The Software Matrix:
An Architecture for Software Salvage

Riddhiman Ghosh

Thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Engineering

Advisor:
James Fawcett

Department of Electrical Engineering and Computer Science,
Syracuse University

November 2004
Syracuse, New York

The Software Matrix:
An Architecture for Software Salvage

Riddhiman Ghosh

(ABSTRACT)

Effective recycling of software assets has been a long-standing goal of the software engineering discipline, holding out promises of reduced development and maintenance costs, gains in development schedule, quicker time-to-market, and robustness and quality over software developed without reuse. However few organizations in the software industry practice 'systematic software reuse'. There is a disconnect between the high-ceremony methods recommended by the reuse research community, and what is practically adopted by software practitioners. Anecdotal evidence shows us that the practice of 'software salvage' is widely practiced, especially in small and medium-sized software shops. By salvage we mean lifting of significant blocks of existing systems and trying and inserting them into a newly developed system.

The goal of this thesis is to simplify software salvage to make it a useful paradigm, and the research here is directly concerned with innovative ways of leveraging major parts of existing software systems. While *reuse* connotes immutability—— the individual pieces may be designed and implemented to be adaptable but with no intent to change even a single character of source code, s*alvage* makes no guarantee of immutability. Very often the source code of individual pieces being salvaged is modified——large pieces we want to recycle often have many dependencies on parts we don't want, which may result in expensive changes to the part being salvaged. While it may seem easy to lift major blocks of code from one system for use within another, the truth is quite different due to these dependencies of the salvaged part on the system from which it was extricated, often making effective salvage very difficult to accomplish.

We propose an architecture for system construction aided by a framework, called the "Software Matrix", that actively supports and promotes the salvage of components. Rather than concentrating only on programming languages as in object-oriented theory, or only on packaging techniques as in component-oriented technologies (both to a certain extent meant to address reuse), our approach goes a step further by viewing applications solely as compositions of different pieces——of "Cells" in a Matrix——and having a framework that actively supports the dynamic composition of applications from the collaboration of these different pieces. In particular the

Software Matrix is a runtime infrastructure into which individual pieces of an application can plug. Applications are then composed from these different pieces by having them collaborate by strictly passing only messages to each other.

By limiting the hard problem of general reuse to a smaller domain——of salvage within an organization, to be used in the construction of modest-sized systems——we try to achieve something close to plug-and-play software construction, which can be likened to the software-IC model.

By enforcing loose coupling of system components (Cells), by employing messaging passing and mediator structures, and by supporting discovery of needed types we've built a pluggable architecture that can gracefully adapt to salvage operations. The Software Matrix takes care of the compositional aspects of software by dynamically composing systems. It connects the right system elements at runtime without one having to bind to anything explicitly at compile time.

The ideas presented in this thesis were implemented and the Matrix infrastructure was built using Microsoft's .NET framework and the C# programming language. This infrastructure was used to construct two real-world tools in order to understand the issues involved. The methodology required to build systems using the Software Matrix is discussed in detail. Our techniques are evaluated on the basis of cognitive distance, performance, code size and ease of salvage, and the results obtained are encouraging. As an experiment these tools were developed both using traditional development, and using the Software Matrix method of system construction. The penalties in terms of performance or code size were not serious with the advantage of having systems that were amenable to salvage operations.

Acknowledgments

I would like to thank my advisor Dr. James Fawcett for the immeasurable guidance and support he has accorded me throughout this work. His cheerful advice and encouragement was something I came to count on through the course of my graduate studies.

I wish to thank the members of my committee for agreeing to spend their valuable time on this dissertation. Thanks are also due to the Chairman, Department of Electrical Engineering and Computer Science for providing me with resources and facilities for research.

I have greatly benefited from my discussions with my colleagues at Hewlett-Packard Laboratories and I wish to thank them for their guidance.

I am indeed lucky to have had the support of my friends through this essentially lonely journey, and would not have been able to make it without them.

Finally, for always believing in me, and for their unconditional love and support, to my parents and sister I give the thanks which they will never demand.

Table of Contents

Chapter 1

Introduction

It has been estimated in the literature that there are approximately 100 billion lines of code at work in the world today [1]. Studies [2] have shown that large fractions of code in systems are functionally equivalent to already existing code. And yet new systems are often constructed without leveraging existing code bases. The same functions have been written several times over——some estimates say that less than 15% of new code serves an original purpose [3]. One can therefore argue that this inclination to reinvent the wheel in the software industry has made the construction of software needlessly error-prone and expensive since we are maintaining multiple copies of essentially the same software. This would imply that the world-wide cost of developing, testing and documenting a piece of

software is multiplied by the number of equivalent copies in existence. Moreover it is likely that the same issues and bugs that have been taken care of elsewhere, surface again if we have not taken advantage of previously tested (and therefore trusted) code. This waste of resources, manpower and time involved in software production has led it to been likened to craftsmanship in a cottage-industry rather than a mature engineering discipline [5], [6].

