005/09/06/what-is-dotnet.html

Badica, C., Giurca, A., & Wagner, G. *Using Rules and R2ML for Modeling Negotiation Mechanisms in E-Commerce Agent Systems.*

Barghouti, N., & Kaiser, G. (1991). Scaling Up Rule-Based Software-Based Enviroments. *3rd European Software Engineering Conf* (pp. 380-395). Berlin: Springer-Verlag.

BARGHOUTI, N., & KAISER, G. (1991). SCALING UP RULE-BASED SOFTWARE-DEVELOPMENT ENVIRONMENTS. *3RD EUROPEAN SOFTWARE ENGINEERING CONF* (pp. 380-395). SPRINGER-VERLAG BERLIN, BERLIN .

Berners-Lee, T. (2000). Retrieved March 14, 2007, from W3C: http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html

Berstel, B., Bonnard, P., Bry, F., Eckert, M., & Patranjan, P.-L. (2007). *Reactive Rules on the Web.*

Biazetti, A., & Gajda, K. (n.d.). *Achieving complex event processing with Active Correlation Technology.* Retrieved February 10, 2008, from IBM DeveloperWorks: http://www.ibm.com/developerworks/autonomic/library/ac-acact/index.html

Boley, H., & Tabet, S. (n.d.). *RuleML HomePage*. Retrieved March 10, 2007, from RuleML: http://www.ruleml.org

Boley, H., Grosof, B., & Tabet, S. (2005, May 13). *RuleML Tutorial*. Retrieved May 23, 2007, from RuleML: http://www.ruleml.org/papers/tutorial-ruleml-20050513.html

Booth, D., Haas, H., & McCabe, F. (2004, February 11). *Web Services Architecture.* Retrieved 08 21, 2007, from W3C: http://www.w3.org/TR/ws-arch/

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. (2006, August 16). *Extensible Markup Language (XML) 1.0.* Retrieved August 21, 2007, from W3C: http://www.w3.org/TR/REC-xml/

Castle Stronghold. (2004, December 27). *CastleProject*. Retrieved January 4, 2008, from http://www.castleproject.org/

Dabek, F., Zeldovich, N., Kaashoek, F., Mazieres, D., & Morris, R. *Event-driven Programming for Robust Software.* MIT Laboratory for Computer Science.

Data & Object Factory. (2001). *Gang of Four (GOF)*. Retrieved August 3, 2007, from Design Patterns in C# and VB.NET: http://www.dofactory.com/Patterns/Patterns.aspx

Del.icio.us. (n.d.). Retrieved from Del.icio.us Social Bookmarking: http://del.icio.us/

Deransart, P., Ed-Dbali, A., & Cervoni, L. (1996). *Prolog: The Standard: Reference Manual.* Springer.

Digg Inc. (n.d.). Retrieved April 20, 2007, from Digg: http://www.digg.com

ECMA General Assembly. (1996, November). Retrieved January 23, 2008, from ECMAScript Language Specification: http://www.ecma-international.org/publications/standards/Ecma-262.htm

Eder, J., Kappel, G., & Schre, M. (1992). *Coupling and Cohesion in Object-Oriented Systems.* Klagenfurt: Institut fur Informatik, Universit at Klagenfurt.

Fallside, D. C., & Walmsley, P. (2004, October 28). *XML Schema Part 0.* Retrieved September 9, 2007, from W3C: http://www.w3.org/TR/xmlschema-0/

Ferg, S. (2006, January). *Event Driven Paradigm.* Retrieved April 16, 2007, from SourceForge: http://eventdrivenpgm.sourceforge.net/event_driven_programming.pdf

Forgy, C. (1982). *Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem.*

Fowler, M. (2004, September 8). *Closure*. Retrieved January 28, 2008, from Martin Fowler: http://martinfowler.com/bliki/Closure.html

Fowler, M. (2006, July 18). *Passive View Design Pattern*. Retrieved January 28, 2008, from Martin Fowler: http://martinfowler.com/eaaDev/PassiveScreen.html

Galiano, P. (2007, May). *VSSDK Assist*. Retrieved August 21, 2007, from Codeplex: http://www.codeplex.com/vssdkassist

Gamma, E. (1995). Design Patterns: Elements of Reusable Object Oriented Software. *Addison-Wesley Professional Computing Series* .

Giurca, A., & Wagner, G. (n.d.). Retrieved January 15, 2008, from R2ML -- The Rewerse I1 Rule Makrup Lanugage | Working Group I1: http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=R2ML

Goldberg, A., & Robson, D. *SmallTalk-80 The language and its implementation.*

Hejlsberg, A. (2001). *Patent No. 6,185,728 B1.* United States of America.

