Model Driven Development using Software Matrix

BY

TILAK PATEL

Thesis submitted in partial fulfillment of the requirements for the

Degree of Master of Science in Computer Science

ADVISOR:

DR. JAMES FAWCETT

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

SYRACUSE UNIVERSITY

December 2007

Syracuse, New York

Abstract

Effective recycling of software assets has been long standing goal of software industry. However only few organizations in software industry could successfully implement practice of effective software reuse to improve productivity, to improve robustness and quality and to reduce development and maintenance cost. Computer Aided Software Engineering tools looked promising for rapid application development but failed to create a revolution as they are not designed for software salvage process. By salvage we mean lifting of significant blocks of existing systems and trying and inserting them into newly developed systems.

Many general and sophisticated Computer Aided Software Engineering tools have been developed previously. In this research, our goal is to develop a prototype Computer Aided Software Engineering tool for Model-Driven Software Development. The prototype tool developed in this tool is unique as compared to previous approaches as we intent to show that effective salvage of software assets could be effectively integrated into Computer Aided Software Engineering tools. Using our tool, we intend to show how the Software Matrix Framework [9] can be used for rapid application development and in some cases, for instant software development by salvaging software assets. The Software Matrix is a framework that supports and promotes the reuse of components. In particular it is a runtime infrastructure into which individual pieces of application can plug. We intend to explore more of the Software Matrix concept using our tool.

Table of Contents

Table of Figures

Chapter 1 Introduction

1.1. Research Problem

Effective recycling of the software assets of an organization has been a long-standing goal of the software engineering discipline. Salvage is a form of recycling that does not insist that there are no changes in the recycled component, as is the case with reuse, but rather, that large pieces of software can be used again with very few, very controlled, changes.

Consider a hypothetical Radar System Development firm. It develops customized Radar Systems based on the client requirements. The company has developed many radar systems in past projects, based on an original design, but responsive to current customer requirements. Now suppose they get a new client that asks for set of features where some are almost similar to systems developed in past and some of the features require new code development. To build this new radar system, large pieces of software from the existing radar systems software - with

pieces not necessarily designed for reuse – can be extracted. But modifying and integrating these pieces is often a challenging task, as extracted code may have many dependencies with code in the system in which it was originally embedded. Adding a new feature set and code pieces adds more problems and the whole process becomes complex. In this scenario, we can't take help from CASE tools to generate code to speed up development as integration with old blocks is very complex, and there are no hooks in the existing codebase for those tools. If we could convert our software assets into pluggable blocks and we could plug them using some CASE tool, it would make the development process much more productive.

In this research, our goal is to develop a prototype Computer Aided Software Engineering tool for software development predicated on the making software salvage effective. Using our tool, we intend to show how the Software Matrix Framework [9] can be used for rapid application development and in some cases, for instant software development by salvaging software assets. We intend to explore more of the Software Matrix concept using our tool.

1.2. Prior Approaches

CASE (Computer Aided Software Engineering) [5] tools are designed to support rapid application development [6] [8]. There are many Computer Aided Software Engineering tools generating code based on the UML [1] diagrams, supporting

different versions of UML. There are also many different approaches that encourage software salvage.

1.2.1 Software Reuse

1.2.1.1 Object Oriented Reuse

One of the prime motivations behind object orientation is reuse. Object oriented techniques like encapsulation, inheritance and polymorphism are, indeed, an effective mechanism for reuse. Object oriented languages like C++ and Ada introduced the concept of class libraries, where libraries consist of thousands of classes and functions. They promoted reuse by providing access to large sets of functionality residing inside them through inheritance and composition.

These ideas created a spark, but however, failed to burn down the woods. They provided good solutions for some specific domains like data structure libraries, C++ standard library, for example, or user interface frameworks like the Microsoft Foundation Classes. However most business organizations failed to effectively reuse their own pre-existing software assets through creation and consumption of object oriented libraries. One reason is that the objects are necessary, but not sufficient, for the salvage model. Objects are collections of data and methods to operate on the data. Salvage units are something different from objects; they are entities deployed, versioned and mostly having the scale of subsystems, much larger than objects, e.g., Objects are too granular to be used as salvage units.

1.2.1.2 Component Oriented Reuse

As objects are too granular, the next approach was component oriented systems [7]. A software component is a system element offering a predefined service and able to communicate with other components [12]. A software component should satisfy most of the following properties: Multiple-use, Non-context-specific, Composable with other components, Encapsulated and a unit of independent deployment and versioning [2] [3]. Components are different from objects. Following the object oriented technique, one's mental model of the imagined or actual object is the main focus while developing software, that is, an object is an abstraction of some small set of functionality, crafted specifically for that representation. In contrast, software based on components, much like electronic or mechanical components, should be developed by composition of prefabricated components. Physically, objects and components also differ in granularity, components are larger, coarser. Some of the popular component specifications are Microsoft's COM [13] ("a platform-independent, distributed, object-oriented system for creating binary software components that can interact [14]"), Enterprise Java Beans [15] ("a component architecture for the development and deployment of object-oriented, distributed, enterprise-level applications") [16], OMG's CORBA (Common Object Request Broker Architecture) ("a language-independent object model and specification for a distributed applications development environment").

The common objective of all these technologies is to develop independently deployable components to promote reuse. Another feature of component oriented technology is to promote an architecture where a system, composed of components, can be upgraded by changing components without rebuilding the whole system. This is a very important feature if the system is composed of millions of lines of code. The component-based software construction technology is beneficial as compared to object oriented approaches in terms of reuse, however component oriented approach has not solved, itself, solved the problem of supporting software salvage.

1.2.2 Model Driven Architecture

The heart of Computer Aided Software Engineering tools lies in MDA (Model Driven Architecture™, approach launched by the Object Management Group [20]). The MDA approach defines system functionality using a platform-independent model (PIM) which is translated to one or more platform-specific models, using an appropriate domain-specific language or a General Purpose Language like C++, Java, or C#. The MDA model is related to multiple standards, including the Unified Modeling Language (UML), the Meta-Object Facility (MOF), XML Metadata Interchange (XMI), Enterprise Distributed Object Computing (EDOC), the Software Process Engineering Metamodel (SPEM), and the Common Warehouse Metamodel (CWM).

1.2.2.1 Rational Rose [17]

As per it's product website, the IBM Rational Rose family is IBM's classic UML modeling and model-driven development [27] "solution". It is one currently popular UML tool in the CASE market. It offers a modeling environment that supports code generation from models in Ada, ANSI C++, C++, CORBA, Java™/J2EE™, Visual C++® and Visual Basic®.

1.2.2.2 Sparx Systems' Enterprise Architect [18]

According to the product website, Enterprise Architect combines the power of the UML 2.1 specification with a high performance, intuitive interface, to bring advanced modeling to the desktop, and to the complete development and implementation team. It can be easily integrated with some of the popular IDEs (Integrated Development Environments) like Microsoft Visual Studio® [22] and Eclipse (open source universal IDE) [19].

There are many other CASE tools similar to two discussed above. They work very well in some cases e.g. database centric applications and when developing an empty framework or developing applications from scratch. Some of the CASE tools remain "point solutions" [4]. They support multiple standards and tend to be more general and broader in scope. But while doing that they fail to solve problem of software asset salvage discussed earlier. They fail when we try to

integrate a newly designed framework with existing assets - they tend to complicate the process and consume time fixing integration issues.

1.3. Our Approach

From the above discussion, component oriented technology looks promising if there is a framework which promotes and actively supports it. However they are designed to solve generic industry problems. They fail to support vertical applications, where developing a huge framework for some specific market niche is impractical from a return on investment point of view. CASE tools are useful, but they failed to create a revolution as was expected early in their development. They are good for creating empty frameworks or database centric applications, but, however, while combining existing code and new development, they fail as they are not designed for the software salvage process. They generally follow models like UML which are actually language independent ways of expressing the design of a system. We actually want the exact opposite of that, a way to represent the system using existing software assets.

We already have a framework developed by Riddhiman Ghosh and Dr. James Fawcett [9] which promotes software salvage. The basic idea of the Software Matrix framework is to create software Integrated Circuits (ICs) - using a hardware analogy - to build a system composed of standard reusable blocks. Hardware designers have standard IC components so they already have basic

building blocks. If we can convert our software assets into such pluggable assets, it would be a very valuable approach indeed.

Now if we can integrate this idea with a CASE tool, we could build an entire system graphically given all required assets available in the form of pluggable blocks. In this research we intent to build such a software architectural tool to lay out the structures of systems graphically and later populate them with existing software assets, where available, and develop the remaining blocks. This approach promises reduced development and maintenance cost and quicker time-to-market for product families composed of the same base code blocks. Code blocks - offering new required functionality - become matrix cells and can be plugged into any system. Following the matrix-based architecture allows very fast development by registering existing cells and putting in empty, place-holder, cells for future blocks and developing their functionality later.

Of course a system composed using the Software Matrix framework inherits all of its advantages. Although the Software Matrix is not component oriented technology, its structure supports dynamic construction where blocks can be upgraded or replaced at run time. This Software Matrix framework is flexible enough that testing can be made more dynamic and systems can be tested at run time by replacing working blocks from the system. The Software Matrix framework also provides a set of abstractions that we can build in almost any language [10].

Chapter 2 Tools and Architecture used

This chapter focuses on the technologies used in this research. We describe the Software Matrix technology, developed in research activities at Syracuse University, and extensions specific to this research. We also describe the designer used to construct diagrams and generate code according to the Software Matrix framework.

2.1 Software Matrix

The Software Matrix framework is an architecture developed by previous research work by Dr. James Fawcett, Riddhiman Ghosh [9], Anirudha Krishna [11] and Vijay Appadurai [10]. The Software Matrix framework is used for building cells that can be reused with no transformation cost. Software Matrix has significant advantage over traditional component based architectures in that it simplifies the way existing cells can be integrated into new projects. Applications composed using Software Matrix technology can add cells dynamically at run time. Software Matrix is a collection of code blocks called cells which are the building blocks out

of which applications are built. The cells communicate with each other by means of XML messages. The interaction between cells is governed by a special cell called mediator.

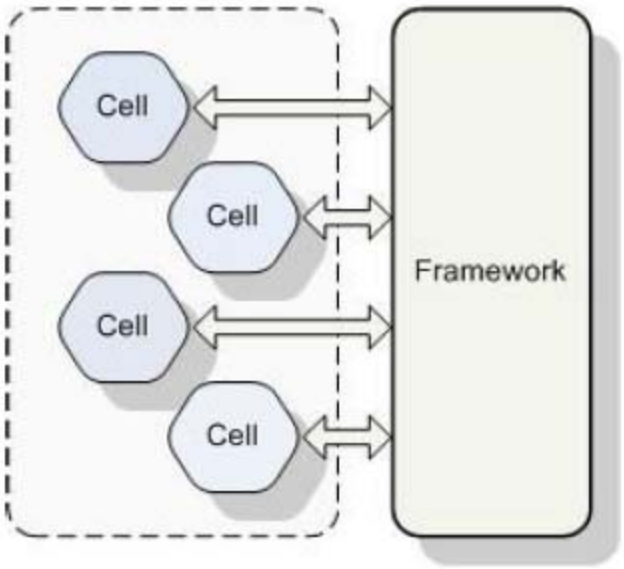


Figure 1 : The Software Matrix

The framework is composed of well defined components called Cells and a Mediator based communication framework that allows interaction between the cells. This combination allows loose coupling between components to such an extent that Cells can be integrated just by placing their files into a pre defined folder.

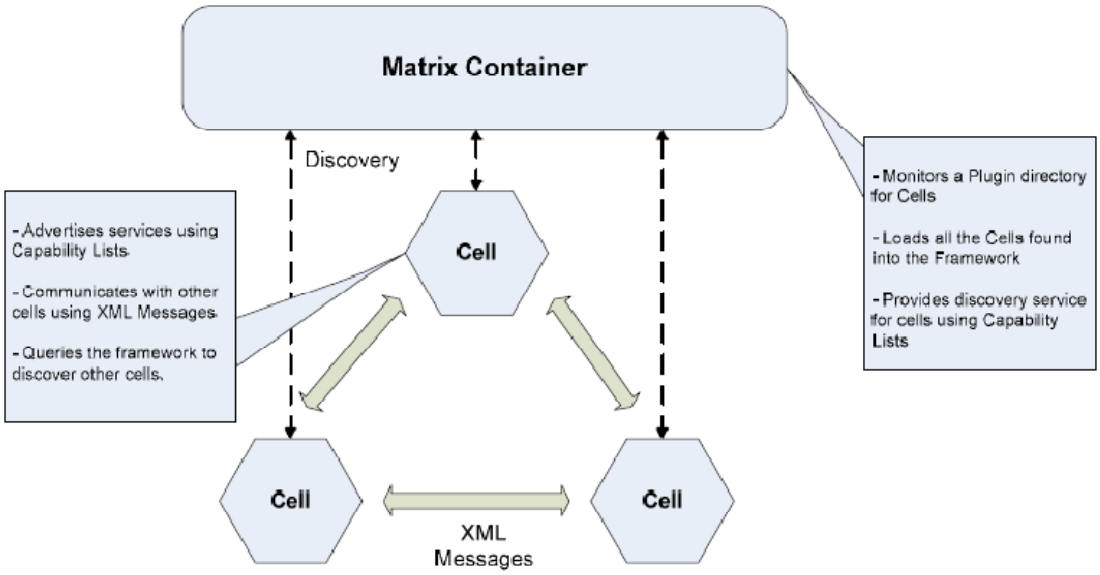


Figure 2 : The Software Matrix Structure

The Matrix has three important components – the Cell, Message Passing infrastructure, and dynamic construction. The Matrix Framework contains a Loader which monitors a Plug-in directory. The Plug-in directory contains all the cells that need to be loaded into the Matrix. The Loader loads all the cells in the Plug-in directory into the Software Matrix and registers all the cells into the matrix. Each cell contains a capability list describing the functionality it provides. The mediator contains information about the capabilities of each cell in the Matrix. The mediator provides discovery service for particular functionality using this information. All communication between the cells is in the form of XML Messages.

2.1.1 Matrix Cell

The Cell is the basic building block of the Software Matrix. It is similar to classes in object oriented design but is larger in scope, usually demonstrating component level functionality. Applications are built by combining cells that possess the desired characteristics.

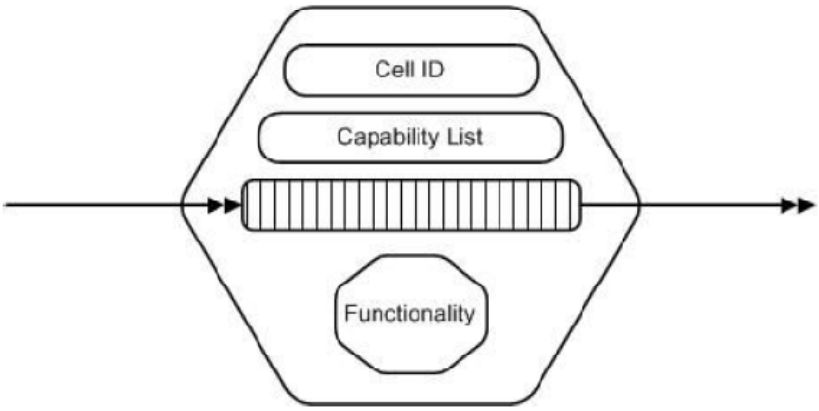


Figure 3 : The Matrix Cell Structure

Since all components are Cells, they can function either as Servers or as Executives. A server Cell provides certain services. Requests are collected, processed and replies are sent. Executive Cells control the flow of the Application by making requests to servers, and displaying results. Most UI components are implemented as Executive Cells. Each Matrix Cells consists of certain common components which are discussed below:

2.1.2 Cell ID:

The Cell ID is a Globally Unique Identifier (GUID) called cell ID. Each cell instance is discovered by the Matrix using this unique ID and is used to aid delivery of Request and Reply messages to the Cell.