It is no surprise therefore that a long-standing goal of the software industry has been effective recycling of software assets; and the study of *reuse* has been an important branch of the software engineering discipline, holding out promises [7] of reduced development and maintenance costs, gains in development schedule and quicker time-to-market, and robustness and quality over software developed without reuse.

1.1 THE PROBLEM

The concept of reuse is not new. In fact the simple sub-routine construct is an example of reuse, which allows carrying out a set of operations repeatedly without having to explicitly write out the same set of instructions every time. In current vocabulary the term reuse almost always implies systematic reuse——“an institutionalized organizational approach to product development in which software assets are intentionally created or acquired to be reusable” [8]. However in spite of a general acknowledgment of the importance of reuse and a recognition of

its potential benefits, there has been no widespread reuse revolution——few organizations practice systematic reuse. The observation is that systematic reuse is hard [9].

There is a disconnect between what is recommended by the reuse research community and what is adopted by software practitioners. This “gap between theory and practice” in relation to reuse R&D has been described this way [10]:

> Not all the “theoretical oriented” and “sophisticated” solutions presented by researcher[s] have ...thrilled the practitioners...
> ...the reuse community has worked on complex technologies and methods with high ceremony, yet most of the software community seems to be looking for simpler solutions... High ceremony methods require an organization with high process maturity to achieve success.

1.1.1 REUSE VS. SALVAGE

One can see in the industry the practice of *software salvage*. By salvage we mean the lifting of significant blocks of existing systems and inserting them into a newly developed system.

In this thesis, a distinction is made between the terms software reuse and software salvage: *reuse* connotes immutability——the individual pieces meant for reuse cannot be modified. They are designed and implemented to be adaptable but with no intent to change even a single character of source code. The motivation is to maintain only a single copy not multiple copies of essentially the same software. *Salvage* on the other

hand makes no guarantee of immutability. Very often the source code of individual pieces being salvaged is modified —— large pieces we want to recycle often have many dependencies on parts we don't want, which may result in expensive changes to the part being salvaged. While it may seem easy to lift major blocks of code from one system for use within another, the truth is quite different due to these dependencies of the salvaged part on the system from which it was extricated, often making effective salvage very difficult to accomplish.

As anecdotal evidence of salvage we cite the case of radar systems software development at The Radar Systems Department, formerly a component of the General Electric Company in Syracuse, New York. During our experience (Fawcett) with this particular engineering organization we have observed, and participated in, software salvage that was routinely attempted to minimize development costs so that competitive bids for new systems could be prepared. In the building of a new radar, large pieces of the software of existing radar systems——pieces not necessarily designed for reuse——were extracted. Modifying and getting these extracted pieces to work together was often a challenging task, given the dependencies of the extracted piece on the system in which it was originally embedded.

One way to look at salvage is to think of it as analogous to transplanting the digestive system or heart from one living organism to another. The problems of terminal arterial bleeding in this seemingly contrived analogy are also very like the

problems we see when we pull out a part from one software system, due to the connectedness inherent in typical software.

The problem as we see it, therefore, is the difficulty in salvaging existing parts of a system and building new systems from them, with ease. The research in this thesis is directly concerned with innovative ways of leveraging major parts of existing software systems——of simplifying software salvage to make it a useful paradigm.

Whether the formal, process-heavy systematic reuse methodologies, in their current form, will be widely adopted by organizations is still an open question [11]. Regardless, we feel the practice of software salvage will still be in use in the foreseeable future, especially in small and medium-sized software shops. Our contribution in this thesis centers on significantly managing the problem of salvage. The techniques presented here aim at eliminating or reducing changes required by salvage, so that salvage moves closer to the reuse model in the sense of immutability, but does not require high-process organization.

In the next few sections we consider prior approaches to tackling the problem of reusing software assets and will give an overview of our approach to simplify the paradigm of software salvage——a means of pragmatic reuse.

1.2 PRIOR APPROACHES

Two major (and fairly recent) approaches towards encouraging reuse have been object-oriented reuse, and component-oriented reuse.

1.2.1 OBJECT-ORIENTED REUSE

Some of the most notable attempts at enabling reuse have come from the proponents of object-oriented (OO) technologies. Booch [12] asserts the use of OO techniques such as inheritance and parameterization as effective mechanisms for reuse. Indeed one of the classic motivations of object-orientation has been reuse. The Booch object-oriented libraries available in languages such as C++ and Ada were meant to promote reuse of code through specialization of library classes. The libraries consist of about 500 classes and functions with variations of collection and container abstractions of varying time and space complexities, and are considered one of the early attempts to popularize software reuse through object-oriented libraries.