JavaDude. (2006, June 05). *Understanding the Generated Code*. Retrieved January 28, 2008, from JavaDude: http://javadude.com/vaj/general/understandingGeneratedCode.html

JBoss. (2007, February 27). *Drools.NET*. Retrieved January 05, 2008, from Source Forge: https://sourceforge.net/projects/drooldotnet/*.*

Jeff Molofee. (2000, January). *Neon Helium Productions*. Retrieved January 6, 2008, from Gamedev.net: http://nehe.gamedev.net

Knolmayer, G., Endl, R., & Pfahrer, M. (2000). Modeling Processes and Workflows by Business Rules. *Business Process Management, Models, Techniques, and Empircal Studies* , 16-29.

L. Chamberland IBM. (1998). *IBM Visual Age for Java.* Retrieved January 16, 2008, from IBM Systems Journal Volume 37 Number 3: http://www.research.ibm.com/journal/sj/373/chamberland.html

Lin, P. (2004, June 04). *Dingo Schema Compiler*. Retrieved January 08, 2008, from Source Forge: http://dingo.sourceforge.net/

Liu, Y., Gorton, I., & Le, V. K. (2007). A Configurable Event Correlation Architecture for Adaptive J2EE Applications. *Australian Software Engineering Conference (ASWEC'07)*, (pp. 49-58).

Martin Webb. (1999, June 26). *JavaScript Guidelines and Best Practice*. Retrieved July 24, 2007, from Internet Related Technologies: http://parallel.ru/docs/Internet/IRT/articles/js169/index.htm

Microsoft. (n.d.). *About the W3C Document Object Model*. Retrieved January 13, 2008, from Microsoft Developer Network (MSDN): http://msdn2.microsoft.com/en-us/library/ms533043(VS.85).aspx#unknown_25

Microsoft Corporation. (2007, November). The Manycore Shift. Seattle, Washington, USA.

Microsoft. (2006, June). *ECMA-335: Common Language Infrastructure (CLI)* . Retrieved May 28, 2008, from ECMA international: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-335.pdf

Microsoft. (n.d.). *MSDN Forums*. Retrieved January 21, 2008, from MSDN Forums: http://forums.microsoft.com/msdn

Microsoft. (2003). *System.EventHandler.* Retrieved March 20, 2007, from Microsoft Developer Network (MSDN): http://msdn2.microsoft.com/en-us/library/system.eventhandler.aspx

Microsoft. (2004, March). *The Avalon Input System*. Retrieved June 21, 2007, from Microsoft Developer Network: http://msdn2.microsoft.com/en-us/library/aa480167.aspx

Microsoft. (2007). *XAML Overview*. Retrieved May 28, 2007, from MSDN: http://msdn.microsoft.com/en-us/library/ms752059.aspx

Motakis, I., & Zaniolo, C. (1995). *motakis95composite.pdf.* Retrieved April 21, 2007, from Citeseer: http://citeseer.ist.psu.edu/cache/papers/cs/16961/ftp:zSzzSzftp.cs.ucla.eduzSzpubzSzzanioloz Szdood95zSzmotakis-zaniolo.pdf/motakis95composite.pdf

Myers, B. A. (1998). A Brief History of Human Computer Interaction Technology. *ACM interactions. Vol. 5, no. 2* , 44-54.

Myers, B. A. (1994). *User Interface Software Tools.* Pittsburgh: Carnegie Mellon University.

NDepend. (2005). *Metrics.* Retrieved February 15, 2008, from NDepend: http://www.ndepend.com

Norman, W., & Paton. (1998). *Active Rules in Database Systems.* Springer.

Novell. (2004). *Main Page*. Retrieved February 2, 2008, from Mono: http://www.mono-project.com/Main_Page

Object Managment Group. (2004, May 7). *UML 2.0 Superstructure.* Retrieved January 3, 2008, from Object Management Group: http://www.omg.org/docs/formal/05-07-04.pdf

Opera. (1996). *Home Page*. Retrieved May 26, 2008, from Opera Browser: http://www.opera.com/

*OWL Web Ontology Language*. (2004, February 10). Retrieved March 10, 2007, from W3C: http://www.w3.org/TR/owl-features/

Potel, M. (1996). Retrieved from http://www.wildcrest.com/Potel/Portfolio/mvp.pdf

Proctor, M., Neale, M., Lin, P., & Frandsen, M. (n.d.). *Drools Documentation*. Retrieved January 11, 2008, from JBoss: http://downloads.jboss.com/drools/docs/3.0.6/html_single/index.html#d0e209

Refsnes Data. (n.d.). *W3C XML Activites*. Retrieved January 8, 2008, from W3Schools: http://www.w3schools.com/w3c/w3c_xml.asp

Rewerse. (2004, March 1). Retrieved January 15, 2008, from Rewerse - Reasoning on the web: http://rewerse.net/

Rewerse. (2006, March 13). *URML - A UML rule modeling language*. Retrieved August 21, 2007, from Rewerse: Reasoning on the web: http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=URML

Rewerse Working Group. (2006, March 12). *Strelka - An URML based visual rule modeling tool.* Retrieved June 29, 2007, from Rewerse: http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/10

Rivieres, J., & Wiegand, J. (2004). Eclipse: A platform for integrating development tools. *IBM systems journal, Vol 43, No2* , 371-383.