2.1.3 Capability List:

Each cell advertises the services it provides using a Capability List. Each service provided by a cell is given a name called a Message Type. The Capability List of each cell is the collection of all the Message Types it can process. To make a request of this service, a message, with the desired Message Type, is sent to the Cell.

2.1.4 Message Queues:

All communication between the cells is through message passing. Cells can send and receive messages from multiple locations at different times. A queue is used to buffer incoming requests so that no message will be lost when the Cell is busy processing a previous request.

2.1.5 Functionality

Functionality is the processing that is performed when a cell receives a request. The parameters contained in a request message are used to process the request and

generate a response. For example, a FileInfo Cell might get a matrix.fileInfoCell.getFileList message with some directory path and in response the cell constructs a message consist of names of all files in a given directory.

All Cells follow a common protocol that specifies how to register and unregister with the Matrix, advertise their capabilities, send and accept messages and communicate with other Cells.

2.2 Design

We shall now discuss the implementation details of the Software Matrix. The Software Matrix was developed for this research using C++. Matrix Applications are composed of cells. The cells were written as dynamically loaded libraries (dll). The Cells need to follow some conventions to effortlessly integrate into applications. Each Cell is required to derive from ICell class. This class consists of functions that implement basic cell characteristics like Message Queue and Message Dispatcher Thread.

2.2.1 Loader

The Loader is responsible for identifying and loading all valid Cells into the Software Matrix. The Loader is also responsible for registering each Cell with other Cells according to the diagram constructed on designer canvas. Using a

model diagram as the basis for cell connection is one of the contributions of this research. We actually use a hybrid approach where cell connections are established by both the model and by mediator lookups. We are currently pursuing a totally model-driven connection scheme.

2.2.2 Matrix

The Matrix consists of Capability Lists of service provided by all the cells registered into the Software Matrix. It provides the discovery service for the cells in the Software Matrix. Given a Capability, it returns the address of the cell providing the service. The Matrix also ensures that cell execution begins only after all the cells are loaded into the Matrix. This is accomplished by using a Blocking Queue which queues up the messages sent before all the cells are loaded into the Matrix.

2.2.3 ICell

Each Cell must derive from the ICell class. It defines the protocol that should be followed by all the cells in the Software Matrix. All concrete cells that are derived from the ICell class are provided default implementations for the functions in the ICell class. Some important functions which perform message passing and so aid in communication between Cells are discussed below:

- `PostMsg(std::string& msg)-` This method is used to enqueue the request XML message from other cells into the queue for processing.

- `AddResponse(std::string& guid,std::string& resp)-` This method is used to enqueue the response XML message from the server cell into the queue.

- `dispatch(std::string& msg)-` This method specifies what is to be done in response to the various messages that arrive to this cell. It appropriately delegates calls to the code that actually implements the functionality required to handle a particular message type.

Cells use the ExecObject class to assist in packaging of arguments, return values and other information regarding requests and responses between cooperating Cells in the Software Matrix.

Sample Request Message:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Message>
    <RequestID>3434508d-ba23-4edc-b6d0-52cb686cdfad</RequestID>
    <type>Request</type>
    <SyncWaitFlag>diabled</SyncWaitFlag>
    <networkMsg>disabled</networkMsg>
    <RequestCellGUID>7af71d4d-4cd1-46ea-a5ed-01c4e53fb912</RequestCellGUID>
    <ResponseCellGUID>D66F09EB-8641-4c78-B9D0-FC5D1CE07B76</ResponseCellGUID>
  - <MessageType>
      <Name>matrix.cell2.readFile</Name>
      <value>YzpcdGVtcFxyZWxhdGlvbi54bWw=</value>
    </MessageType>
  </Message>
```

Figure 4 : readFile message to read c:\temp\test.xml

Sample Response Message:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Message>
    <RequestID>3434508d-ba23-4edc-b6d0-52cb686cdfad</RequestID>
    <type>Response</type>
    <SyncWaitFlag>diabled</SyncWaitFlag>
    <networkMsg>disabled</networkMsg>
    <RequestCellGUID>7af71d4d-4cd1-46ea-a5ed-01c4e53fb912</RequestCellGUID>
    <ResponseCellGUID>5F4C4784-B5DA-45df-AB66-738CEF004557</ResponseCellGUID>
  - <MessageType>
      <Name>matrix.fileInfo.reply</Name>
      <value>PGZpbGVkYXRhPlBHdGxlWE0rQ2p4amFHbHNaRDR5Wm1Wak5HTTVNUzAzTm1ZNUx
        UUXlaR0l0WVRJMk1pMWlPREV6WlRRd01EQTFaV1U4TDJOb2FXeGtQanh3WVhKbGJuUStOe
        md5TUdGbU9EY3Raamd4TXkwME5EWmlMVGc0TjJVdE5qTmlZVEk1WWpRM05qUXdQQzl3WVh
        KbGJuUStDanhqYUdsc1pENDNPREl3WVdZNE55MW1PREV6TFRRME5tSXRPRGczWlMwMk0yS
        mhNamxpTkRjMk5EQThMMk5vYVd4a1BqeHdZWEpsYm5RK2JuVnNiRHd2Y0dGeVpXNTBQZ28
        4WTJocGJHUStPVFUwWW1Fd1pEY3RNVEF3TXkwME1EUmtMV0l6TnpZdE0ySTBaR0ZqWWpjM
        1ltSXpQQzlqYUdsc1pENDhjR0Z5Wlc1MFBqYzRNakJoWmpnM0xXWTRNVE10TkRRMllpMDR
        PRGRsTFRZelltRXlPV0kwTnpZME1Ed3ZjR0Z5Wlc1MFBnbzhZMmhwYkdRK1lXVTBPRGcyT
        kdNdE1HVXhZeTAwWWpBNUxUbGpOVGd0TTJJMFpXVTBPREpoT1RJM1BDOWphR2xzWkQ0OGN
        HRnlaVzUwUGpjNE1qQmhaamczTFdZNE1UTXRORFEyWWkwNE9EZGxMVFl6WW1FeU9XSTBOe
        lkwTUR3dmNHRnlaVzUwUGdvOEwydGxlWE0rPC9maWxlZGF0YT4=</value>
    </MessageType>
  </Message>
```

Figure 5 : reply message with file contents

2.2.4 Serialization/Deserialization:

The data such as arguments in request messages and return values in response messages need to be serialized and deserialized so that the resulting persistent data can be encapsulated in XML messages, passed across various platforms. A base64 encoder/decoder is used to serialize/deserialize the data in the request and response messages.

2.3 Code Generation Framework

Software Matrix is a runtime infrastructure that acts as a substrate into which individual pieces of application can plug and applications can be composed dynamically. In this research, we developed a prototype Computer Aided Software Engineering tool, called the Athena Designer, for software development predicated on making software salvage effective. It has been developed using Windows Presentation Foundation (WPF) [21] framework, developed by Microsoft, to be released as part of Visual Studio 2008.

At design time, a system is seen as a collection of blocks where some blocks are existing pieces and some blocks are going to be developed in the future. For this research we call these blocks Cells. Athena Designer adds design time support and lets users construct an application graphically by decomposing application functionality into Cells, constructing a Cell diagram, defining Cell types based on existing pieces or by putting in empty placeholders and then generating code, at a later time.

Consider a real world application like a Directory Synchronizer, used as an example by Riddhiman Ghosh in his original Software Matrix Research. The Synchronizer allows the content of a directory on one machine to be synchronized with contents of a directory on a remote machine so that we always have the latest

version of a specified set of files on one or both machines. If we think in terms of building blocks of this application, there are four major building blocks:

- User interface block that allows the user to operate the application, view errors, etc.
- Communication block to transfer file data between local and remote machines.
- Policy block that decides which files to transfer, update, overwrite or delete based on pre-defined rules.
- File Management block which does all file and directory operations.

The User interface is the main block which uses services of communication block and policy block directly and policy block uses the service of the File Management block. So a typical Cell Diagram will look something like this:

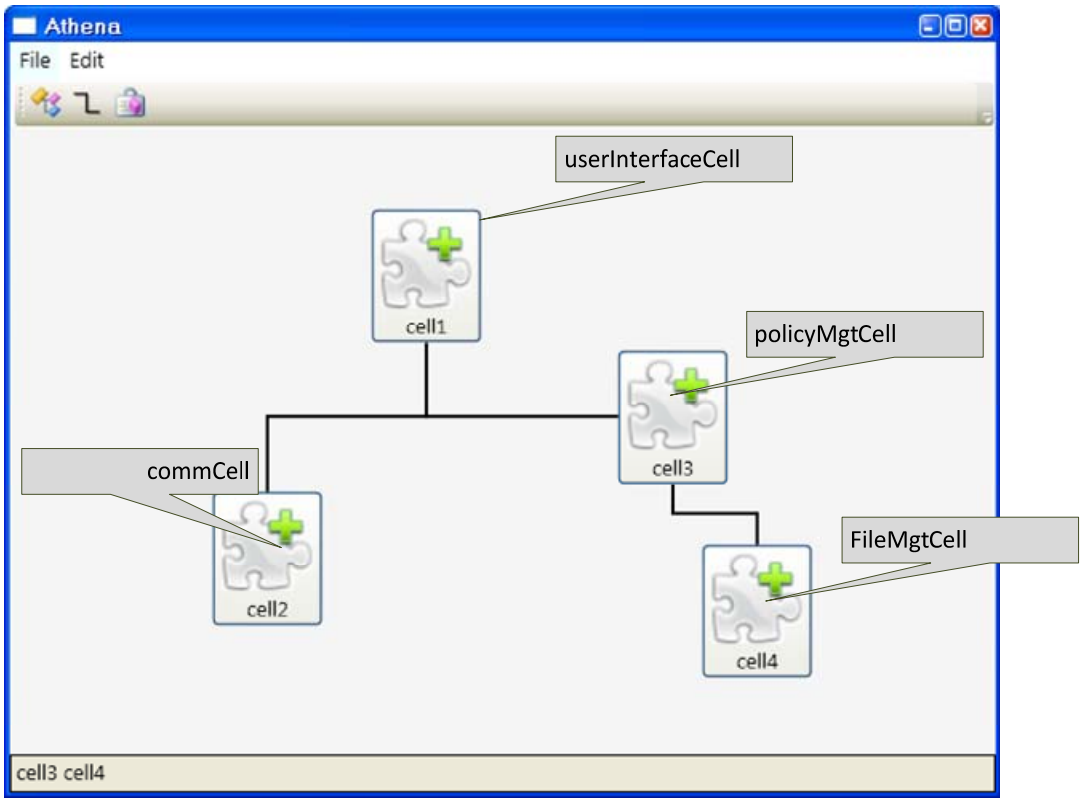


Figure 6 : Cell Diagram for Directory Synchronizer

The File Management block and Communication block are common blocks and might be already developed for some other application. Let's assume that they are available when constructing this Cell Diagram. Policy block is the heart of this application and might not be encountered before so let's assume that it is not available.

For the functionality which is not present at the design time, we put empty placeholder cells and the system can be assembled using existing cells and placeholder cells. Each Cell on diagram represents a piece of code and the connections between Cells show how they will be registered with each other to exchange message and use each other's services.

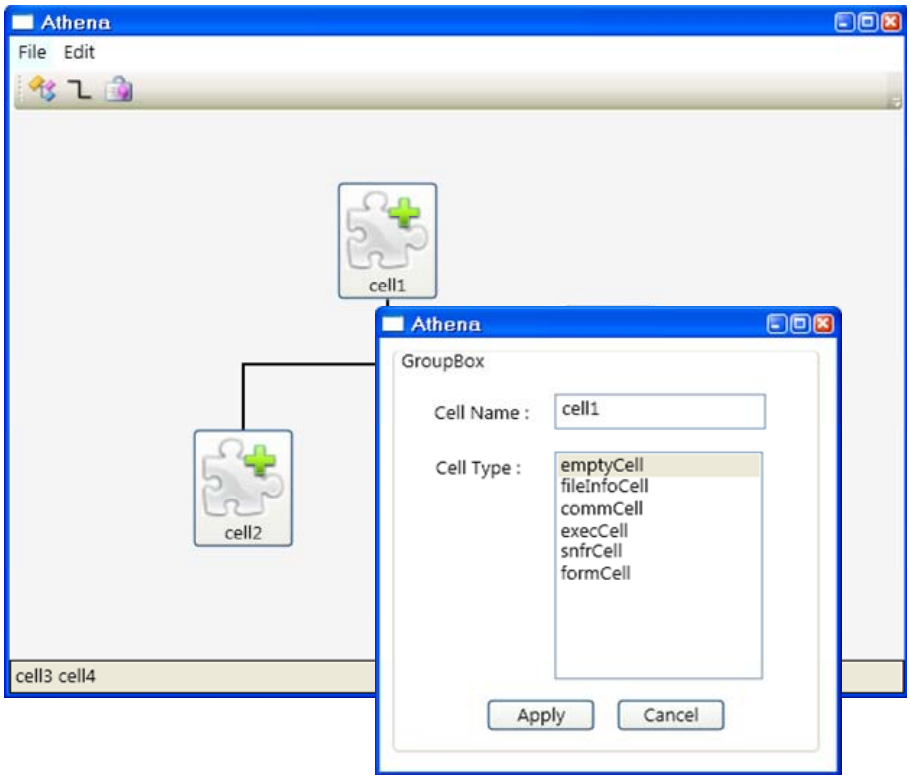
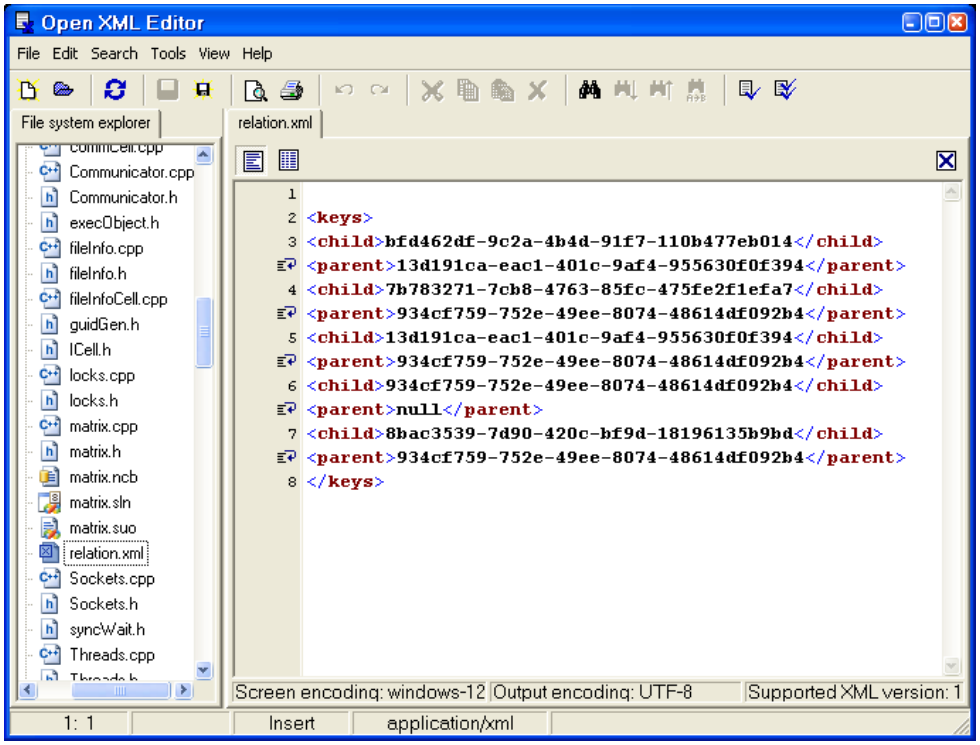


Figure 7 : creating empty placeholder Cell

Based on connections between Cells on Athena Designer canvas, a relation.xml file is generated which has all the information about how Cells should be registered with other Cells.



The Loader reads this relation.xml file and registers Cells accordingly.