There have been several such object-oriented libraries since, for instance [23]. Object-oriented theory mandated discipline in writing code through best-practice guidelines such as separating interface from implementation, and encouraged reuse through inheritance and composition. While these steps are very helpful and object-oriented technologies are popular, they clearly have not engendered the reuse revolution that was hoped

for. In fact the only object-oriented libraries that are widely used are user-interface frameworks such as the Microsoft Foundation Classes [13] and libraries for data structures [23]. Object Oriented techniques have made compiler libraries a very effective mechanism of software reuse. However, business organizations have had a much harder time getting leverage from their pre-existing software assets only through the creation and consumption of object-oriented libraries.

One reason perhaps is that the definition of an object——an encapsulation of state and behavior——is a purely technical definition [14], with no direct mapping to the actual physical unit that is deployed, versioned and potentially reused. Instead of black-box reuse deep entangled multiple inheritance hierarchies were common such as in MFC [13]. Also objects do not quite fit the salvage model very well. From a granularity perspective they are usually of inappropriate size to be mixed and matched——they are of the size of grains of sand, while typically what salvagers look for in building systems are bricks.

1.2.2 COMPONENT-ORIENTED SYSTEMS

The logical next-step was the notion of components and component-oriented systems. A component is a coherent package of software implementation that can be independently deployed and composed with other components. It is the unit of composition and versioning with contractually specified interfaces [15], [16]. Components are not the same as objects. Among other

things, a key difference is the level of granularity——components are usually of a coarser granularity; they are typically larger than classes (e.g. in COM[1] technology a component is a set of classes with one class acting as the interface between the other classes in the component and the outside world. While object technology is commonly used in the implementation of components, it is not a requirement for building components.) The notions of independence and deployability (which implies executable or loadable code) of components are important in relation to their potential for reuse [17]. The defining feature of component technology is that a system, which is composed of components, can be repaired or made better by updating components, without rebuilding the whole system. This is a very important feature for systems composed of millions of lines of code. Two such systems have been built in Syracuse —— the Over The Horizon Radar, and the BSY – 2, submarine battle management system. Other examples come to mind like the Windows operating system.

Some of the popular component technology standards include Microsoft's COM ("an architecture and supporting infrastructure for building, using and evolving component software in a robust manner" [18]), CORBA (Common Object Request Broker Architecture) from the OMG ("provides a foundation for making diverse applications work together in distributed heterogeneous environments" [19] and Sun Microsystems' Enterprise JavaBeans(EJB) ("a component architecture for the development

[1] Component Object Model (COM) is a component technology offering from Microsoft.

and deployment of component-based distributed business applications" [20], and the Microsoft .Net Framework, which exposes nearly all of the Windows API and much more as components. Although the specifics of these technologies are different, some with perhaps differing goals, a common objective of each of these is to enable component reuse.

The existence of component-oriented technologies has not solved the problems that plague software salvage raised in the beginning of this chapter. Surely adopting a component-oriented approach to software construction, rather than only concentrating on the programming language as in object-oriented theories, is beneficial. As a matter of fact, the implementation of the concepts outlined in this thesis uses .NET Component technology. But packaging techniques alone are not enough.

1.3 OUR APPROACH

We are of the view that *components are useful only if there is a framework that actively supports and promotes their reuse.* J2EE and .Net both have such framework support, and widespread use indicates that it is indeed successful. However, that success is focused on providing effective solutions for generic industry problems, e.g., network communication, web publishing, and enhancing application security. We seek to support vertical, perhaps proprietary, applications, where building a huge framework is impractical from a return on investment point of view. Our approach to make salvage easier

is to view applications solely as compositions of different pieces and have a framework of collaborating pieces from which applications can be composed dynamically.

An enduring vision in the software industry has been that of “Software-ICs”——a term first coined by Brad Cox [21]. We have always wanted to achieve in software something analogous to what is common in hardware——to be able to build systems from standard reusable blocks. Electronic circuits today are almost always assemblages of standard components. Hardware designers never start entirely from scratch with only resistors and capacitors; to the contrary, they heavily use standard components ranging from simple parts such as logic gates, timers and multiplexers to complex functional blocks, in the form of integrated circuits (ICs). In software exactly the converse is true——developers largely start from scratch. The ability to build software systems by plugging in appropriate components——a plug-and-play approach to software construction——would be very valuable indeed. While it may be a trifle idealistic to expect that we will be able to achieve this hardware-like construction methodology in all software, by limiting the hard problem of general reuse to a smaller domain——of salvage within an organization, to be used in the construction of modest-sized systems (about 200,000 lines of code)——we try to achieve some benefits of the Software-IC model. (To address this domain would be to provide solutions of value to the small- and medium-sized software shops that have been resistant to adopt systematic reuse.)