Rubiolo, D., Meier, J., Jezierski, E., & Mackman, A. (n.d.). *Microsoft Papers.* Retrieved January 3, 2008, from Microsoft Developer Network: http://docs.msdnaa.net/ark_new3.0/cd3/content/Papers%5Cnxp2.doc

S. White IBM. (n.d.). *Introduction to BPMN.* Retrieved February 12, 2008, from Buisness Process Management Initiative: http://www.bpmn.org/Documents/Introduction%20to%20BPMN.pdf

Scott W. Ambler Ambysoft Inc. (n.d.). *UML 2 Communication Diagrams*. Retrieved June 19, 2007, from Agile Modelling: http://www.agilemodeling.com/artifacts/communicationDiagram.htm

Sun Microsystems. (n.d.). *Designing Enterprise Applications with the J2EE platform*. Retrieved 05 2007, from http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/images/app-archa2.gif

Sun Microsystems. (2007, 06 28). *Java Docs.* Retrieved from http://java.sun.com/j2se/1.4.2/docs/api/java/awt/event/ActionListener.html

Sun Microsystems. (n.d.). *Presentation Frameworks.* Retrieved April 15, 2007, from Patterns & OO Design: http://www.sun.com/software/sunone/docs/arch/chapter8.pdf

Taveter, K., & Wagner, G. (2001). *Agent-Oriented Enterprise Modeling Based on Business Rules.* Eindhoven: Eindhoven Unviersity of Technology.

URI Planning Interest Group, W3C/IETF. (2001, September 21). *URIs, URLs, and URNs: Classifications and recommendations*. Retrieved January 29, 2008, from W3C: http://www.w3.org/TR/uri-clarification/

W3C. (2004, April 7). *Document Object Model (DOM)*. Retrieved January 12, 2008, from W3C: http://www.w3.org/DOM/

W3C. (2001, April 5). *Metadata at W3C*. Retrieved January 24, 2008, from W3C: http://www.w3.org/Metadata/

W3C. (2004, February 10). *RDF/XML Syntax Specification*. Retrieved April 14, 2007, from W3C: http://www.w3.org/TR/rdf-syntax-grammar/

W3C. (2004, May 21). *SWRL: A Semantic Web Rule Lanugae Combining OWL and RuleML*. Retrieved April 19, 2007, from W3C: http://www.w3.org/Submission/SWRL/

W3C. (2008, May 22). *W3C Semantic Web Activity*. Retrieved May 28, 2008, from W3C: http://www.w3.org/2001/sw/

Walnes, J. (2006, September 16). *The Power of Closures in C# 2.0*. Retrieved January 28, 2008, from Joe Walnes: http://joe.truemesh.com/blog//000390.html

Wikimedia. (n.d.). Retrieved April 20, 2007, from Wikipedia: http://www.wikipedia.org/

Wilburn, R. (2007, November 10). *Event Abstraction API for .NET*. Retrieved February 12, 2008, from CodePlex: http://www.codeplex.com/eventabstractionapi

XUnit. (2007, August). *XUnit*. Retrieved December 10, 2007, from Codeplex: http://www.codeplex.com/xunit

Y. Yongqing EuropeLoan Bank. (2002, 07 25). Rock Climbing and Extreme Programming. Brussels, Belgium.

Zimmer, D., & Unland, R. (1999). *On the Semantics of Complex Events in Active Database Management Systems.*