For the Cell Diagram, code is generated as a Visual Studio solution file. Each Cell represents the unit of composition and reuse in our system and so for every Cell in the diagram, code is generated as a separate project and has output build type as a DLL file, and is part of the solution file[1].

2.3.1 Building an Application using Athena Designer

This section outlines the concrete steps needed to build system taking advantage of the Software Matrix.

[1] See appendix for more details about project and solution file formats for Visual Studio

In order to enable salvage, pieces of an application are wrapped in a Cell. All Cells conform to the ICell protocol. For example:

FileInfo : public ***ICell*** *{ // code here };*

All Cells have to enter required source and header files information in specific.xml file. It is manifest file which is used by Athena Designer to generate code. All cells are placed inside "filedb" directory.

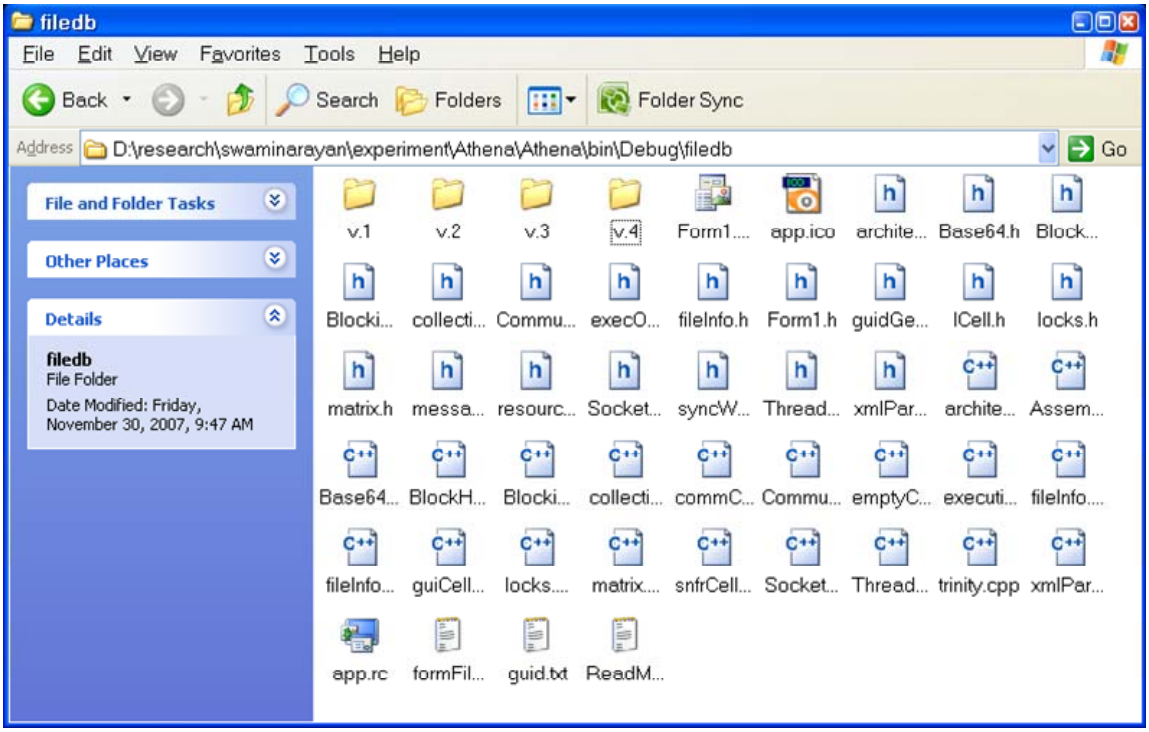


Figure 8 : source and header files in filedb directory

Athena Designer loads Cell profiles at startup by reading specific.xml file.

Next one needs to construct a diagram for the target application using our Athena Designer. Each block in the diagram maps to a particular Cell and the user does the mapping by selecting a block and mapping CellType to Cell profile loaded by Athena Designer. If a particular block is not mapped to any Cell profile, Athena Designer will generate an empty cell as a place holder for that block and the user is free to add functionality later.

After constructing a diagram one can instruct Athena Designer to generate code and Athena Designer generates the application as a Visual Studio Solution[2] [22] and generates project files as part of the main solution for each individual Cell.

[2] Solution file is compatible with Visual Studio 2005 and later versions.

2.4 Extensions to the Software Matrix:

In this section we will discuss the various improvements to the original Software Matrix Framework made by previous developers of this research technology.

2.4.1 Changes made to core framework by Anirudha Krishna

2.4.1.1 Directory watcher for the loader module

Software Matrix loader monitors plugins directory for the Cells and as soon as new Cell is added or existing Cell is modified, it loads new Cell. Earlier framework used polling method and it was modified to use event signaling.

2.4.1.2 Integration of Network Access – the network cell

Software Matrix framework elements were modified to exchange message across computers to build distributed applications. Now message can be addressed to both local and remote Cell in exactly the same fashion.

2.4.2 Changes suggested to core framework by Vijay Appadurai

2.4.2.1 Improving Efficiency

A new functionality “Matrix Query” was added to improve the messaging efficiency. It is used to query the Matrix for a Cell providing a particular service which returns a pointer to a specific Cell providing the service to make direct, unmediated, communication between Cells possible.

2.4.3 Changes suggested by this research

2.4.3.1 Testing Support

Software Matrix Cells derive from the ITest interface. It is a protocol that should be followed by all the cells in the Software Matrix to support automated test harness based testing of cells. Each individual cell is packaged in the form of a dll file, so an automated test harness server can load dlls (Cells) and call each test routine and log the test results.

2.4.3.2 Intelligent loader

In the original Software Matrix framework, the loader continuously monitors a plugins directory to discover and to add new Cell or to replace an existing Cell. In this research, we have fixed a set of Cells which should be loaded in the Matrix and the loader has information about a particular Cell's connections with other Cells via the relation.xml file. So now the loader is not required to monitor plugins directory continuously but rather it loads them all once, reads the relation.xml file and registers Cells as connected in the Cell Diagram.

2.4.3.2 Auto Cell Discovery

Cells are registered with each other based on connections between them in Cell diagram. In this research, we added an auto Cell discovery feature so that if one Cell is not registered with some other Cell and it tries to send a message to that Cell, given both Cells are registered in Matrix, the other Cell will automatically be registered to the requesting Cell upon service request.

Chapter 3 Sniffer Application

In this chapter we present a process and port sniffer application that was built by adopting the Software Matrix approach to show how effectively Software Matrix can support Model Driven Development for Rapid Application Development, or nearly Instant Application Development in some cases.

3.1 Sniffer Application

The Sniffer Application is a useful tool often used by network administrators to check the security of their network. It scans all machine ports and observes all ports currently hosting a connection. It also scans machine to observe processes running on that machine. It maintains a persistent collection of all ports on which it has observed connection and a property that records if the client had indicated it to be safe and similarly it maintains a persistent collection of safe/unsafe processes and allows user to mark process safe or unsafe.

On a client request it discloses to the client all unsafe or all safe port connections on which connection was observed. It can disclose list of safe and unsafe processes in the same way upon user request. On user command it marks any group of ports/processes as safe or unsafe.

The Sniffer application has a server-client architecture consisting of a sniffer server and sniffer client. Sniffer server is installed on the machine on which we want to observe all processes running or ports currently hosting a connection and client can be running on the same machine or on any other machine within local network which can connect to Sniffer server through socket connection.

3.1.1 Sniffer Application Components

The Software Matrix is concerned with the large building blocks of applications, rather than small fine-grained units of functionality. For this application three major blocks were identified for Sniffer Server, these blocks are ideal candidates to be developed as Matrix Cells:

- Sniffer block which continuously observes all processes running or ports currently hosting a connection and updates safe/unsafe process/port list on user request.
- Communication block to share information about all ports hosting connection, exchange command from client to server to add port to safe/unsafe list.

- Executive block to host above two blocks and provide service to client using sniffer and communication block.

Similarly two blocks were identified on Sniffer client side:

- User interface block that allows the user to operate on application, view processes running or ports hosting connection, view safe/unsafe process/port list, view errors etc.
- Communication block to communicate with Sniffer server

These blocks can be easily developed as a Software Matrix Cell. The application as built contains the required blocks as Cells.

This is screen shot of sniffer application showing safe and unsafe process list:

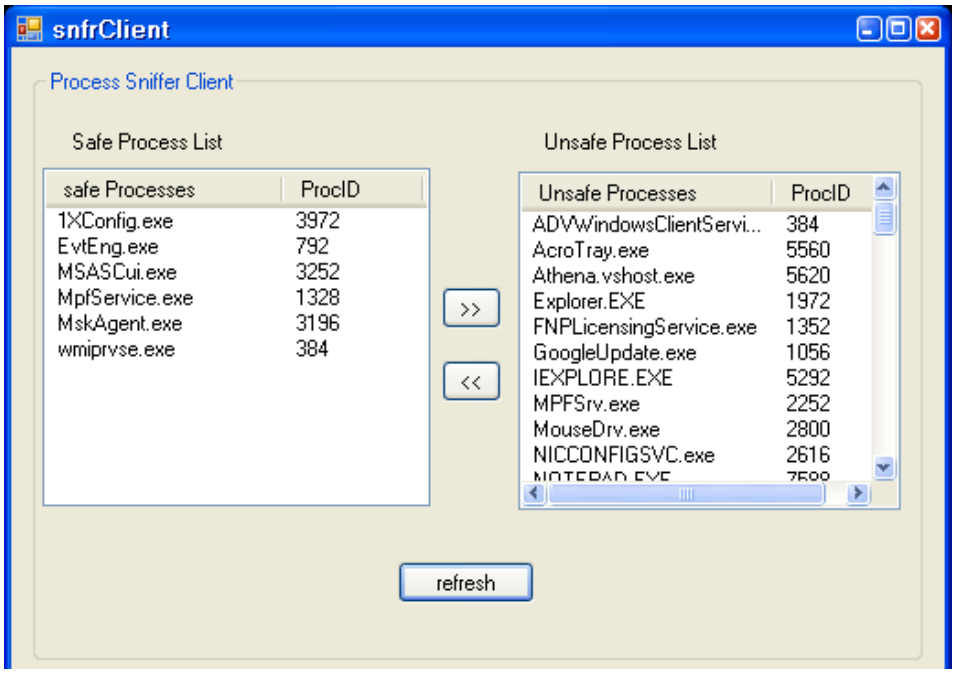


Figure 9 : screenshot of sniffer Client

3.1.1.1 Sniffer Cell

The main function of the Sniffer Cell is to maintain a persistent collection of processes running and ports currently hosting connections and marking them as safe and unsafe on user command.

The Sniffer Cell maintains two threads to monitor running processes and to observe ports hosting connections at regular intervals. Both threads also maintain safe and unsafe process/port list. The persistent safe/unsafe list is thread safe and

synchronization is achieved using GLock[3] class. Users have the option to initialize a safe process/port list, and processes/ports not marked as safe are considered to be unsafe.

The capability list of the Sniffer Cell is populated to indicate the types of messages it is capable of handling. Sniffer Cell can handle service requests for the messages with the following message type:

- matrix.snfrCell.getProcList – this type of message is a request to get list of all the processes running on machine hosting Sniffer Cell.
- matrix.snfrCell.getSafeProcList - this type of message is a request to get list of all processes marked as safe.
- matrix.snfrCell.getUnsafeProcList - this type of message is a request to get list of all processes marked as unsafe.
- matrix.snfrCell.addToSafeProcList - this type of message is a request to add name of process passed as message parameter to safe process list.
- matrix.snfrCell.addToUnsafeProcList - this type of message is a request to add name of process passed as message parameter to unsafe process list.
- matrix.snfrCell.initSafeProcList - this type of message is a request to initialize safe process list with process names passed as message

[3] GLock, Thread classes are taken from Dr. Fawcett's code repository

parameter. If user does not initialize safe process list, all processes are considered to be unsafe by Sniffer Cell.

- matrix.snfrCell.initUnsafeProcList - this type of message is a request to initialize unsafe process list with process names passed as message parameter.
- matrix.snfrCell.getPortList - this type of message is a request to get list of all the ports currently hosting connection on machine hosting Sniffer Cell.
- matrix.snfrCell.getSafePortList - this type of message is a request to get list of all ports marked as safe.
- matrix.snfrCell.getUnsafePortList - this type of message is a request to get list of all ports marked as unsafe.
- matrix.snfrCell.addToSafePortList - this type of message is a request to add port number passed as message parameter to the list of safe ports.
- matrix.snfrCell.addToUnsafePortList - this type of message is a request to add port number passed as message parameter to the list of safe ports.
- matrix.snfrCell.initSafePortList - this type of message is a request to initialize safe port list with port numbers passed as message parameter. If user does not initialize safe port list, all ports hosting connection are considered to be unsafe by Sniffer Cell.

- matrix.snfrCell.initUnsafePortList - this type of message is a request to initialize unsafe port list with port numbers passed as message parameter.

Class diagram for sniffer Cell

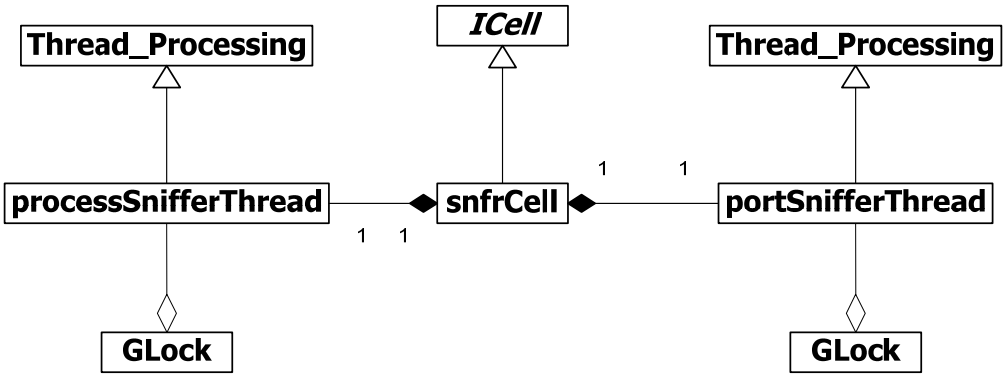


Figure 10 : Sniffer Cell – Class diagram

3.1.1.2 Communication Cell

In the Communication Cell implemented by Anirudha Krishna, the Communication/Network Cell created a channel between different Mediator Cells through which messages could be transferred. Messages may be sent to both Local and Remote Cells in exactly the same fashion. Messages to Remote Cells contained some extra connection information i.e. IP Address and Port number. So to pass a message to a Remote Cell, the Local Cell creates a message similar to messages addressed to other Local Cell but with added IP Address and Port

information so that Mediator can forward that message to Communication Cell for remote delivery.

In this research, Cells are registered to each other by the loader according to connections in the Cell diagram. So if any Cell needs to communicate with a Remote Cell, it could be connected directly with Communication Cell so that it can exchange messages to the remote Cell through Communication Cell without Mediator intervention. Every Cell maintains a list of Cell references based on connections in the Cell diagram. So it can get a reference to the Communication Cell from its local list and can directly pass messages to it. If some Cell is not connected with Communication Cell in Cell diagram and if it requests service of the Communication Cell, in our research, we have provided an auto Cell discovery mechanism. Thus if a Communication Cell reference is not available in the local list, the sender will query the Matrix for Cell reference providing the service and Matrix returns the Cell reference and Cell updates its local list by adding a reference to the Communication Cell. Thus the Communication Cell is establishing channel between different Cells instead of just different Mediators to exchange messages.

The capability list of the Communication Cell is populated to indicate the types of messages it is capable of handling. Communication Cell can handle service request for the messages with the following message type:

- matrix.commCell.listen – this type of message is a request to commCell to start listening on port number passed as message parameter.
- matrix.commCell.sendMessage – this type of message is a request to send xml string message to destination. Xml string, destination IP Address and destination port number are passed to Communication Cell as message parameters.