1.3.1 SOFTWARE MATRIX

The “Software Matrix” is a framework that actively supports and promotes the reuse of components. (As mentioned earlier the form of reuse developed in this thesis is actually salvage, in that, rather than supporting the reuse of low-level components it focuses on major blocks of code.) In particular, it is a runtime infrastructure that acts as a substrate into which individual pieces of applications can plug. By employing generic fine-grained message-passing and mediator structures, and by supporting discovery of needed types, we’ve built a pluggable architecture that can gracefully adapt to salvage operations.

This infrastructure was named the Software Matrix in order to connote a very structured pattern of building software—— system elements are embedded in this substrate, a matrix, through which applications are dynamically composed. The image of endless banks of incubators of humans in a matrix, as from “The Matrix” [22] motion picture, isn’t entirely unintentional, if somewhat flippant.

Chapter 2

The Software Matrix

This chapter will start with an overview of the Software Matrix and will then go on to discuss specifically how it intends to help software salvage.

2.1 MATRIX OVERVIEW

A conceptual representation of the Matrix infrastructure is shown in Figure 1. As can be seen it is essentially an aggregation of different blocks of code. These code blocks are called “Cells”, and represent the unit of composition and reuse in our system, i.e. Cells are the building blocks out of which applications are built. The Matrix is designed to be a dynamic