# Glossary

| | |
|---|---|
| Action | An action is what is executed after an event is triggered. An event will fire an action if it is a handled event and the code within the action will be executed. |
| Add-in | This is a type of Visual Studio SDK project. This project has a medium amount of functionality in terms of interacting with Visual Studio. An add-in can start when Visual Studio starts where as a VSPackage starts once it is required by default. |
| API | Application programming interface. An API is an interface for letting a program communicate with another program. |
| CLR | Common Language Runtime is the core runtime engine in the .NET framework for executing applications. The CLR is aimed at providing a platform for multiple languages to interact, for example C# and Visual Basic.NET can be compiled together into MSIL which is executed by the CLR. |
| C#/C-Sharp | C# is an ECMA standard as is the platform it runs on, the .NET framework. C-Sharp can be denoted by C#. C# is a programming language with a syntax compared to Java and C++. |
| DLL | Dynamic Link Library – This is a form of reusable assembly that can be accessed via COM or via .NET depending on what DLL type it is and what options it uses. |
| ECA | Rules Event Condition Action Rules are a type of Reaction Rule. The rule should be executed when the event occurs that matches the event part of the ECA rule. The condition must be met to execute the action once the event occurs. |
| Event | An event is what is triggered when a user interacts with the system or the system receives an internal message/event. In this research we focus on User events. |
| IDE | Integrated Development Environment. Some IDE's mentioned in this thesis are Net Beans, Eclipse and Visual Studio |
| Metadata | Meta data is user contributed comments or data that can be appended to system data to provide context. A popular use of Meta data is to rate user submissions on a website. This enables inappropriate user submitted content to be removed or minimized. |
| MSIL | Microsoft Intermediate Language is the language that was made to provide interoperability between various .NET languages allowing parts an executable to be comprised of many languages. The CLR runs the MSIL. |

| | |
|---|---|
| Production Rule | A production rule is a rule that provides conditions to data. If a data value changes then production rules should be re-evaluated to check if they should execute. |
| Reaction Rule | A reaction rule is a rule that is triggered when an event it is mapped to is fired. A reaction rule can sometimes have a condition to decide if it wants to execute an action. |
| R2ML | Rewerse Rule Mark-up Language. This is an XML language that provides Reaction Rules, Production rules, Integrity and Derivation rules. |
| VSPackage | A VSPackage is a project type provided by the VSSDK. A VSPackage provides the highest level of functional influence over Visual Studio. Add-ins and Macros are also available as alternatives. |
| VSSDK | Visual Studio Software Developer Kit. This is a set of tools and examples provide by Microsoft, the makers of Visual Studio. |
| WPF | Windows Presentation Foundation is one of the four libraries introduced in .NET 3.0 . It provides the use of XAML and new Event handling strategies. |
| XAML/XOML | ‘Extensible Application Mark-up Language’ and ‘Extensible Object Mark-up Language’ are two syntaxes for XML that are provided with .NET 3. XAML is used in the design of user interfaces and XOML is used in the rule engine provide by .NET. |
| XML | Extensible Mark-up Language. XML is a W3C recommendation, providing a text based storage standard that provides an open platform for information storage that is programming language independent. |
| XPath | XPath is a W3C recommendation. XPath extends the XML model to provide a way to traverse XML trees. |
| .NET framework | This is a software component that is an extension for Microsoft Windows Operating Systems. It provides a platform for languages compiled under CLR to execute. The .NET part of the name is pronounced as “Dot-Net”. |

# Appendix A

# Use Cases

In this section, URML diagrams are given to visualize the use case scenarios provided in chapter 3.