Class diagram for communication Cell

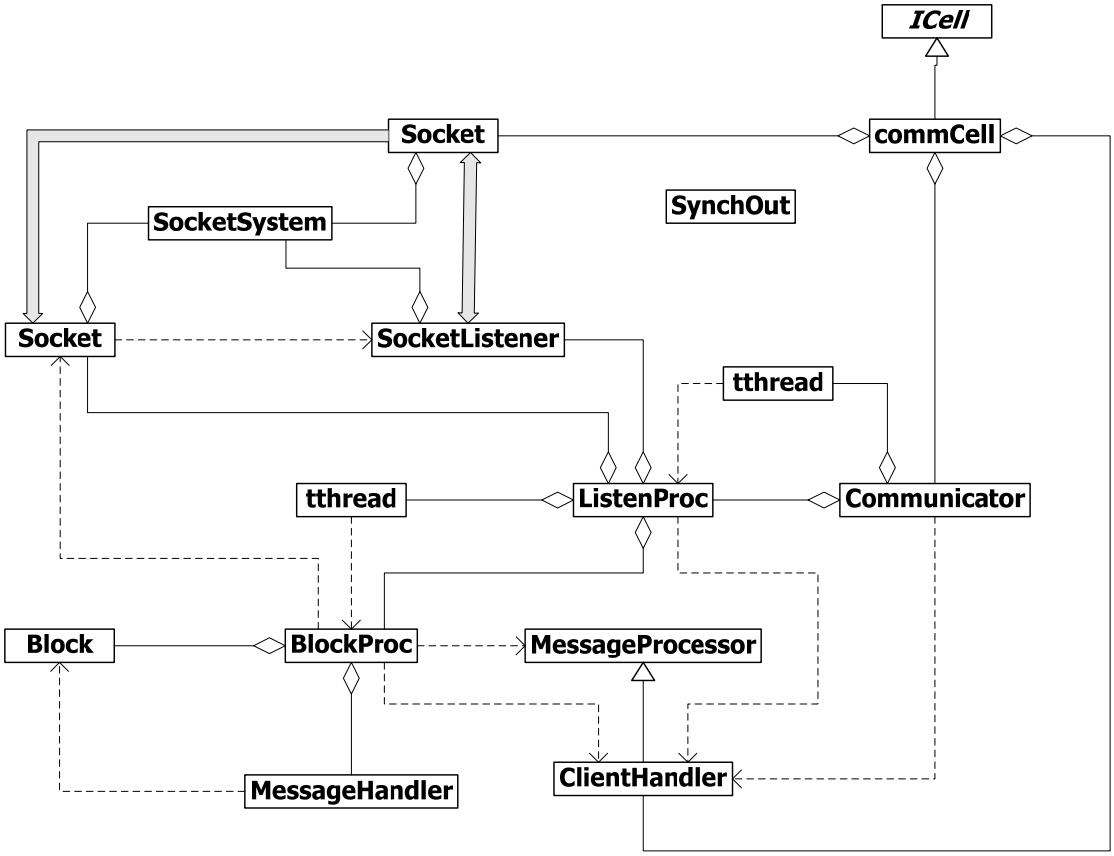


Figure 11 : Communication Cell – Class diagram

The commCell class is the actual Communication Cell and it provides service for above mentioned message types. It is inherited from ICell class which actually provides basic Cell functionality like Blocking Queue, message dispatcher thread etc to exchange messages with other Cells.

The Communicator class provides the abstraction to actual communication activities, i.e. it runs socket listener on its own thread so commCell doesn't block waiting for connections.

Sockets class provides network communication service, using WinSock2. It provides connect request, service to read and write string over socket. SocketSystem provides WinSock loading, unloading and other services.

BlockHandler class provides packaging for raw bytes that sockets know how to send. Blocks are composed of header and data sections.

3.1.1.3 User Interface Cell

As standard C++ Library does not support creation of User Interfaces, User Interface Cell is developed using Managed Extensions for C++ (Managed C++) [24][25]. Managed C++ is Microsoft's set of deviations from C++, including grammatical and syntactic extensions, keywords and attributes, to extend the C++ syntax and language to the .NET Framework. These extensions allow C++ code to be targeted to the Common Language Runtime (CLR) in the form of managed code as well as continue to interoperate with native code.

In Managed C++, multiple inheritances are not supported. This is a design artifact of the CLR. A class managed under the CLR's garbage collector cannot inherit more than one class. A user Interface created using Managed C++ inherits from Form class of Windows Class which is again part of System Class in .NET framework. So the User Interface class can't inherit ICell class and can't communicate with Matrix directly. The problem is solved using a messageHandler class. The User Interface class holds a reference of the messageHandler class and exchanges messages with other cells through this messageHandler class. The messageHandler class inherits from ICell and so it has all the functionality required for message exchange with other Cells. For communication between messageHandler and User Interface class, messageHandler class is required to use interop service from System.Runtime namespace. If any class is inheriting from ICell class, it is required to implement clone() function required for thread processing. If interop functionality is implemented inside messageHandler, it won't be able to implement clone function as it can't marshal between managed type and standard type. The problem is solved with introduction of exchanger class. User Interface class and messageHandler class shares a reference of exchanger class. User Interface class registers one of its methods with a delegate of the exchanger class. So whenever any Cell passes messages to messageHandler, it sends the message to exchanger which forwards it to User Interface Cell through the delegate.

Class diagram for user interface Cell

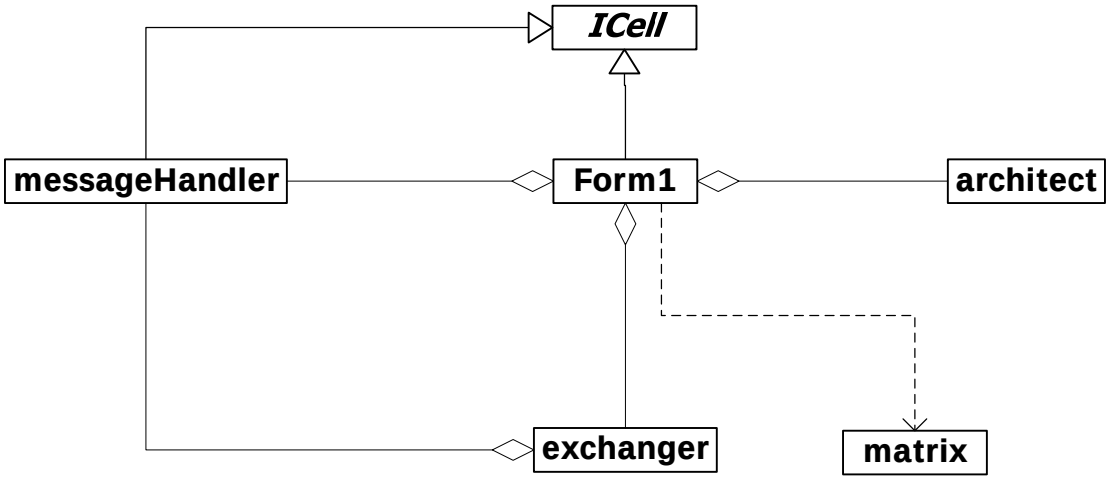


Figure 12 : User Interface Cell – Class diagram

The User Interface Cell is an executive Cell. It loads all Cells (dll files) using the architect class. Architect class is actually a loader class as per the notations used in standard Software Matrix framework defined by Riddhiman Ghosh [9], we call it architect as it does wiring of Cells based on connections in Cell diagrams and constructs the Matrix. Athena Designer generates cellinfo.xml file while generating code. The architect class reads this file and based on information stored in the file, it assigns unique GUIDs to each Cell. The architect class registers Cells based on the connections shown in the Cell Diagram. Athena Designer generates the relation.xml file containing all information about Cell connections shown in the Cell diagram. It generates relation.xml file with the same GUIDs stored in cellinfo.xml and using both files, architect class can register different Cells to create an application. All loaded Cells are also

registered with executive while loaded into the matrix. So if some Cell has no direct reference of some other Cell, it can search for Cell reference using Cells registered with itself. Executive Cell is registered with all Cells loaded into Matrix and all Cells are registered with Executive Cell and this ensures that any Cell can get a reference of another Cell loaded into Matrix even if the two are not registered with each other initially. Executive Cell is an executable file and it creates instance of architect class to load Cells (dll files) into Matrix; so Executive Cell needs to register itself directly using register() function of the matrix class. As it is not loaded by architect, it needs to get a unique GUID on its own. This GUID should match with the GUID assigned to Executive Cell in cellinfo.xml and relation.xml as Athena Designer conveys all information about Cells and Connections through unique GUID. Athena Designer generates guid.txt for Executive Cell to convey this unique GUID and Executive Cell reads guid.txt and uses unique GUID read from file to register itself into matrix.

3.1.2 Constructing Sniffer Application

It is not required to have all Cells ready to build an application. However, to demonstrate application construction, we are building all Cells and generating application using Cells built for this application. If Cells with required functionality are not available at the time of Diagram construction or code generation, we have the option of putting empty place holder Cells for the functionality to be developed in the future. Each Cell is generated, by the Athena

designer, as a separate Visual Studio project and so we can open it anytime and add functionality to it. The Cell Diagram for the sniffer server will look like:

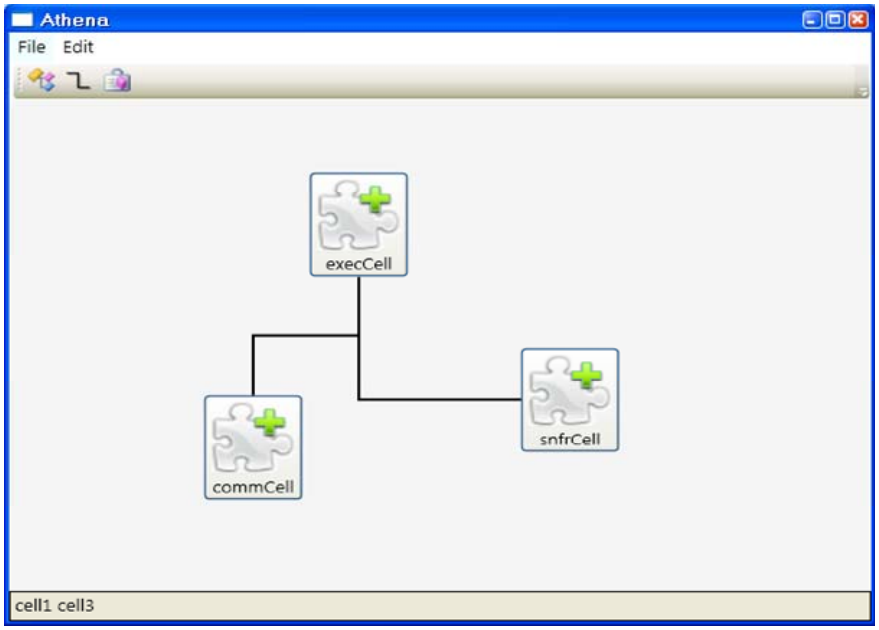


Figure 13 : Cell Diagram for sniffer server

When we click on the generate code button, it will ask for a directory where we want to place code for our application. The Athena Designer will generate code once we provide the path to generate application code and click apply.

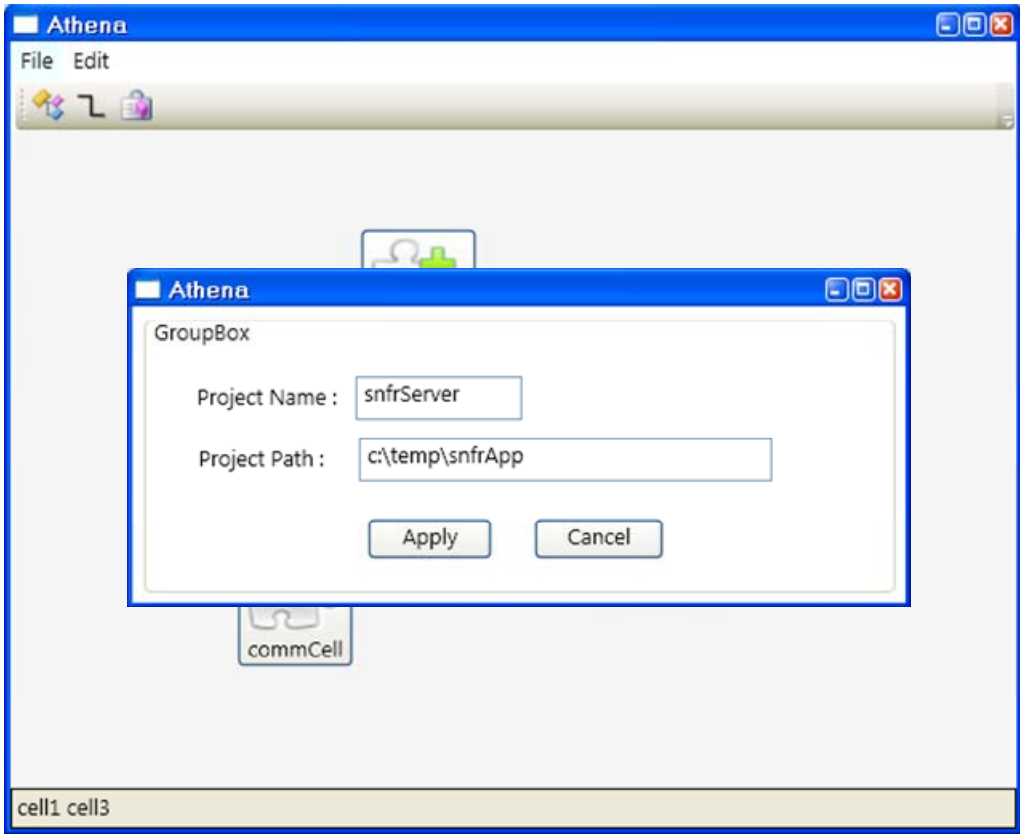


Figure 14 : code Generate properties window

Here is a screenshot of the destination directory where Athena Designer has generated code. Different folders represent different Cells.