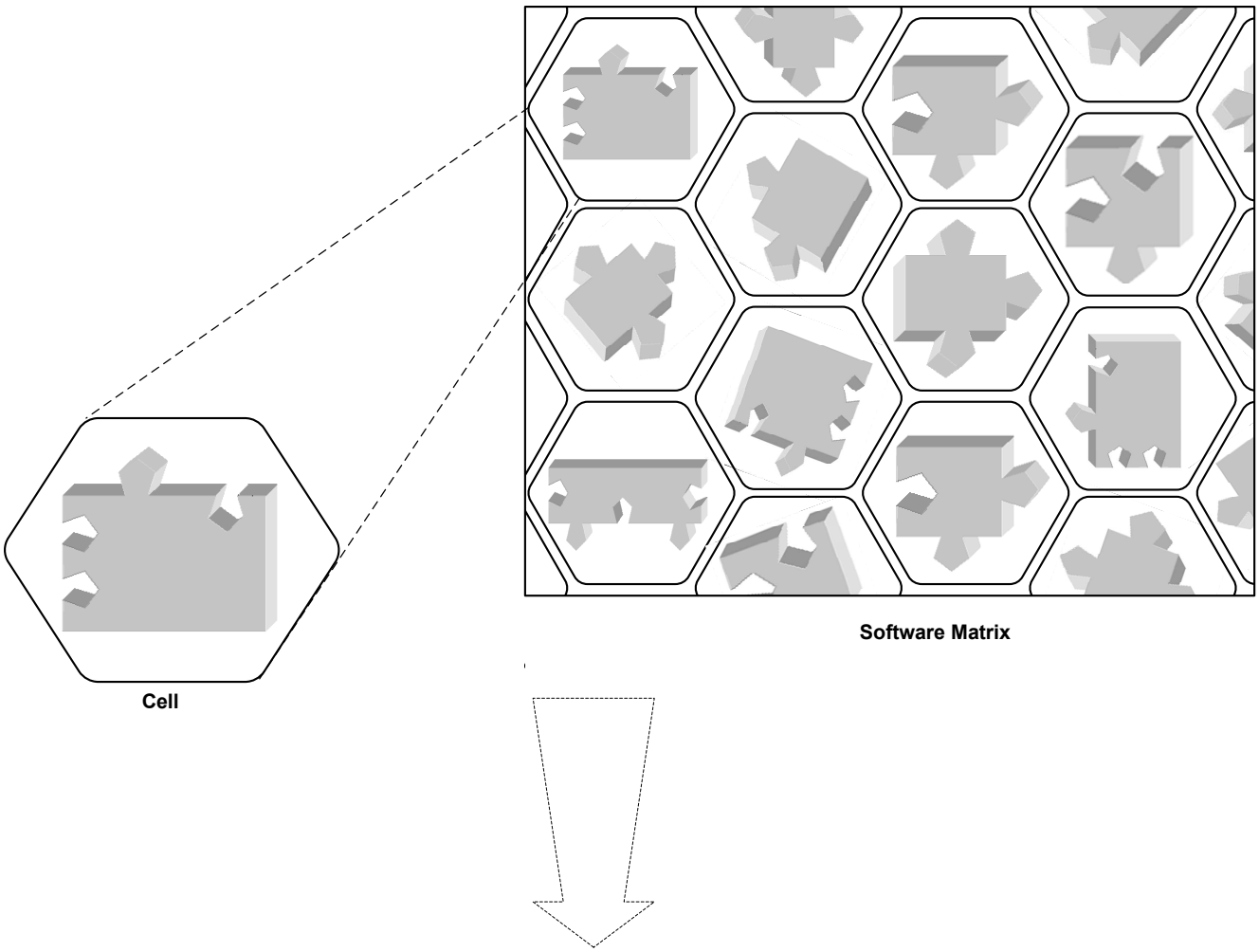

Figure 1: Conceptual representation of the Software Matrix

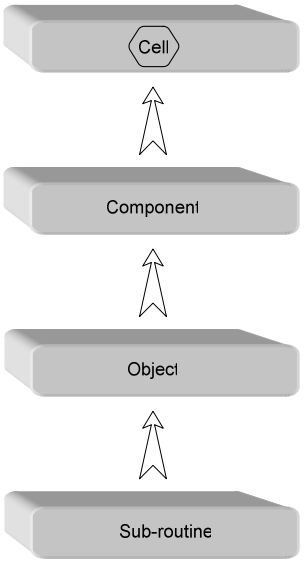

Figure 2: In the Software Matrix the unit of composition and reuse is a Cell.

collection—Cells can plug in and plug out anytime. An application is thus built from the collaboration of well-defined chunks of functionality. All Cells communicate with each other strictly through messages (our implementation uses XML encoded messages). One of the problems of salvaging has been the problem of extracting parts of monolithic (or very tightly-coupled) applications. By enforcing a separation of concern between Cells by inserting a message bus between them we are insisting on loose-coupling between different pieces of an application, which makes salvaging these pieces for use in future applications much easier. The Matrix also facilitates discovery of Cells by dynamically selecting the right pieces needed and composing them in order to build an application as shown in Figure 1.

2.1.1 CELLS

We now take a closer look at Cells, the building blocks of applications. A diagram representing the internal structure of a Cell is shown in Figure 3 below. Every Cell possesses the following:

- **a *message queue:*** Since collaboration between all Cells is strictly through message-passing, the message queue is used to hold request and response messages arriving at the Cell. Request messages are sent by other Cells that need the services of this Cell. Similarly response messages arriving at this Cell represent the results of processing carried out by other Cells on its behalf. The presence of a

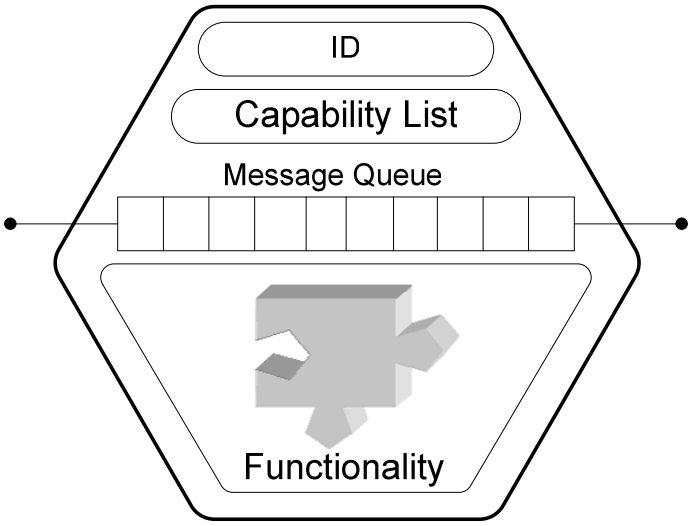

Figure 3: The internal structure of a Cell

message queue achieves temporal decoupling between the senders of request and response messages and their handlers.

- **a *capability list:*** Every Cell possesses a capability list that is used to advertise the capabilities of a particular Cell to other Cells via the Matrix. The capability list is essentially a list of all the message types that can be handled by a Cell. E.g., a Cell used for mathematical convolution may contain in its capability list the entry SU.Math.Convolution, indicating to others that it is capable of convolution operations. The capability list of Cells is used by the Matrix during the discovery process in order to locate and select the right Cells for system construction.

- **a *globally unique identifier (GUID):*** Cells can also be identified uniquely by a GUID. This identifier is used by the Matrix for several operations such as discovery and registration.