### *Use Case 1: On button1.Click, disable textbox2*

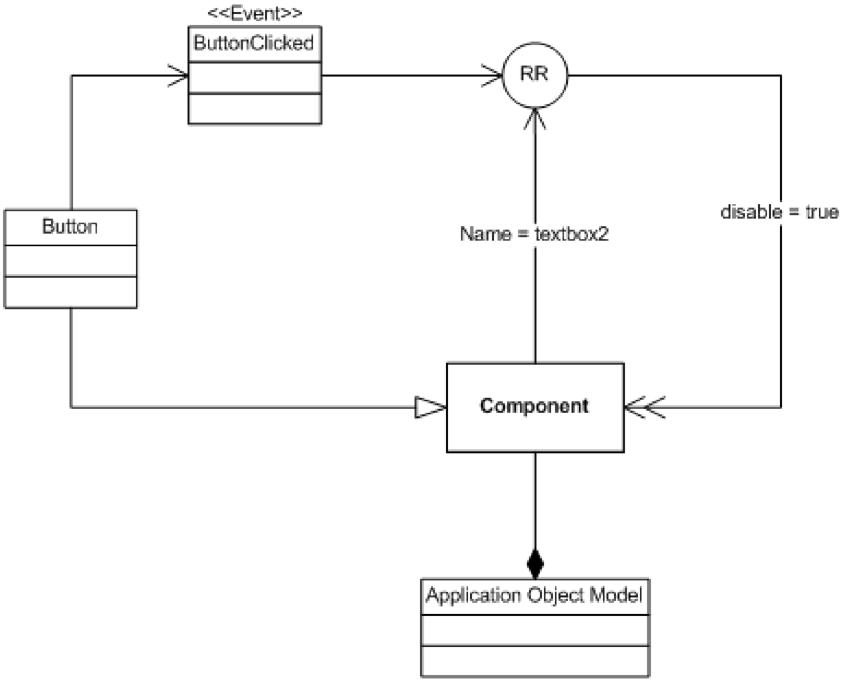


Figure 34 – Use Case 1

### *Use Case 2: On button.Click, disable multiple UI components*

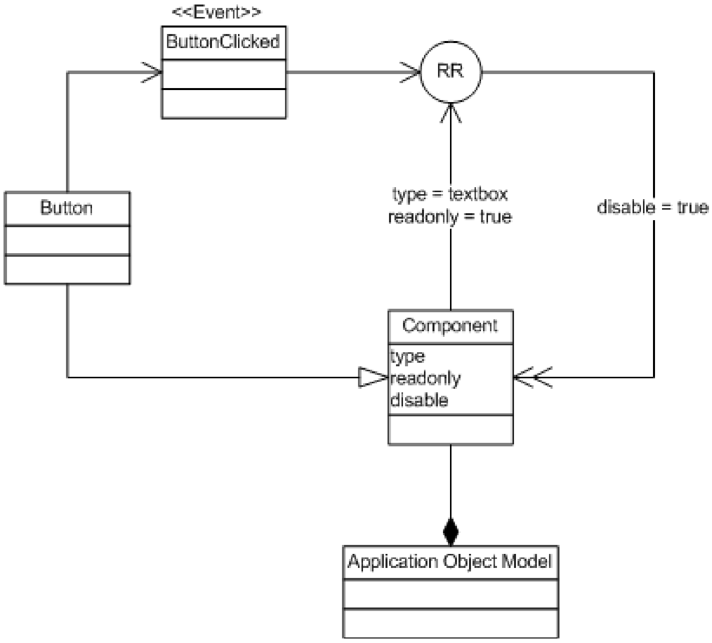


Figure 35 - Use Case 2

### *Use Case 3:* Printing is triggered, disabling GUI components that trigger printing

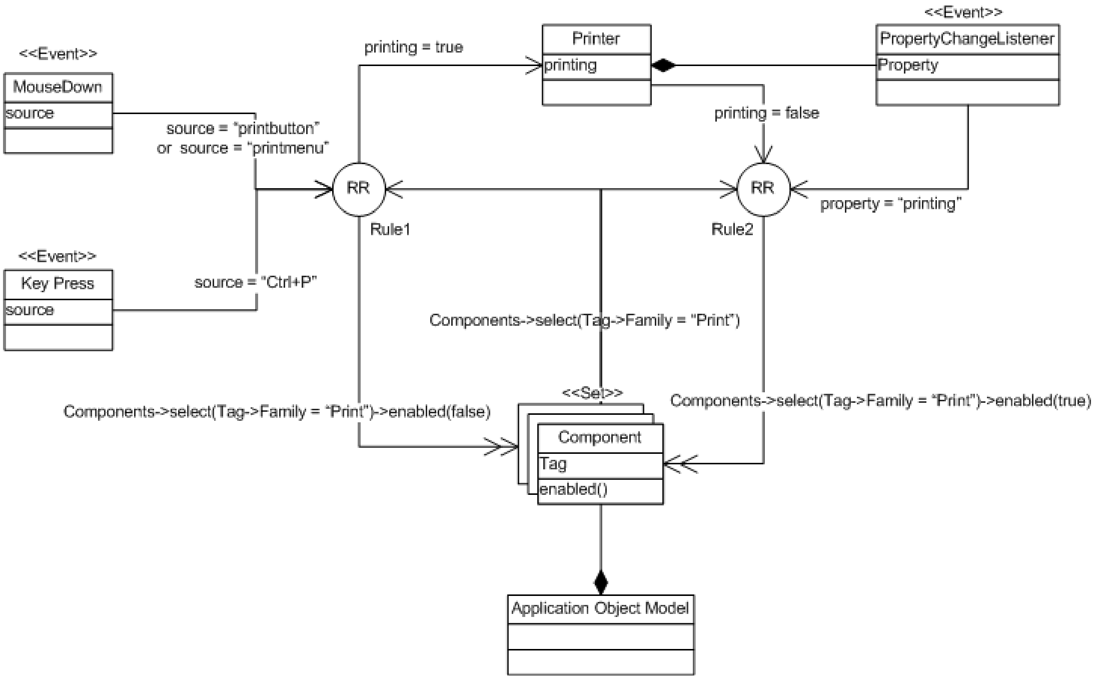


Figure 36 - Use Case 3

The figure previously shown, demonstrates a use case for an 'OR' composite event.