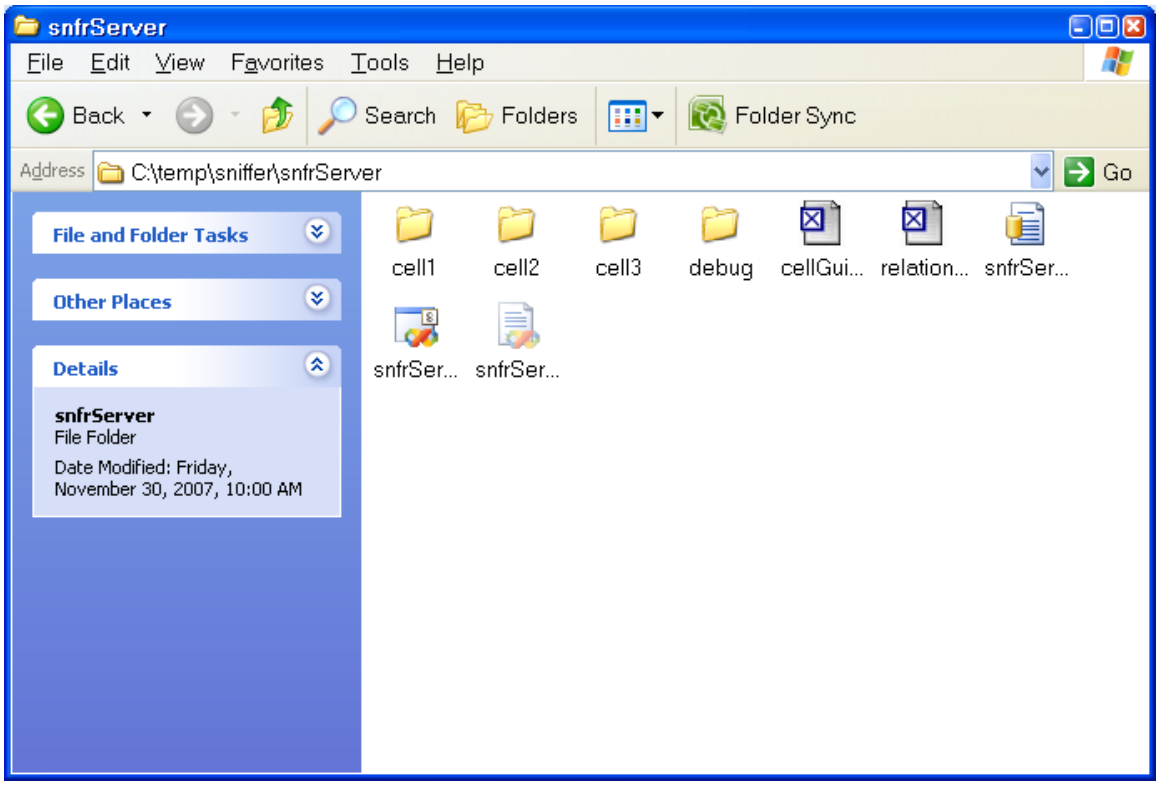


Figure 15 : generated solution for snfrServer

The Cell Diagram for the sniffer client will look like:

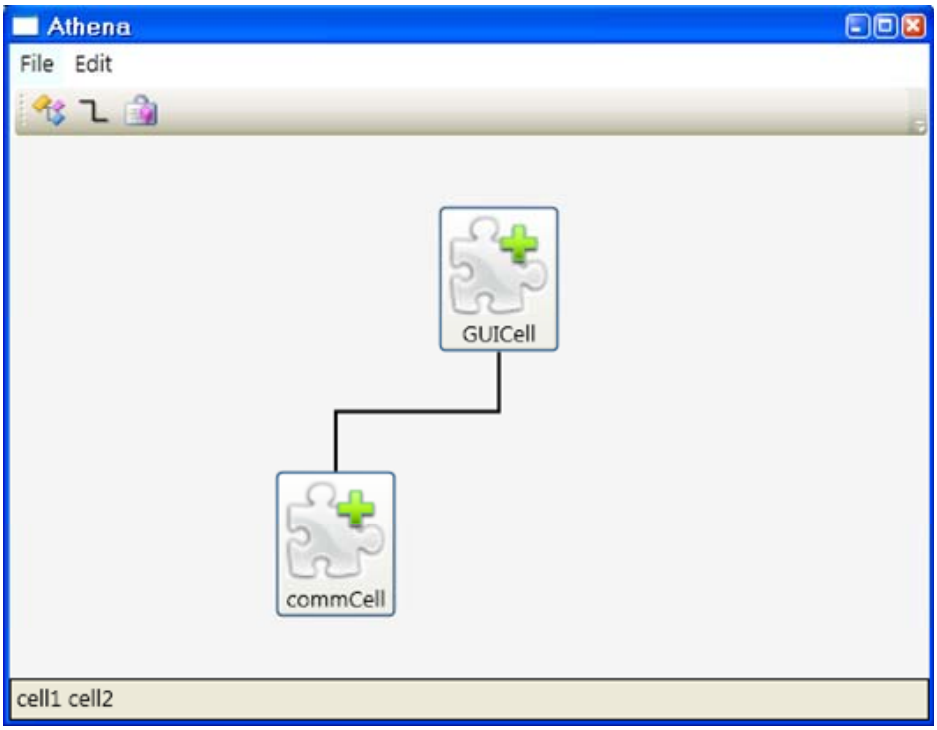


Figure 16 : Cell Diagram for sniffer client

In the same way we need to provide a path for destination directory to generate client application code.

The generated User Interface Cell is just a blank windows form without any controls and so we need to add required components and request messages to get information from sniffer server. Here is the screenshot of the running application after adding all required UI components and request message:

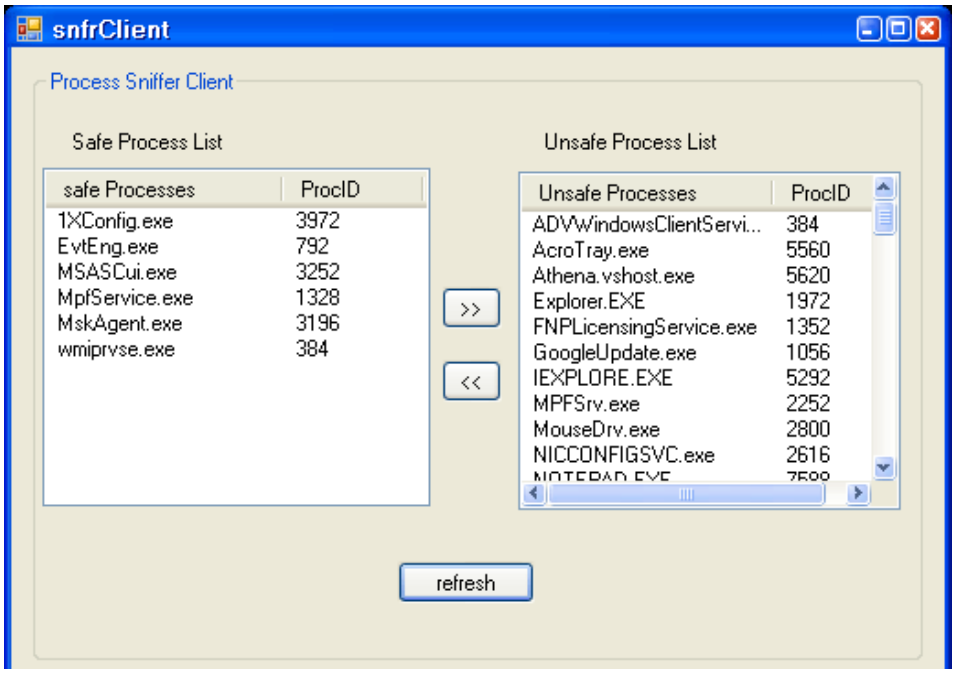


Figure 17 : screenshot of sniffer Client

Chapter 4 Athena Designer

In this chapter we discuss the details of Athena Designer Computer Aided Software Engineering tool that we developed for our research, to demonstrate Model Driven Development using the Software Matrix framework.

4.1 Background

The original idea behind this research work was to build a prototype Computer Aided Software Engineering tool to support Model Driven Development. Instead of reinventing the whole wheel and building a full fledge Integrated Development Environment (IDE), we chose to rather develop a tool providing design time support with compatibility to some well known existing IDE. The tool we developed for our research allows a user to connect Cells visually to construct an application. No coding is required to connect Cells together. The Athena [26] name is chosen for the designer after the Greek Goddess Athena, an armed warrior goddess who is also goddess of wisdom and patroness of weaving, as this tool is actually an assembler using which an application can be constructed by crafting Cells.

Athena Designer is developed using Windows Presentation Foundation (WPF) which is a major component of new .NET 3.0 framework. Windows Presentation Foundation is a framework to deliver high-quality-sometimes cinematic-experience to user. For our research we wanted a tool which can provide a blank

canvas on which user can add Cells, make connections between different Cells, move Cells to any place in the canvas and create a visual model of the application to be constructed. Here are some of the reasons we chose WPF for development of our tool:

- Broad Integration - WPF is a framework for Software Developers and Graphic Designers to create modern user experience without mastering several difficult technologies like GDI+, DirectX etc. To build applications with 2D graphics there are several independent technologies and number of inconsistencies. however WPF covers all these areas with a consistent programming model as well as tight integration when each type of media gets composited and rendered. Graphics effects are consistently applied across different media types.
- Resolution independence - WPF puts emphasis on vector graphics and so elements can be enlarged or shrunk independent of screen resolution.
- Declarative programming – With introduction of Extensible Application Markup Language (XAML) applications can be developed using a large range of expressive artifacts, based on arranging markup in a common-sense fashion. Styles, templates, and skins can be defined for a user interface and can be separated from code which implements behaviors.
- Rich composition and customization – WPF controls are extremely composable. All controls are now containers and so any control can be embedded inside the other control and no code for this customization is required to be written. Using this functionality we created a canvas, and

to present a Cell in Cell Diagram, we just add a button with customized look to canvas. That can then be moved across the canvas as the user chooses.

The Athena Designer generates code in a Visual Studio Solution which is compatible with Microsoft Visual Studio 2005 and later versions. Microsoft Visual Studio is a complete set of development tools for building ASP.NET Web applications, XML Web services, desktop applications, and mobile applications. Visual Basic, Visual C++, Visual C#, and Visual J# all use the same integrated development environment (IDE), which allows them to share tools and facilitates in the creation of mixed-language solutions. Generally Cell diagrams consist of a number of Cells and connections between them which represent the system as a whole. Code is generated using the same idea. If we compare generated code and its Cell diagram, each individual project represents a Cell in the diagram and all such projects become part of one Visual Studio Solution which represents the application as a whole.

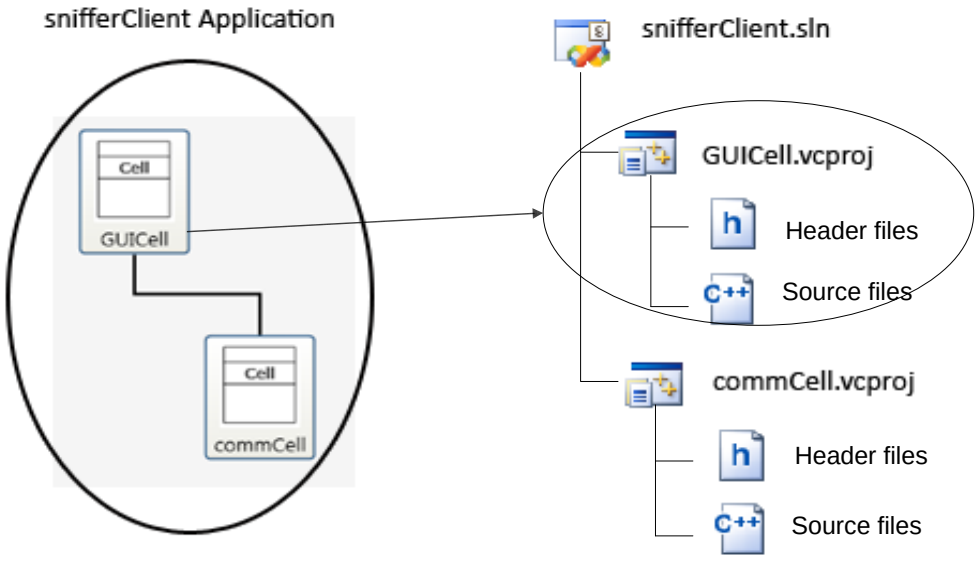


Figure 18 : each Cell is generated as separate project. All projects are part of one solution file which represent whole application

4.2 Building Blocks of Athena Designer

Class Diagram for Athena Designer

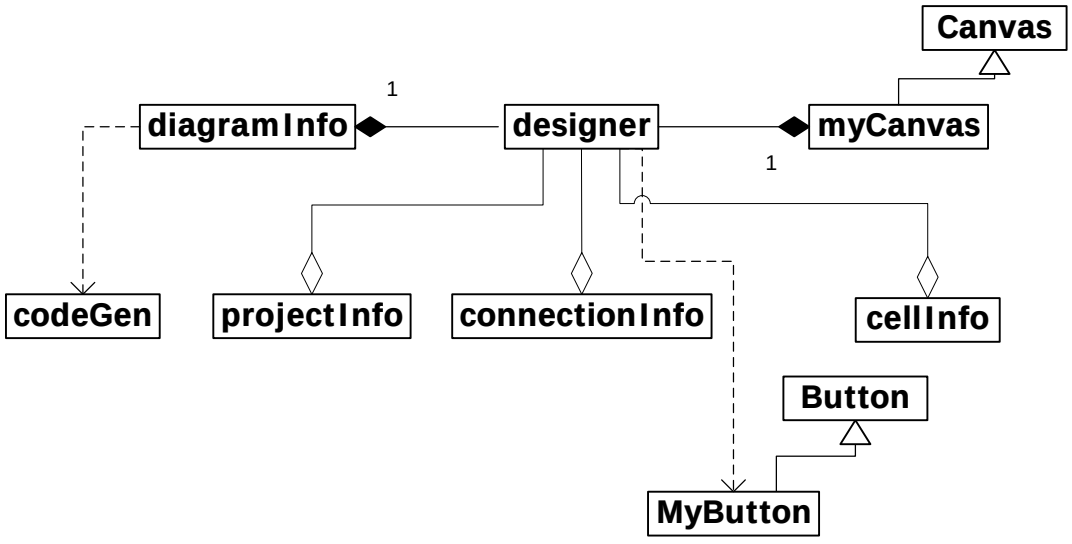


Figure 19 : Athena Designer - Class Diagram

4.2.1 myCanvas Class

This class provides a hosting panel and the user can create a Cell Diagram on the panel. Cells can be moved, and, if moving Cell is connected with other Cells, it will automatically redraw connections between them. myCanvas class derives from Canvas. It also provides support for modifying the z-order of the elements it contains. Most panels in WPF provide automatic layout support such as the docking behavior of DockPanel or the item wrapping behavior of the WrapPanel. The Canvas is the only panel in WPF layout system to provide absolute positioning similar to putting controls on a WinForms panel. The elements in a Canvas are never moved or resized by the Canvas. This property is therefore useful as we do not want to change exact size and the location of visual objects due to window resizing or any similar action.

To create a Cell Diagram it allows hosting of any class which has UIElement class as its base class. However the designer class restricts it and only hosting of MyButton (this class element represents Cells in Cell Diagran) and PolyLine (it draws connection between two Cells) is allowed on myCanvas. So when user clicks on myCanvas panel, the designer class captures user clicks, and based on type of request, it notifies myCanvas class. If a user click is the action to add a new Cell, the designer class creates a new MyButton class instance and adds it to children collection of myCanvas instance. If user click is to add a new connection between two Cells, designer class creates and adds a new PolyLine instance

between two Cells and again PolyLine instance is added to the children collection of myCanvas instance.

By default, the DragCanvas manages dragging of every element in its children collection. Elements AllowDragging property can be set to false to disable dragging of elements. By default Cells (MyButton class instances) have higher Z-index value and so they appear in front of connections (PolyLine instances).

The element dragging logic is comprised of three steps:

1. When the left mouse button is depressed, the myCanvas looks for a child UIElement at the current mouse cursor location. If it finds one, a reference to that element is stored, information about the element's location is saved, and the cursor location is cached.
2. When the mouse moves, if there is a Cell being dragged (i.e., Cell was found when the mouse button was depressed), then that Cell will be relocated by the distance between the old and current cursor locations, relative to the original Cell location.
3. Eventually, when a mouse button is released, the drag Cell reference is nullified, so that when the mouse moves, there is no Cell to relocate.

When mouse moves and if Cell is being dragged and it has any connections with other Cells on myCanvas, it will automatically redraw connection lines based on a

routing algorithm. When dragging a Cell on myCanvas panel, the Cell at other end of the connection is considered to be the center of the coordinate system used to draw the connection between two Cells. The routing algorithm divides the plane into quadrants and again divides each quadrant into two equal parts – a region for line with two folds and a region for line with three folds.