- ***functionality:*** In addition to all of the above Cells also, of course, contain the functionality that allows them to be considered as software assets that are candidates for reuse.

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 (communication could be synchronous, asynchronous or one-way). Also, every Cell has an entry-point, and is given a chance to execute on being plugged in. This decides whether a Cell will be a passive server or itself actively seek collaboration from other peers, depending upon whether the entry-point is empty or not.

2.1.2 MESSAGE-PASSING

As has been mentioned, message-passing has been chosen as the means of collaboration between different Cells in the Matrix. We list here briefly some of the advantages of a message-passing interface (MPI) as opposed to a procedure-call interface (PCI) in the Software Matrix:

- MPI is a universal interface. All entities using message-passing simply use something equivalent to GetMessage() and PostMessage(msg). So, from a calling perspective any Cell providing MPI is guaranteed to be compatible with other Cells accepting MPI, they can be plugged to each other. That doesn’t say that the plug results in the transfer of desired messages, only that the code will compile and the messages will flow. However

when we carve out code for salvage, that uses PCI, and couple it with anything, the changes usually are so overwhelming that it won't compile. Furthermore, to get to the compile stage requires lots of "surgery" on both sides. When you get to that stage, you are at the same place as MPI, i.e. compiles but perhaps the semantics are not right, but with no surgery.

- Messages are self-describing command and state carriers. If a receiver doesn't understand all the elements sent to it, it is free to ignore them and work with what it does understand. There is no such support with PCI.
- Since messages can be easily intercepted, and their contents inspected, message-passing systems are easier to debug.
- Message-passing makes it easier to use the Mediator design pattern [24], since the mediator does not have to know a lot of different interfaces, only the very small MPI. Using Mediator with PCI get very complex. This is completely avoided with MPI. We should say here that the Mediator pattern is significant in the design of the Software Matrix. Gamma et al. in [25] say "Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their

interaction independently." Proliferating interconnections between different partitions of a system reduces the potential for salvage since "lots of interconnections make it less likely that an object can work without the support of others". On introducing a mediator, which is one of the roles played by the Software Matrix, different entities know only the mediator, thereby reducing the number of interconnections.

2.2 SAMPLE APPLICATION

Let us consider a small application taken for the purpose of hypothetical example. Suppose we have an application whose job it is to read data samples from an input file, perform some signal processing operation on the input data (e.g., filtering), plot the results of the operations on the system display, and also log the results to a file. In this case one can conceive that the major pieces of the application would be responsible for file operations, signal processing and graphical plotting. In our approach these would be written as Cells, as shown in Figure 4, and plugged into the Matrix. The Matrix would then assume the responsibility of constructing the application from these individual Cells. In order to perform its intended task a Cell may need to use the services of other Cells. The capability

of a Cell, i.e. what set of operations it can perform, is defined by what messages it can handle, since all communication between Cells is through a lightweight message passing scheme. Cells that only exist for executive processing and do not handle any message types have null capability. The Matrix helps Cells discover other Cells that handle messages of a particular type—Cells do not explicitly bind to a particular Cell. All they specify is the type of message they need handled and the Matrix performs discovery to locate Cells (if any) that are capable of handling the requested message type. (If no suitable cell is found to handle a particular message type the discovery process will return “not supported”. The client Cell can still choose to continue execution if it thinks appropriate, if for instance, the particular piece of functionality it was looking for is not critical to the functioning of the system. Since we are not binding to a particular Cell at compile time, we can continue to run even if the piece we are looking for is not available.)

The just discovered Cell then processes the message it has received and ships the result back to the requesting Cell. In our example, the Cell responsible for executive processing needs the SU.FileOperations.Read, SU.FileOperations.Write, SU.DSP.PeakingFilter and SU.Graphics.PlotBarChart messages to be handled. The Cell that is capable of handling the SU.DSP.PeakingFilter Cell in turn needs the services of SU.Math.Convolution handler. (Message names or types are significant; they are scoped in a way that is similar to namespace scoping in programming languages. If more than one

Cell supports handling of the exact same message type, it is assumed that those Cells are indeed providing same functionality——which Cell is chosen can be decided by the Matrix depending upon certain policies.) So what we see here is dynamic system composition——we are building a program from pieces that exist on the Matrix. The Matrix automatically connects the right pieces at runtime without one having to bind to anything explicitly at compile time. This paradigm of software construction, with its dynamic nature and loosely coupled building blocks, is amenable to salvage operations. The very same Cells could be used to build other applications.