### *Use Case 4: A primitive event becomes a composite event and is executed.*

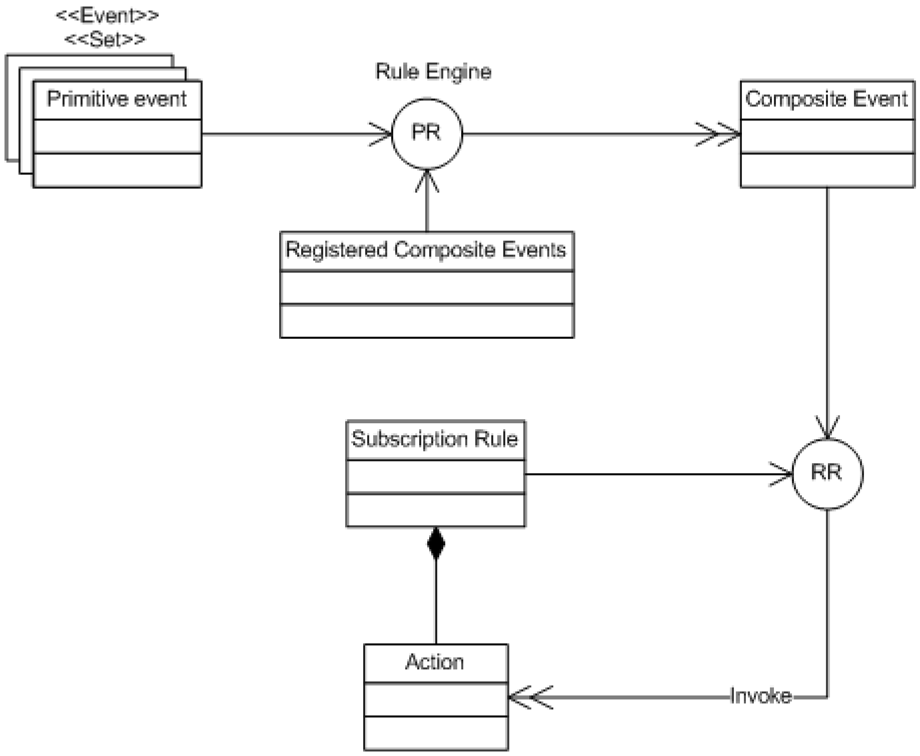


Figure 37 - Use Case 4

Use case 4 shows a set of input primitive events that are processed by the rule engine. The rule engine uses registered composite events which describe what a composite event is comprised of. These registered events allow for the discovery of composite events from the input set of primitive events. Composite events are then check to see if they have been subscribed too, which is denoted by subscription rule. A subscription rule provides the information to map the composite events invocation to an action. It is possible as denoted in this usecase that the action itself is part of the subscription rule.

If we assume that the primitive events are two right mouse button clicks and that a composite event of ‘double right mouse click’ was registered as two single right mouse button clicks in a sequence then we could deduce from this use case diagram that the composite event would be ‘double right mouse click’. If we then assumed that there was a subscription rule for that composite event, then that action would be invoked. For use case 4 that action would be loading context based help.