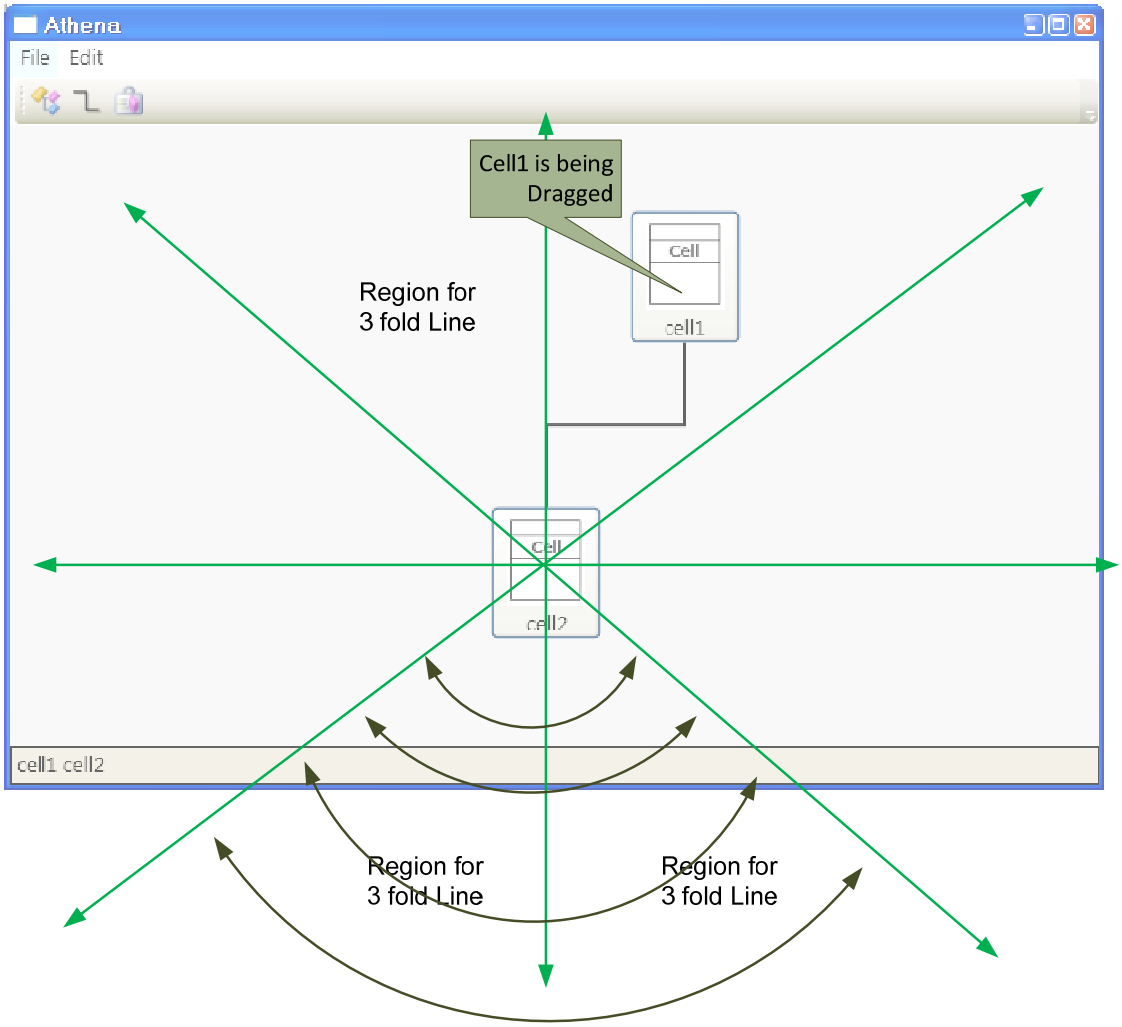


Figure 20 : Region with 3-fold line

The figure, above, shows a region for 3 fold line. Cell1 is connected with Cell2. When Cell1 is being dragged, considering myCanvas as plane, Cell2's center is considered as center of coordinate system. If Cell1 falls in a region of 3 fold lines, a connection with 3 line segments is drawn. If Cell1 falls in a region of 2 fold lines, a connection with 2 line segment is drawn.

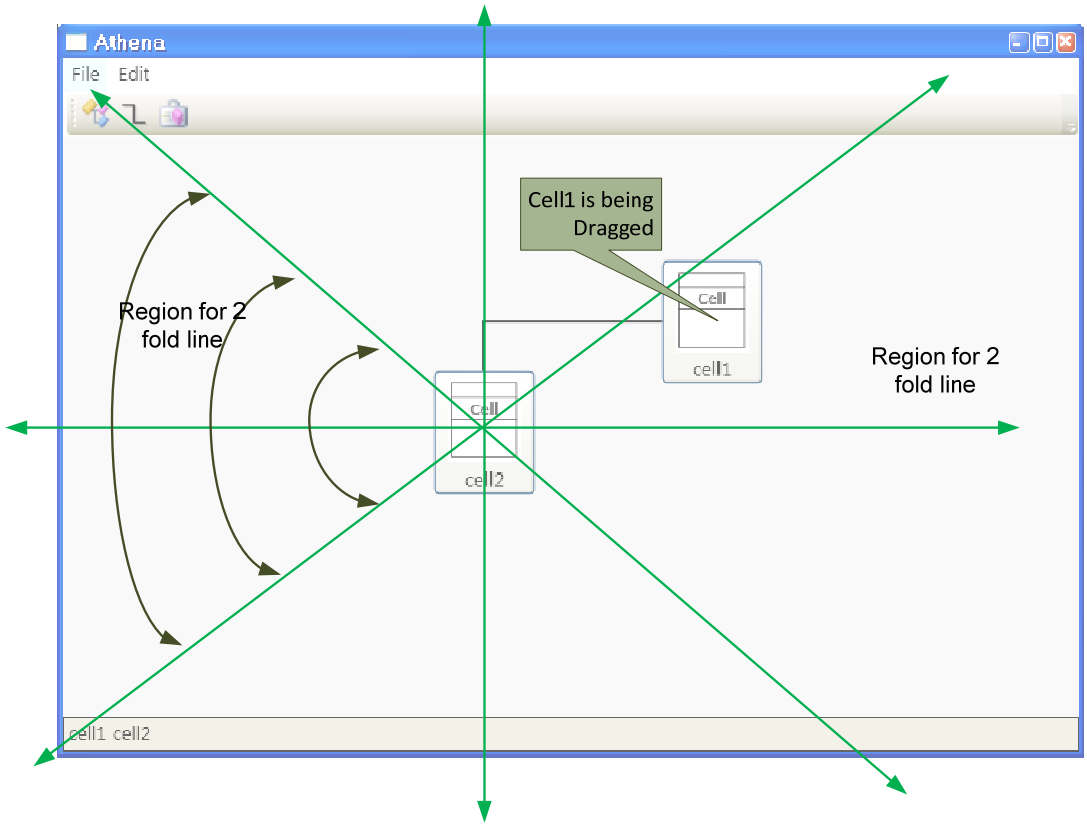


Figure 211 : Region with 2-fold line

This figure shows a region for 2 fold line. To determine in which region a Cell being dragged falls, the slope of the line connecting two cells is calculated as shown in the figure. Based on that slope, different PolyLine point locations are updated and will redraw the line automatically. How slope is calculated and how point locations are assigned is shown in next two figures.

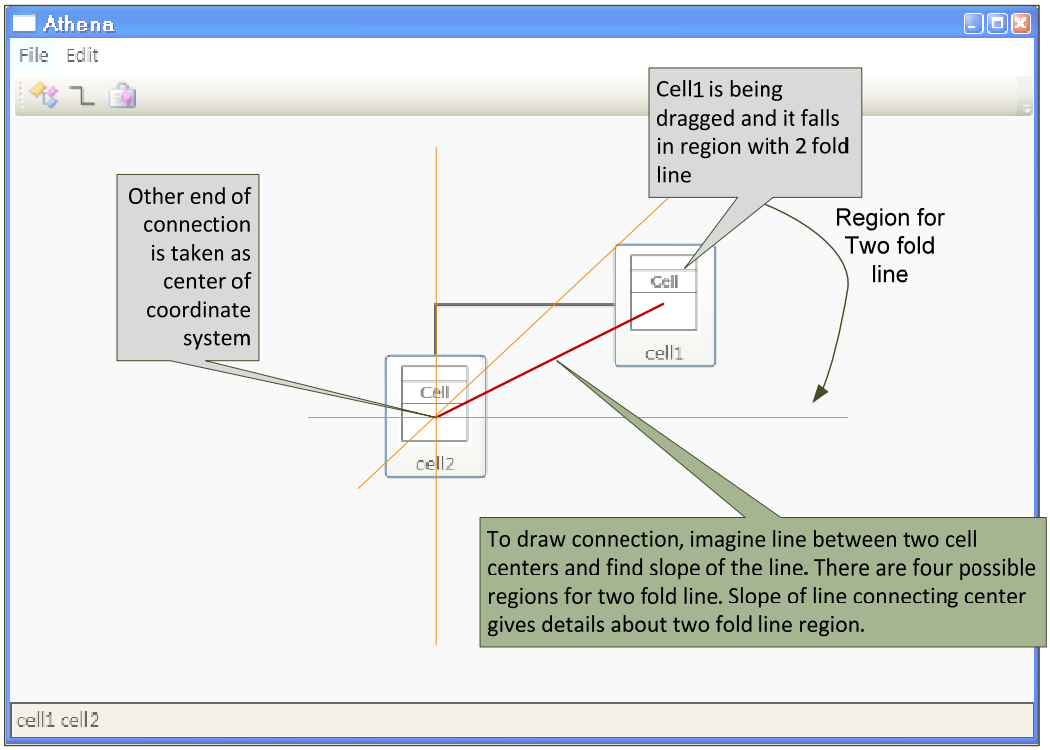


Figure 222 : Selecting Region to draw connection

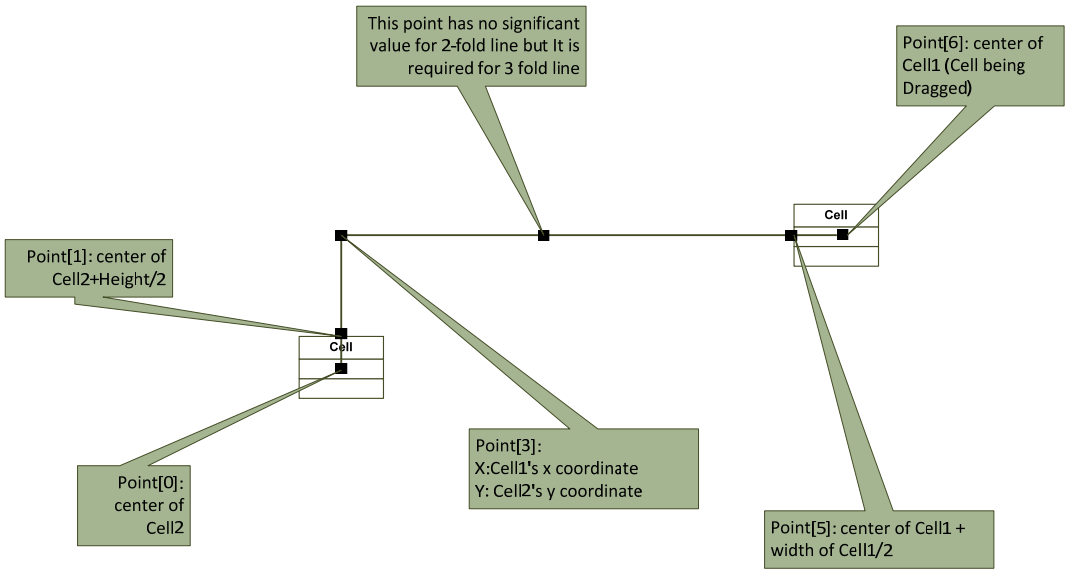


Figure 233 : creating point collection to draw line between Cells

4.2.2 diagramInfo Class

digramInfo class maintains all necessary information required from the Cell Diagram to generate code. The designer class maintains some information necessary for it to redraw connections between Cells and allow dragging of Cell. The information maintained by designer class is mapped to information which can be directly used for code generation and mapping. Both are maintained by diagramInfo class. Whenever a new Cell is added or a new connection is made between two Cells, information about addition or new connection is passed to diagramInfo class and position of the Cell or points collection for a connection is maintained by the designer class. The diagramInfo class correlates both pieces of information.

diagramInfo class maintains information about the number of Cells in a diagram, user defined name of Cell, type of Cell, connection between Cells, type of connection, auto generated GUIDs for each Cell etc. On user command, designer class lets user change some of the auto generated information. Notification of such changes are sent to the diagramInfo class and it updates necessary information locally. When a user clicks on the code generate button from toolbar, diagramInfo class calls methods of codeGen class to generate necessary project files and solution file based on available information.

4.2.3 codeGen Class

This class provides static methods to generate project and solution files compatible with Microsoft Visual Studio 2005. Microsoft Visual Studio has a predefined set of template files for different project types and using those templates we created a template suitable for code generation.

Original project file snippet	Template to create emptyCell snippet
<?xml version="1.0" encoding="Windows-1252"?> <VisualStudioProject ProjectType="Visual C++" Version="8.00" Name=”emptyCell” ProjectGUID=”{BD2CE710-DEE2-4b4b-869F-4B6E0CC612C8}” RootNamespace=”matrix”	<?xml version="1.0" encoding="Windows-1252"?> <VisualStudioProject ProjectType="Visual C++" Version="8.00" Name=" [! safe projectname] " ProjectGUID=" { [! safe GUID] } " RootNamespace="[!safe rootnamespace] "

The table, above, shows snippets from original Visual Studio Project file (files with extension .vcproj) and template file to create empty cell (we have used .tmpl extension). Necessary information is provided by diagramInfo class and the template file is populated with this information. Then a new project file (file with .vcproj extension) is generated in the destination directory. The structure of

Microsoft Visual Studio solution files (file with .sln extension) depends entirely upon the number of projects and their respective GUIDs so it is generated on the fly without using template. For more information about file format of .vcproj and .sln file see appendix item A. Microsoft Visual Studio requires the directory structure to be specified in project and solution file and we have used a simple structure, storing each project file (.vcproj) and files part of that project (source, header, resource files) in a separate directory. Each project directory is stored in the root directory along with the solution file.

4.2.4 designer Class

The whole UI of Athena Designer is defined using markup language XAML and code behind logic is defined in this class.

The designer class is the executive class of the Athena Designer application. It loads myCanvas panel and lets users construct Cell Diagrams. It directs other classes to let a user change default behavior of Diagram i.e. by changing Cell name, Cell type, type of connection etc. It maintains information about Cells available on myCanvas panel, and connections between Cells. It also maintains information necessary to redraw a connection while Cell is being dragged. When a user adds a new Cell to diagram, Cell name is auto generated and diagramInfo class is notified about Cell creation. It will store Cell location locally and add the instance to children collection of myCanvas. In the same way, when two Cells are

connected via a connection, it notifies diagramInfo class about it and stores points collection for that connection locally and points collection instance is added to myCanvas children collection.

When a user right clicks on any Cell added to Cell Diagram, it creates instance of cellInfo class so that the user can specify parameters to change the default behavior and properties of the Cell. When user right clicks on the connection, it creates instance of connectionInfo class so that user can change type of connection.

4.2.5 MyButton Class

This class actually represents Cells in the Cell diagram. It derives from Button class so basically it's a button control with different look and feel, which can be loaded on myCanvas panel and dragged across the panel. It adds new properties of name and image source to default behavior. When an instance of MyButton class is created, it initializes its source property with Cell image and name with default auto generated Cell name. It also stores information about the number of connections maintained with other instances created on Cell Diagram.

4.2.6 ProjectInfo Class

This class creates a window to enter project name and destination directory to generate code. Later this information is passed to diagramInfo and codeGen class by diagram class.

4.2.7 connectionInfo Class

This class creates windows to change default connection type. To draw a connection between two Cells user needs to click on source Cell (“from” Cell) and then destination Cell (“to” Cell). By default connection type is always assumed to be source to destination. That means the loader will register the destination Cell with source Cell. A user can change this default behavior to destination to source type of connection or both Cells holding reference of each other when loaded into the matrix.

4.2.8 cellInfo Class

The cellInfo class creates a window to change default Cell properties. When user adds a new Cell to Cell Diagram, designer class will create default name for Cell and Cell type is set to emptyCell by default. User Can change this to any unique Cell name and any type listed in the window.