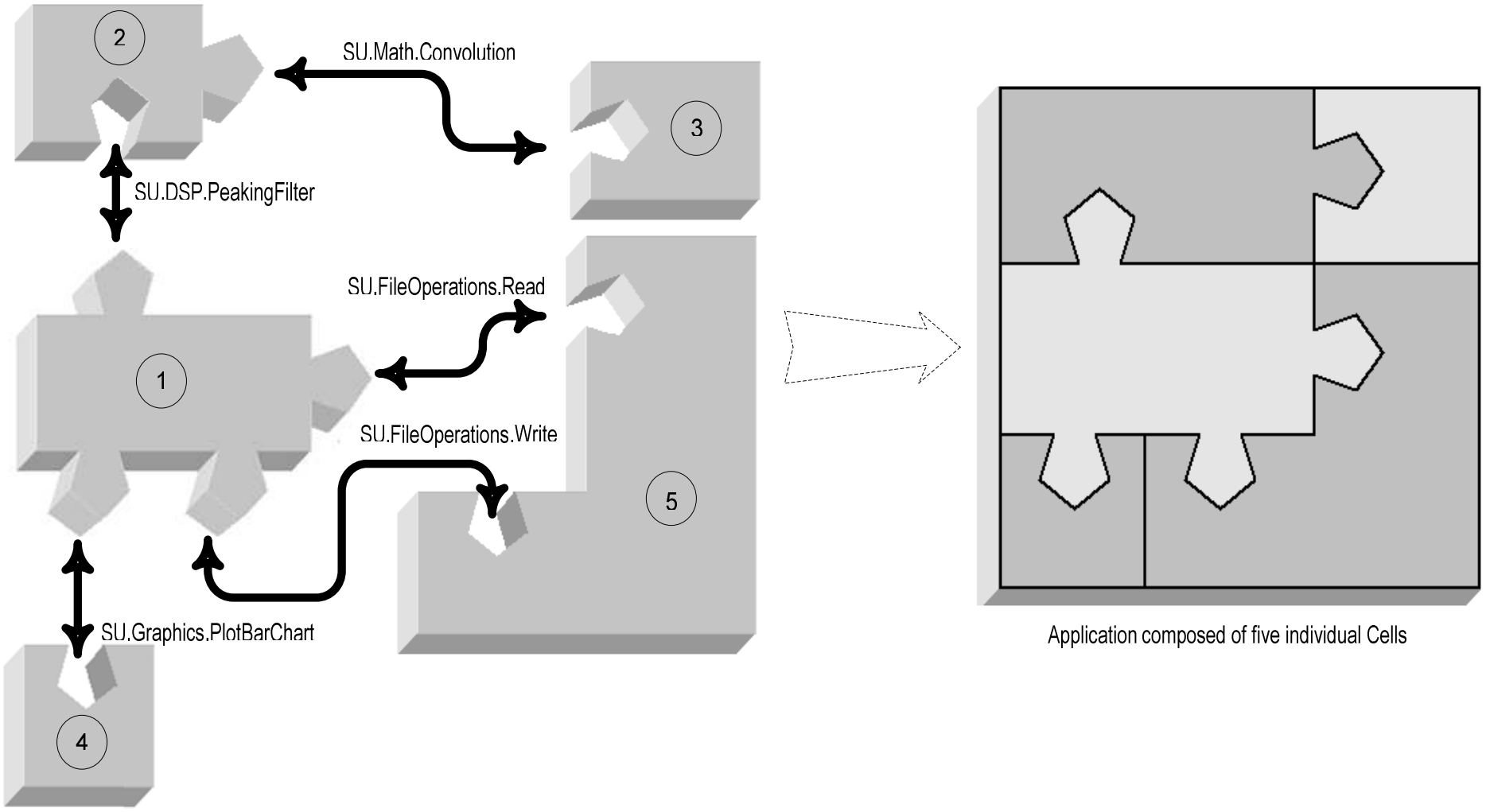

Figure 4: Composing an application from five individual cells

2.3 MATRIX EVENTS

The event trace diagrams that follow outline the sequence of events taking place during the construction of the above-mentioned sample application. The first diagram, Figure 5, represents a sequence of actions continuously being performed by the Matrix runtime infrastructure throughout its lifetime.