## *Use Case 5:* Operating System Boot Loader

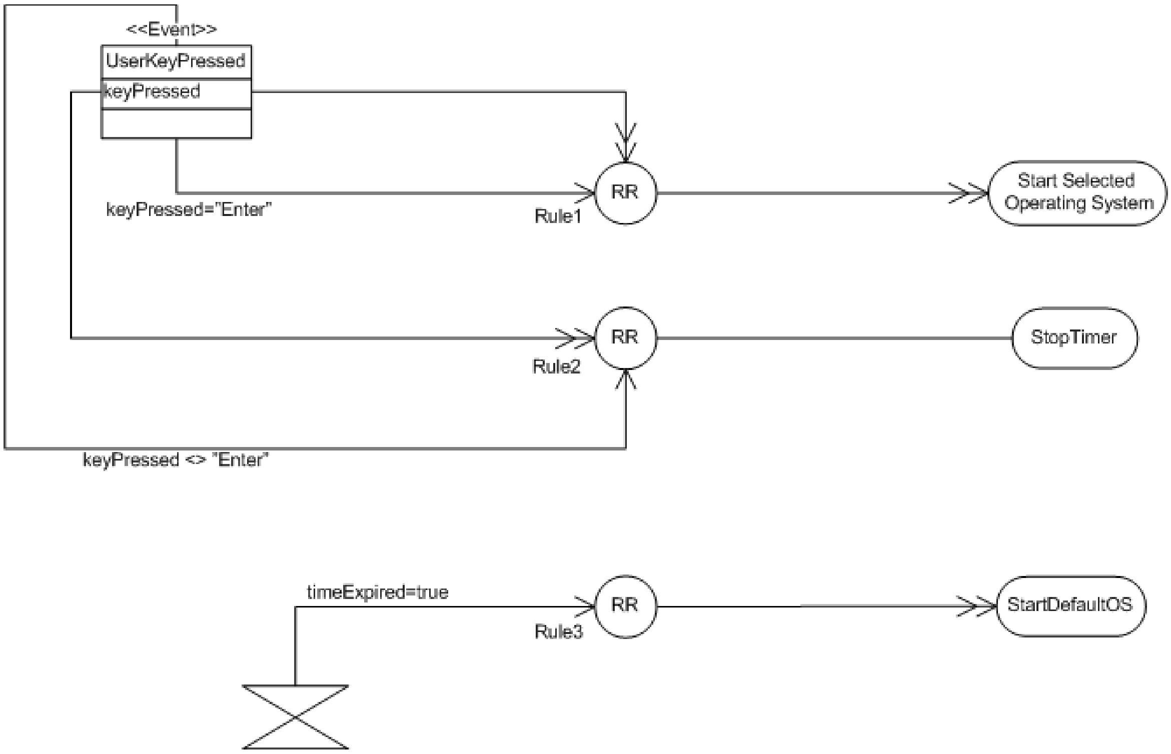


Figure 38 - Use Case 5

# Appendix B

# Source Code

## Example B.1 Message Pump/Event Loop in C++

The example demonstrates a centralized approach to event handling.

```
switch (uMsg)                 // Check For Windows Messages
{
   case WM_ACTIVATE:          // Watch For Window Activate Message
   {
      if (!HIWORD(wParam))    // Check Minimization State
      {
         active=TRUE;         // Program Is Active
      }
      else
      {
         active=FALSE;        // Program Is No Longer Active
      }

      return 0;               // Return To The Message Loop
   }

   case WM_SYSCOMMAND:        // Intercept System Commands
   {
      switch (wParam)         // Check System Calls
      {
         case SC_SCREENSAVE:  // Screensaver Trying To Start?
         case SC_MONITORPOWER: // Monitor Trying To Enter Powersave?
         return 0;            // Prevent From Happening
      }
      break;                  // Exit
   }

   case WM_CLOSE:             // Did We Receive A Close Message?
   {
      PostQuitMessage(0);     // Send A Quit Message
      return 0;               // Jump Back
   }

   case WM_KEYDOWN:           // Is A Key Being Held Down?
   {
      keys[wParam] = TRUE;    // If So, Mark It As TRUE
      return 0;               // Jump Back
   }

}                                               (Jeff Molofee, 2000)
```