4.3 Screen shots of Athena Designer

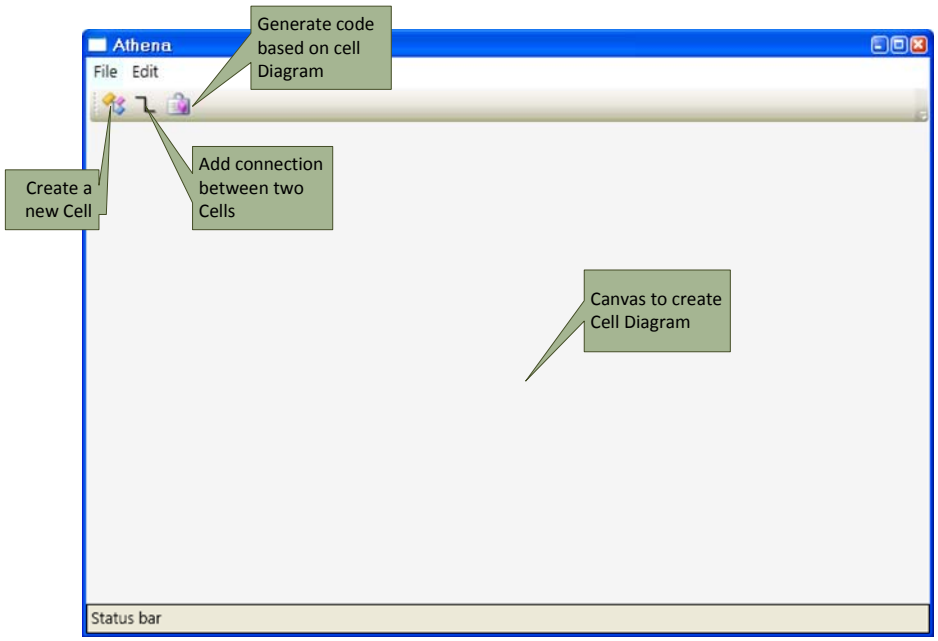


Figure 244 : Different toolbar buttons and their function

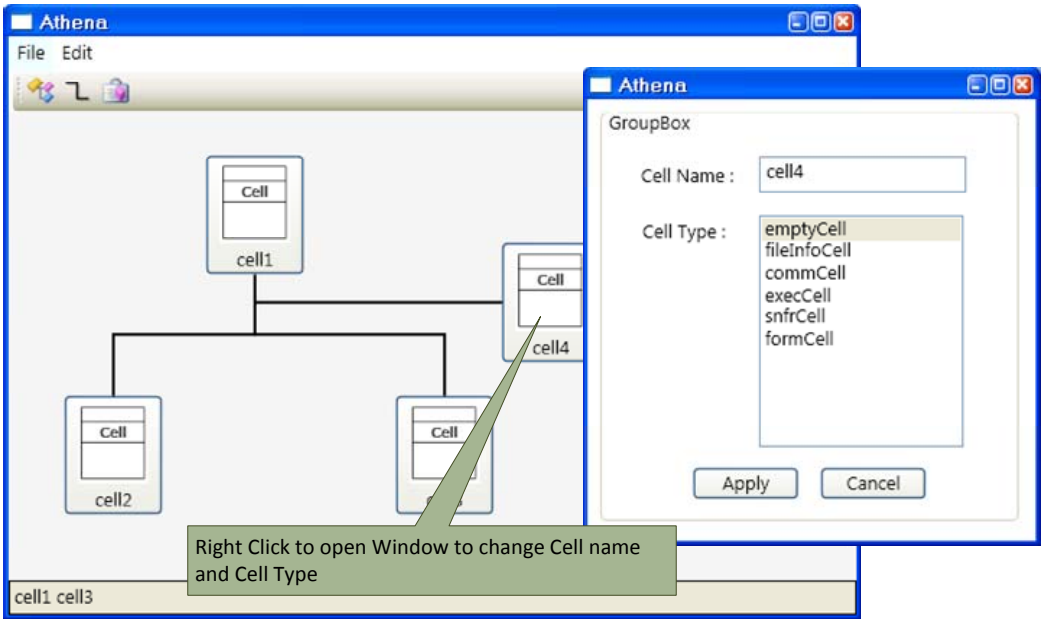


Figure 255 : Edit default Cell properties

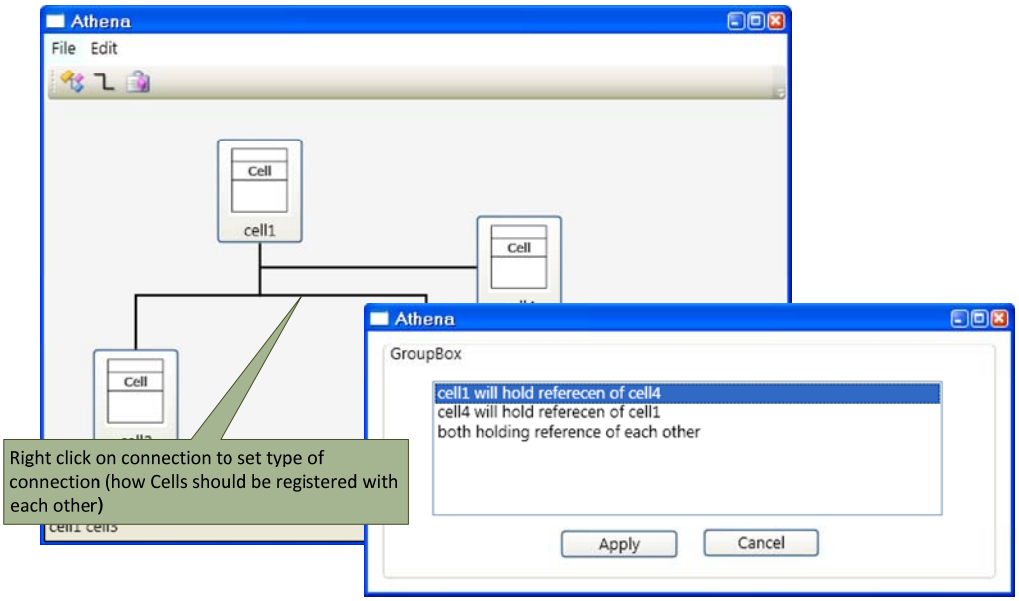


Figure 266 : Edit default connection Type

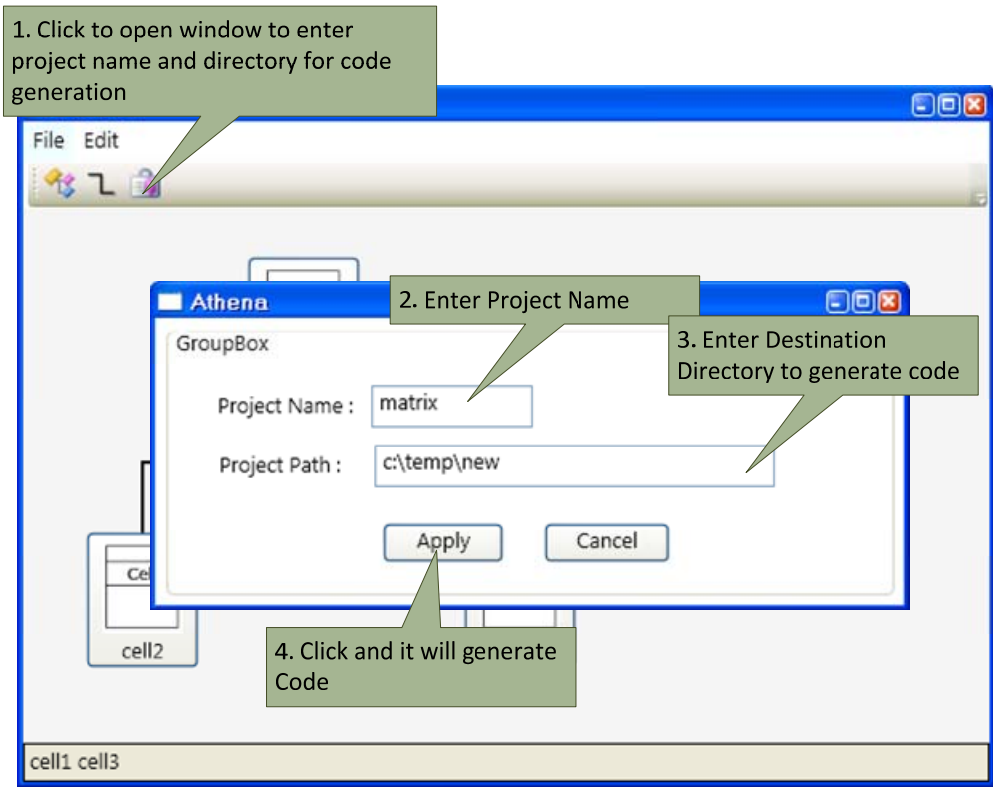


Figure 277 : Parameters required for code generation

Chapter 5 Conclusions

This chapter summarizes the goals of the Model Driven Development, our contributions to Model Driven Development using the Software Matrix for effective software salvage, and future work.

5.1 Literature Review

In the bibliography, we cite some papers and journal articles that are related to our work or have provided insight or inspiration for this effort. In order to provide a context for our contributions, we review some of them that have been particularly useful.

Ju An Wang [7] proposed a solution for highly reusable software components and suggested that a revolutionary change is required to move from writing code line by line to assembling prebuilt components systematically. In our research, we proposed a simple framework for creating reusable components which can

even encapsulate object oriented reuse to provide high level of abstraction. The framework proposed by our research does not required drastic changes in the software development process and can be easily adapted with current software development processes.

Giuliano Armano and Michele Marchesi [6] described application development using a tool named UMLTALK to generate prototype code in Smalltalk language based on diagrams constructed using UML. They claimed that small and medium sized projects based on refactoring and supported by suitable language and tools allow team productivity to be greatly enhanced. In the architecture proposed by this research, an application is viewed as compositions of different pieces-of Cells in the Matrix-where some of the Cells are to be developed in future and some Cells encapsulate existing functionality. We also observed that by using refactoring activities at design and analysis level instead of just using at the implementation level improved productivity significantly.

Albert F. Case, Jr. [8] described an evaluation and implementation strategy for using Computer Aided Software Engineering to improve systems development productivity. In his conclusion he wrote that Computer Aided Software Engineering can have dramatic impact on our ability to improve systems development productivity. We also observed with the architecture and tool proposed by our research, that model driven development increases development

speed and helps to ensure that quality is built into system at every phase of development.

L. Balmelli, D. Brown, M. Cantor and M. Mott [27] described a Model Driven Development approach using RUP® SE (Rational Unified Process® for Systems Engineering) and described Model Driven Development approach well-suited to a system development environment characterized by rapidly changing conditions and requirements. In our research we observed that Model Driven Development is based on system decomposition rather than requirement decomposition and the resulting systems are better suited for internal reuse. Model Driven Development leads to robust solutions that provide a solid basis for evolving the system as stakeholders require change.

5.2 Addressing Model Driven Development

Effective recycling of software assets has been an important, yet partially unfulfilled goal of the software engineering discipline; if integrated with a Model Driven Development approach it holds out the promises of robustness and improved quality, gains in development schedule and quicker time to market, and reduced development and maintenance cost. While these benefits are widely recognized, practice of systematic and effective reuse with Model Driven Development approach is certainly not widespread.

In this thesis we have been concerned with finding the missing link between effective reuse of software assets and Model Driven Development. We also aimed to facilitate building applications with effective software asset salvage without changing the software development process drastically.

5.3 Contributions of this Thesis

Our contribution in this thesis focused on significantly improving Model Driven Development by promoting software assets reuse using Software Matrix framework.

- We proposed an architecture which supports Model Driven Development using the Software Matrix, a framework supporting software salvage. So we demonstrated that the Software Matrix can be used effectively to reuse existing software assets to build a new application with very little effort. We also demonstrated that an application could be built graphically by just connecting different Cells with each other.
- We also added testing support to Software Matrix. Matrix Cells now follow a common testing protocol and so could be loaded independently into a test harness server for testing purpose. We also improved performance of the Software Matrix by allowing direct communication links between Cells residing on different machines instead of establishing links between two mediators and exchanging messages through mediator links. We also added an auto Cell discovery feature into Software Matrix so now Cells can discover a Cell containing required functionality but not registered for direct message exchange and can exchange message directly after discovery.

- We also developed a sample sniffer application using Model Driven Development approach. We generated code for the sniffer application by constructing a Cell Diagram consisting of required functionality Cells and connections between them.

5.4 Future Work

During the process of working on Model Driven Development using the Software Matrix, directions for future research and development were identified. Pursuing these areas will add several improvements to the Software Matrix.

5.4.1 Integrated Designer, Repository and Test Harness

Athena Designer reads manifest file to get information about required files to generate code for particular Cell types. Based on information in a manifest file, it copies required files from local codedb directory. This could be improved by integrating a code repository system and test harness server[4]. So while generating code, designer could make a request to the repository system for required information. While developing a system, user creates empty Cells for functionality to be developed in future. User may even change or enhance existing functionality. In such cases, loader could detect such changes and request test harness server to perform testing before loading dll into the system.

[4] This structure was discussed by Vijay Appadurai [10]

5.4.2 Auto upgrading Systems

The current architecture could load Cells dynamically and add new functionality to a system which is alive. We could enhance it further by adding support to replace an existing Cell if a newer version of Cell becomes available. Mediator detects the change and moves old Cell into special zone and loads new Cell in same zone. In special zone mediator lets old version Cell finish message processing based on messages in its queue and drops a kill message for old version Cell. All new messages are passed to new version Cell. When old version Cell finishes its message processing, mediator replaces it with the new version Cell, maintaining temporary message queue while upgrading Cell reference in Matrix so no message is lost and later delivers all messages received while upgrading to new version Cell for processing.

5.4.3 Cross Platform Wrapper Cells

Vijay Appadurai [10] demonstrated cross platform development using the Software Matrix. This same idea could be integrated with designer, and while constructing a Cell Diagram, it could be constructed with Cells developed using different platforms. When two Cells developed using different platform are connected in diagram, designer would automatically add a necessary wrapper between them based on user specified configuration. It would require different mediators responsible for a specific platform type. The wrapper Cell actually hides message exchange

between two cross platform Cells and does message exchange through platform specific mediators using cross platform message exchange protocol.

5.4.4 Load Balancing

Software Matrix could be enhanced for multiprocessor systems where mediator could automatically clone Cells in high demand including itself and make them run on separate thread to improve overall response from the Cells in high demand.

5.4.5 Communication wrappers for frequent message exchange with remote Cell

We could create server client application using Software Matrix. It is possible that one local Cell communicates frequently with remote Cell. We could develop communication wrapper Cell which makes permanent channel with remote Cell to improve performance and local Cell could just talk to remote Cell assuming its available locally. Communication wrapper Cell creates illusion that remote Cell is now available locally and manages all message exchange with remote Cell.

Appendix

A. Visual Studio 2005 project and solution file format details [23]

The format of these files is not published or documented by Microsoft and could change drastically in future versions.

Solution File

Visual Studio creates two separate files when we create a new solution in Visual Studio.

- The first file is the *.suo* (solution user options) file (it is a hidden file), which stores user settings such as the location of your breakpoints. If we don't generate this file, Visual Studio will generate this file when we open the solution in Visual Studio. So in this research we are not generating this file. This file is stored in binary format, which does not lend itself to easy editing.
- The second file that Visual Studio creates is the *.sln* file. This file stores information about the projects that make up this solution, source control settings, and other global settings that apply to all of the projects in the solution. In this research we are generating a *.sln file.

Details of the Solution file:

The first line of the solution file contains a declaration including the version of Visual Studio that this solution is built for:

```
Microsoft Visual Studio Solution File, Format Version 9.00
```

Visual Studio 2005 contains an additional line under this first line, which contains just the following:

```
# Visual Studio 2005
```

This line causes the icon of the file to be changed to a Visual Studio 2005-specific icon, and if removed, will prevent Visual Studio from opening this file when double-clicked on it.

The next portion of the solution file contains a section for each of the projects that are contained in this solution:

```
Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") = "matrix",
"cell5\cell5.vcproj", {7b783271-7cb8-4763-85fc-475fe2f1efa7}
EndProject
Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") = "matrix",
"cell4\cell4.vcproj", {bfd462df-9c2a-4b4d-91f7-110b477eb014}
EndProject
Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") = "matrix",
"cell1\cell1.vcproj", {934cf759-752e-49ee-8074-48614df092b4}
EndProject
Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") = "matrix",
"cell3\cell3.vcproj", {13d191ca-eac1-401c-9af4-955630f0f394}
EndProject
Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") = "matrix",
"cell2\cell2.vcproj", {8bac3539-7d90-420c-bf9d-18196135b9bd}
       EndProject
```

Each project has a `Project` and `EndProject` tag as well as a `ProjectSection` tag to track the dependencies for the project. The first GUID in the project tag is used to identify what type of project this is. In this instance, the GUIDs for all projects are same as they are Win32 console applications. The strings on the right

side of the equals sign include the name of the project, its path relative to the solution root, and the unique GUID for this project. This GUID is used for a number of things including tracking dependencies.

*Project GUIDs can be found in registery at key HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\8.0\Project*s.

Under this key are all the GUIDs for the project available for local machine; the name and the extension of the project are listed as well.

In Visual Studio 2005, project dependencies are also tracked. These are not the implicit dependences that are created by project references. When Project A references Project B, there is an implicit dependency. Project B must be built before Project A since it references the other project's output. Since this type of dependency is project-specific information, it is stored in the project file. Project files and dependencies are described in the later section.

The dependences stored in the ProjectSection tag are configured at the solution level. They define when a project must be built before another project, but they may not directly reference each other. This is stored in the ProjectSection tag of the project that is dependent on the other project.

Visual Studio 2005 completely omits the ProjectSection tag if there are no dependencies for the project.

The next section in the solution file is the Global section, which begins with a Global tag and ends with an EndGlobal tag. Inside these tags are a number of GlobalSection tags that store an array of different pieces of information, including the configuration settings for various projects as well as source control information. Here is a look at the Global section of this example solution file from Visual Studio 2005:

```
Global
	GlobalSection(SolutionConfigurationPlatforms) = preSolution
		Debug|Win32 = Debug|Win32
		Release|Win32 = Release|Win32
	EndGlobalSection
	GlobalSection(ProjectConfigurationPlatforms) = postSolution
		{7b783271-7cb8-4763-85fc-475fe2f1efa7}.Debug|Win32.ActiveCfg = Debug|Win32
		{7b783271-7cb8-4763-85fc-475fe2f1efa7}.Debug|Win32.Build.0 = Debug|Win32
		{7b783271-7cb8-4763-85fc-475fe2f1efa7}.Release|Win32.ActiveCfg =
Release|Win32
		{7b783271-7cb8-4763-85fc-475fe2f1efa7}.Release|Win32.Build.0 =
Release|Win32
		{bfd462df-9c2a-4b4d-91f7-110b477eb014}.Debug|Win32.ActiveCfg = Debug|Win32
		{bfd462df-9c2a-4b4d-91f7-110b477eb014}.Debug|Win32.Build.0 = Debug|Win32
		{bfd462df-9c2a-4b4d-91f7-110b477eb014}.Release|Win32.ActiveCfg =
Release|Win32
		{bfd462df-9c2a-4b4d-91f7-110b477eb014}.Release|Win32.Build.0 =
Release|Win32
		{934cf759-752e-49ee-8074-48614df092b4}.Debug|Win32.ActiveCfg = Debug|Win32
		{934cf759-752e-49ee-8074-48614df092b4}.Debug|Win32.Build.0 = Debug|Win32
		{934cf759-752e-49ee-8074-48614df092b4}.Release|Win32.ActiveCfg =
Release|Win32
		{934cf759-752e-49ee-8074-48614df092b4}.Release|Win32.Build.0 =
Release|Win32
		{13d191ca-eac1-401c-9af4-955630f0f394}.Debug|Win32.ActiveCfg = Debug|Win32
		{13d191ca-eac1-401c-9af4-955630f0f394}.Debug|Win32.Build.0 = Debug|Win32
		{13d191ca-eac1-401c-9af4-955630f0f394}.Release|Win32.ActiveCfg =
Release|Win32
		{13d191ca-eac1-401c-9af4-955630f0f394}.Release|Win32.Build.0 =
Release|Win32
		{8bac3539-7d90-420c-bf9d-18196135b9bd}.Debug|Win32.ActiveCfg = Debug|Win32
		{8bac3539-7d90-420c-bf9d-18196135b9bd}.Debug|Win32.Build.0 = Debug|Win32
		{8bac3539-7d90-420c-bf9d-18196135b9bd}.Release|Win32.ActiveCfg =
Release|Win32
		{8bac3539-7d90-420c-bf9d-18196135b9bd}.Release|Win32.Build.0 =
Release|Win32
	EndGlobalSection
	GlobalSection(SolutionProperties) = preSolution
		HideSolutionNode = FALSE
	EndGlobalSection
EndGlobal
```

The `SolutionConfiguration` and `ProjectConfiguration` sections contain the build configuration settings for the solution and its projects. The

`ExtensibilityGlobals` and `ExtensibilityAddIns` sections are included for the benefit of add-in authors. The `ExtensibilityGlobals` section can be used to store global information about the solution, and the `ExtensibilityAddIns` section lists all the add-ins that are used in this solution. In this research we haven't added any add-in so that section is not present in solution file.

If the FALSE in `HideSolutionNode` in `GlobalSection` is switched to TRUE, then the solution node is hidden in the IDE.

Project File

Each project creates a number of files to store information about itself. This includes a project file and a user settings file. The extension for the project file is based on the language type; for example, a C# Project is saved with the extension *.csproj* and a C++ Project is stored with the extension *.vcproj*. Thankfully, the internal formats of these various files are based on the same XML schema. The beginning of each project file includes some basic information about the project, including the version of Visual Studio that it was created for as well as the GUID for this project. Here is an example of this section:

```
<?xml version="1.0" encoding="Windows-1252"?>
<VisualStudioProject
      ProjectType="Visual C++"
      Version="8.00"
      Name="cell1"
      ProjectGUID="{934cf759-752e-49ee-8074-48614df092b4}"
      RootNamespace="matrix"
      Keyword="Win32Proj"
      >
```

Next section has information about platform, various project properties and settings like output directory, character set etc. Mostly it is all information that is seen when right clicked on project to open properties dialog box.

```
<Platforms>
		<Platform
			Name="Win32"
		/>
	</Platforms>
	<ToolFiles>
	</ToolFiles>
	<Configurations>
		<Configuration
			Name="Debug|Win32"
			OutputDirectory="$(SolutionDir)$(ConfigurationName)"
			IntermediateDirectory="$(ConfigurationName)"
			ConfigurationType="1"
			CharacterSet="0"
			>
			<Tool
				Name="VCPreBuildEventTool"
			/>
			<Tool
				Name="VCCustomBuildTool"
			/>
			<Tool
				Name="VCXMLDataGeneratorTool"
			/>
			<Tool
				Name="VCWebServiceProxyGeneratorTool"
			/>
			<Tool
				Name="VCMIDLTool"
			/>
			<Tool
				Name="VCCLCompilerTool"
				Optimization="0"

PreprocessorDefinitions="WIN32;_DEBUG;_CONSOLE"
				MinimalRebuild="true"
				BasicRuntimeChecks="3"
				RuntimeLibrary="3"
				UsePrecompiledHeader="0"
				WarningLevel="3"
				Detect64BitPortabilityProblems="true"
				DebugInformationFormat="4"
			/>
			<Tool
				Name="VCManagedResourceCompilerTool"
			/>
			<Tool
				Name="VCResourceCompilerTool"
			/>
			<Tool
				Name="VCPreLinkEventTool"
			/>
			<Tool
```

```
                        Name="VCLinkerTool"
                              LinkIncremental="2"
                        GenerateDebugInformation="true"
                              SubSystem="1"
                        TargetMachine="1"
                  />
                  <Tool
                        Name="VCALinkTool"
                  />
                  <Tool
                        Name="VCManifestTool"
                  />
                  <Tool
                        Name="VCXDCMakeTool"
                  />
                  <Tool
                        Name="VCBscMakeTool"
                  />
                  <Tool
                        Name="VCFxCopTool"
                  />
                  <Tool
                        Name="VCAppVerifierTool"
                  />
                  <Tool
                        Name="VCWebDeploymentTool"
                  />
                  <Tool
                        Name="VCPostBuildEventTool"
                  />
            </Configuration>
            <Configuration
                  Name="Release|Win32"
                  OutputDirectory="$(SolutionDir)$(ConfigurationName)"
                  IntermediateDirectory="$(ConfigurationName)"
                      ConfigurationType="1"
                  CharacterSet="1"
                  WholeProgramOptimization="1"
                  >
                  <Tool
                        Name="VCPreBuildEventTool"
                  />
                  <Tool
                        Name="VCCustomBuildTool"
                  />
                  <Tool
                        Name="VCXMLDataGeneratorTool"
                  />
                  <Tool
                        Name="VCWebServiceProxyGeneratorTool"
                  />
                  <Tool
                        Name="VCMIDLTool"
                  />
                  <Tool
                        Name="VCCLCompilerTool"

PreprocessorDefinitions="WIN32;NDEBUG;_CONSOLE"
                        RuntimeLibrary="2"
                        UsePrecompiledHeader="0"
                        WarningLevel="3"
                        Detect64BitPortabilityProblems="true"
```

```
                DebugInformationFormat="3"
            />
            <Tool
                Name="VCManagedResourceCompilerTool"
            />
            <Tool
                Name="VCResourceCompilerTool"
            />
            <Tool
                Name="VCPreLinkEventTool"
            />
            <Tool
                Name="VCLinkerTool"
                LinkIncremental="1"
                GenerateDebugInformation="true"
                     SubSystem="1"
                OptimizeReferences="2"
                EnableCOMDATFolding="2"
                TargetMachine="1"
            />
            <Tool
                Name="VCALinkTool"
            />
            <Tool
                Name="VCManifestTool"
            />
            <Tool
                Name="VCXDCMakeTool"
            />
            <Tool
                Name="VCBscMakeTool"
            />
            <Tool
                Name="VCFxCopTool"
            />
            <Tool
                Name="VCAppVerifierTool"
            />
            <Tool
                Name="VCWebDeploymentTool"
            />
            <Tool
                Name="VCPostBuildEventTool"
            />
        </Configuration>
</Configurations>
```

The next part of the Build section contains all of the references for this project. Here is an abbreviated example of this section as we have not added any reference:

```
<References>

</References>
```

After the `Build` section is the `Files` section of the project file, which can be seen here:

```
<Files>
		<Filter
			Name="Source Files"
			Filter="cpp;c;cc;cxx;def;odl;idl;hpj;bat;asm;asmx"
			UniqueIdentifier="{4FC737F1-C7A5-4376-A066-
2A32D752A2FF}"
			>
			<File
				RelativePath="..\BlockingQueue.cpp"
				>
			</File>
			<File
				RelativePath="..\Threads.cpp"
				>
			</File>
			<File
				RelativePath="..\cell4.cpp"
				>
			</File>
		</Filter>
		<Filter
			Name="Header Files"
			Filter="h;hpp;hxx;hm;inl;inc;xsd"
			UniqueIdentifier="{93995380-89BD-4b04-88EB-
625FBE52EBFB}"
			>
			<File
				RelativePath="..\ICell.h"
				>
			</File>
			<File
				RelativePath="..\BlockingQueue.h"
				>
			</File>
			<File
				RelativePath="..\Threads.h"
				>
			</File>
		</Filter>
		<Filter
			Name="Resource Files"

	Filter="rc;ico;cur;bmp;dlg;rc2;rct;bin;rgs;gif;jpg;jpeg;jpe;resx;t
iff;tif;png;wav"
			UniqueIdentifier="{67DA6AB6-F800-4c08-8B7A-
83BB121AAD01}"
			>
		</Filter>
	</Files>
	<Globals>
	</Globals>
```

This section simply tracks all of the files that are included in this project and in the end we have globals section.

Bibliography

1. The Object Management Group, not-for-profit computer industry consortium [Online] www.omg.com.
2. **Szyperski, Clemens.** *Component Software: Beyond Object-Oriented Programming ISBN 0-201-74572-0.* 2nd. Boston : Addison-Wesley Professional, 2002.
3. **Messerschmitt, David G.** *Software Ecosystem: Understanding an Indispensable Technology and Industry.* s.l. : MIT Press, 2003
4. **Pressman, Roger S.** Software Engineering: A practitioner's Approach 5th Edition, McGraw Hill, 2001
5. **McKinney, Alfred L.,** A Report on Computer Aided Software Engineering (CASE), Journal of Computing Sciences in Colleges, Volume 6 , Issue 5 (May 1991), Pages: 85 – 92
6. **Armano, Giuliano and Marchesi, Michele**, A Rapid Development Process with UML, ACM SIGAPP Applied Computing Review, Volume 8 , Issue 1 (April 2000), Pages: 4 - 11
7. **Wang, Ju An**, Towards Component-based Software Engineering, Consortium for Computing Sciences in Colleges & Proceedings of the seventh annual CCSC Midwestern conference on Small colleges, Valparaiso Univ., Valparaiso, Indiana, United States, Pages: 177 – 189, Year of Publication: 2000

8. **Case, Albert E, Jr.,** Computer Aided Software Engineering (CASE): Technology for improving software development productivity, ACM SIGMIS Database, Volume 17 , Issue 1 (Fall 1985), Pages: 35 - 43
9. **Riddhiman Ghosh and Dr. James Fawcett,** *An Architecture for Software Salvage, Master's Thesis 2004*

   http://www.ecs.syr.edu/faculty/fawcett/handouts/webpages/softwarematrix.htm
10. **Vijay Appadurai and Dr James Fawcett,** *Cross Platform Development using the Software Matrix Master's Thesis 2007*

    http://www.ecs.syr.edu/faculty/fawcett/handouts/webpages/SWMatrixCrossPlatformDev.htm.
11. **Anirudha Krishna and Dr James Fawcett,** *A self-healing framework using the Software Matrix structure, Master's Thesis 2005*

    http://www.ecs.syr.edu/faculty/fawcett/handouts/webpages/SelfHealingSoftwareMatrix.htm
12. Component-based software engineering (CBSE)

    [Online] http://en.wikipedia.org/wiki/Software_component.
13. Microsoft Corp, Component Object Model Technologies

    [Online] http://www.microsoft.com/com/default.mspx.
14. Microsoft Developer Network, Component Object Model

    [Online] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/componentobjectmodelanchor.asp.
15. Enterprise JavaBeans Technology

    [Online] http://java.sun.com/products/ejb/index.jsp.

16. Sun EJB Sepcification

[Online] http://java.sun.com/products/ejb/docs.html.

17. IBM Software-Rational Rose

[Online] http://www-306.ibm.com/software/awdtools/developer/rose/index.html.

18. *Sparx Systems' Enterprise Architect.*

[Online] http://www.sparxsystems.com/products/ea.html.

19. *Eclipse - an open development platform.*

[Online] http://www.eclipse.org/

20. *The official MDA Guide Version 1.0.1.*

[Online] http://www.omg.org/docs/omg/03-06-01.pdf.

21. Windows Presentation Foundation.

[Online] http://wpf.netfx3.com/

22. Visual Studio Home

[Online] http://msdn2.microsoft.com/en-us/vstudio/default.aspx

23. OnDotNet.com Hacking Visual Studio

[Online] http://www.ondotnet.com/pub/a/dotnet/excerpt/vshacks_chap1/index.html?page=4

24. Managed Extensions for C++

[Online] http://en.wikipedia.org/wiki/Managed_Extensions_for_C++

25. Managed Extensions for C++ Background

[Online] http://msdn2.microsoft.com/en-us/library/aa712625(VS.71).aspx

26. Athena [Online] http://en.wikipedia.org/wiki/Athena

27. L. Balmelli, D. Brown, M. Cantor, and M. Mott, Model Driven System Development, IBM Systems Journal, Volume 45, Number 3, 2006