## Example B.2 Plug-in Generated Event Loop

This example contains the Event Loop that is generated by the Visual Studio Addin when the user initiates the build process. The code has been edited for readability. The code in this example was taken from the 101 project provided with the API.

```
using System;
using System.Text;
using System.Collections.Generic;
using System.Windows;
using System.Reflection;
using System.IO;
using System.Windows.Threading;
using System.Threading;
using System.Windows.Forms;

using EventAbstractionAPI.RuleEngine;
using EventAbstractionAPI.ApplicationObjectModel;
using R2ML;

namespace WindowsApplication
{
   /// <summary>This class is autogenerated.</summary>

   public class EventLoop
   {
       /// <summary>Constructs the event to Action mappings
       /// Run this constructor when you want to hook events up to your UI.
       /// Or alternatively you do it manually by using the constructor with no
       /// arguments.
       /// That will require you to map events manually to the OnEvent method.</summary>

       public EventLoop()
       {
          // if there is an event we do not want to handle we must handle it
          // so that we can know the sequence/history of events.
          //for every event that is possible in the eventmanager we need to subscribe to
          //it.Initialize the model that stores rules so that we can pull parts of rules
          //to use them later.

          Assembly assembly = Assembly.GetExecutingAssembly();
          string [] resNames = assembly.GetManifestResourceNames();
          Stream stream = assembly.GetManifestResourceStream
                                              ("WindowsApplication.ReactionRules.r2ml");

          string contents;
          using (StreamReader reader = new StreamReader(stream))
          {
             contents = reader.ReadToEnd();
          }

          R2ML.RuleFactory.Instance.LoadRules(contents);

          _ruleList = R2ML.RuleFactory.Instance.ruleBase;

          _appObjectModel = new ApplicationObjectModel(true);

          _actions = new Actions();

          //attach the event listener to the event buffer, and handle it in this class.
          _eventBuffer.RaiseEvent +=new Fire(OnExpiredEvent);

          //TODO: for now we are only using 1 window in the application.
          //This will not always be the case so this will need to be updated with a
          // foreach loop.
          //This has been left as is for simplicity in testing the concept.
          Window w = (Window)System.Windows.Application.Current.Windows[0];

          //currently for simplicity of testing and demonstration and due to event spam
          // we have limited the event loop to catching clicks of buttons.
          w.AddHandler(System.Windows.Controls.Button.ClickEvent, new
          RoutedEventHandler(OnEvent), true);
          //This foreach loop below is how we can subscribe to all events.
```

```
        //This has been commented due to current problems with event spam.

        //foreach (RoutedEvent re in EventManager.GetRoutedEvents())
        //{
        //      w.AddHandler(re, new RoutedEventHandler(OnEvent), true);
       //}
    }

    public void OnEvent(object sender, EventArgs evt)
    {
        //this is put in for debug purposes only. This removes the problem of the
        // event continually reoccuring as you debug. If you debug the window loses
        // focus and that event will be fired causing debugging issues. There are a
        // lot of filtered events which have been replaced with .. due their
        //illegability.
        if (evt.GetType().Name != "MouseMove" && ..)
          {
            ICapturedEvent capturedEvent = new CapturedEvent();
            capturedEvent.EventArgumentsList.Add(evt);

            //try get the event name. It might be input args or it could be routed
            // event args.
            try
            {
                   capturedEvent.EventName = ((System.Windows.Input.InputEventArgs)

                                                             (evt)).RoutedEvent.Name;
            }
            catch (Exception)
            {
               try
               {
                  capturedEvent.EventName = ((RoutedEventArgs)evt).RoutedEvent.Name;
               }
               catch (Exception)
               {
                  throw;
               }
            }

            capturedEvent.Sender = sender;
            _eventBuffer.Receive(capturedEvent);
        }
    }

public void OnExpiredEvent(List<ICapturedEvent> expiredEvents)
{

    //This is declared here so that there are not multiple declarations in the switch
    //statement, which upsets the compiler.
    ReactionRule rule;

     foreach (ICapturedEvent capturedEvent in expiredEvents)
     {
        string elementName;
        FrameworkElement element;

        //we must work out what events we want and then generate a switch statement
        // and only log other events.
        switch(capturedEvent.EventName + capturedEvent.ComplexName)
        {
            case "Click":

                       //rulenumber = 1

               //get the rule object.
               rule = _ruleList.ruleSet.getReactionRules()[1 -1];

               elementName = "";

               try
               {

                   element = ((FrameworkElement)((RoutedEventArgs)capturedEvent

                                           .EventArgumentsList[0]).OriginalSource);
```

```
                      //This is a way of avoiding UI threading issues
                      //As the UI is on another thread we must use the Dispatcher.Invoke

                      element.Dispatcher.Invoke(DispatcherPriority.Normal,
                      (ThreadStart)delegate
                                             {
                                                elementName = element.Name;
                                             });

                  }
                  catch(Exception)
                  {

                  }

                  //if sender is in eca1's xpath query.
                  if (_appObjectModel.satisfies(rule.triggeringEvent.
                                            messageEventExpression.sender ,elementName))
                  {
                      //if precondition is satisfied
                      //if(rule.conditions are satisfied)
                      _actions.method1(capturedEvent);
                  }

                         break;

                  default:
                  break;
              }
           }
        }

        #region Properties

        public EventAlgebra EventBuffer
        {
           get { return _eventBuffer; }
           set { _eventBuffer = value; }
        }

        #endregion

        #region Member Variables

          EventAlgebra _eventBuffer = new EventAlgebra();
          ApplicationObjectModel _appObjectModel;
          RuleBase _ruleList;
          Actions _actions;

        #endregion
    }
}
```

### Example B.3 Plug-in generated Actions

This example is taken from the 101 example provided with the API. This code can be found in Actions.cs.

```
using System;
using System.Text;
using System.Collections.Generic;
using System.Windows;
using System.Windows.Forms;

using EventAbstractionAPI.ApplicationObjectModel;
using R2ML;
using EventAbstractionAPI.RuleEngine;
using Calltype =
EventAbstractionAPI.ApplicationObjectModel.ApplicationObjectModel.CallType;


namespace WindowsApplication
{
    /// <summary>This class is autogenerated.</summary>

    public class Actions
    {
        /// <summary> Contains a list of actions that the event loop can call.
        /// That will require you to map events manually to the OnEvent method.
        /// If you need to add code here please use the following commented XML tags to
        /// add code.
        /// //<preaction>
        /// ...    (your code to happen before the generated code)
        /// //</preaction>
        /// ...    (The Generated code will go here)
        ///    //<postaction>
        /// ...    (your code to happen after the generated code)
        ///    //</postaction>
        ///</summary>

        public void method1(ICapturedEvent evt)
        {

            AOM.Execute("xpath://*[@Name='button1']",ApplicationObjectModel.CallType
                                                                  .Property,"Width",10);

        }

        //Member Variables
        ApplicationObjectModel AOM = new ApplicationObjectModel(true);
    }
}
```