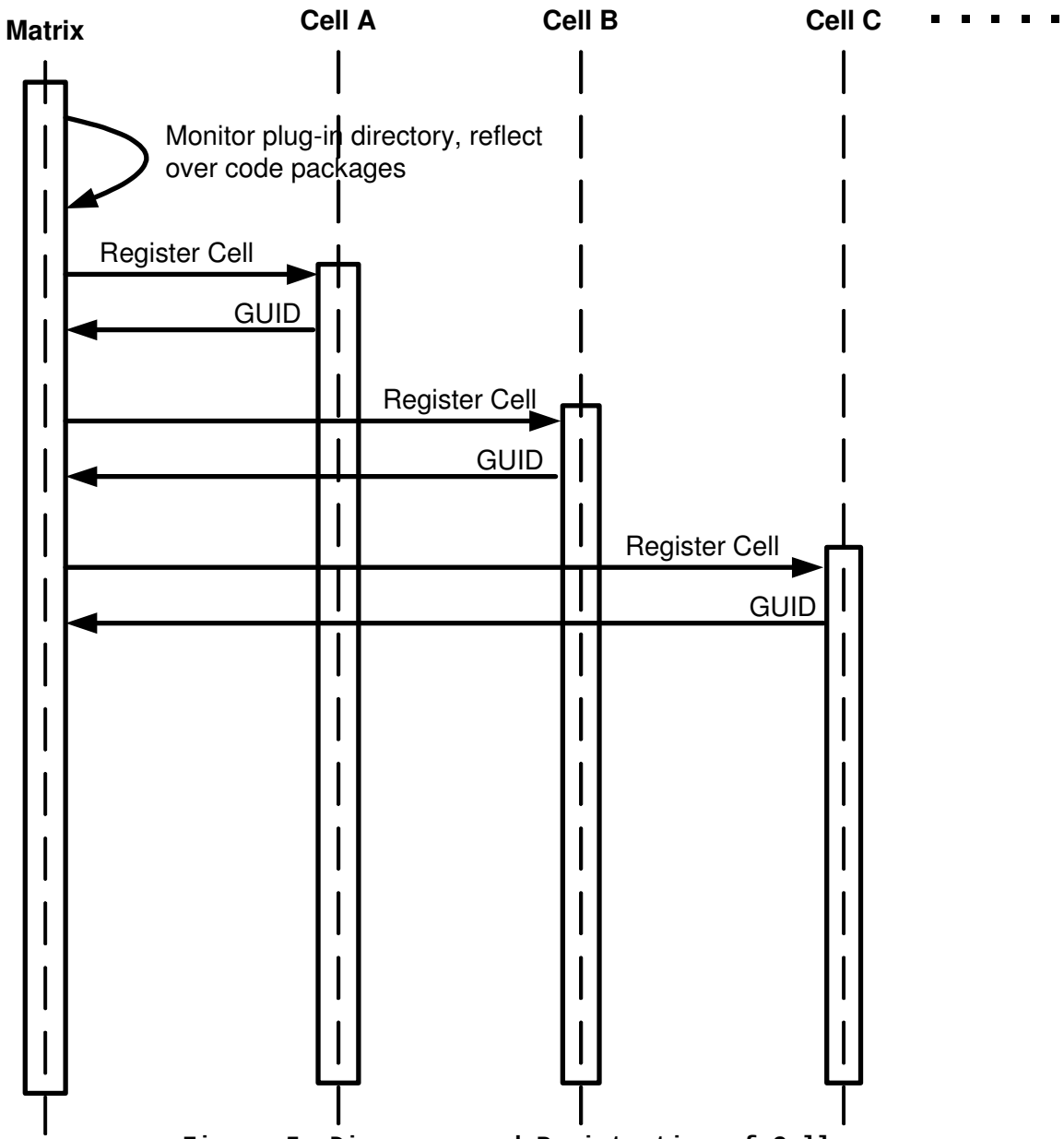

Figure 5: Discovery and Registration of Cells

At periodic intervals the Matrix interrogates a known location in order to discover if any new Cells are present. The Cells are physically deployed as plug-ins, and the Matrix looks for them in a predetermined plug-in directory. It performs type reflection in order to decide whether a piece of code is a valid Cell or not. The instant it determines a valid Cell is present, it registers the Cell with itself. This Cell, along with other Cells that the Matrix has discovered and plugged-in can then be used to build applications.

As seen in Figure 6, in this application the executive Cell (Cell 1) needs the services of Cells providing DSP filtering implementation (Cell 2), which in turn needs mathematical computational services from another cell (Cell 3). File operation (Cell 5) and charting (Cell 4) requests are also sent. Request messages, with necessary arguments are sent to the Matrix. On receipt the Matrix performs its discovery routine in order to locate a cell that is capable of handling that message type. On locating an appropriate cell, the Matrix forwards to it the message in question. The discovered cell then performs the required processing and ships the results back to the requesting Cell. (If there is no cell that can handle a message of a given type, the discovery process of the Matrix will return a “not supported” message, and the client can then choose to continue with its processing if it thinks appropriate.)

The entire process of collaboration between 2 cells is inherently asynchronous due to the message-passing paradigm adopted. Messages, whether request or response are dropped into

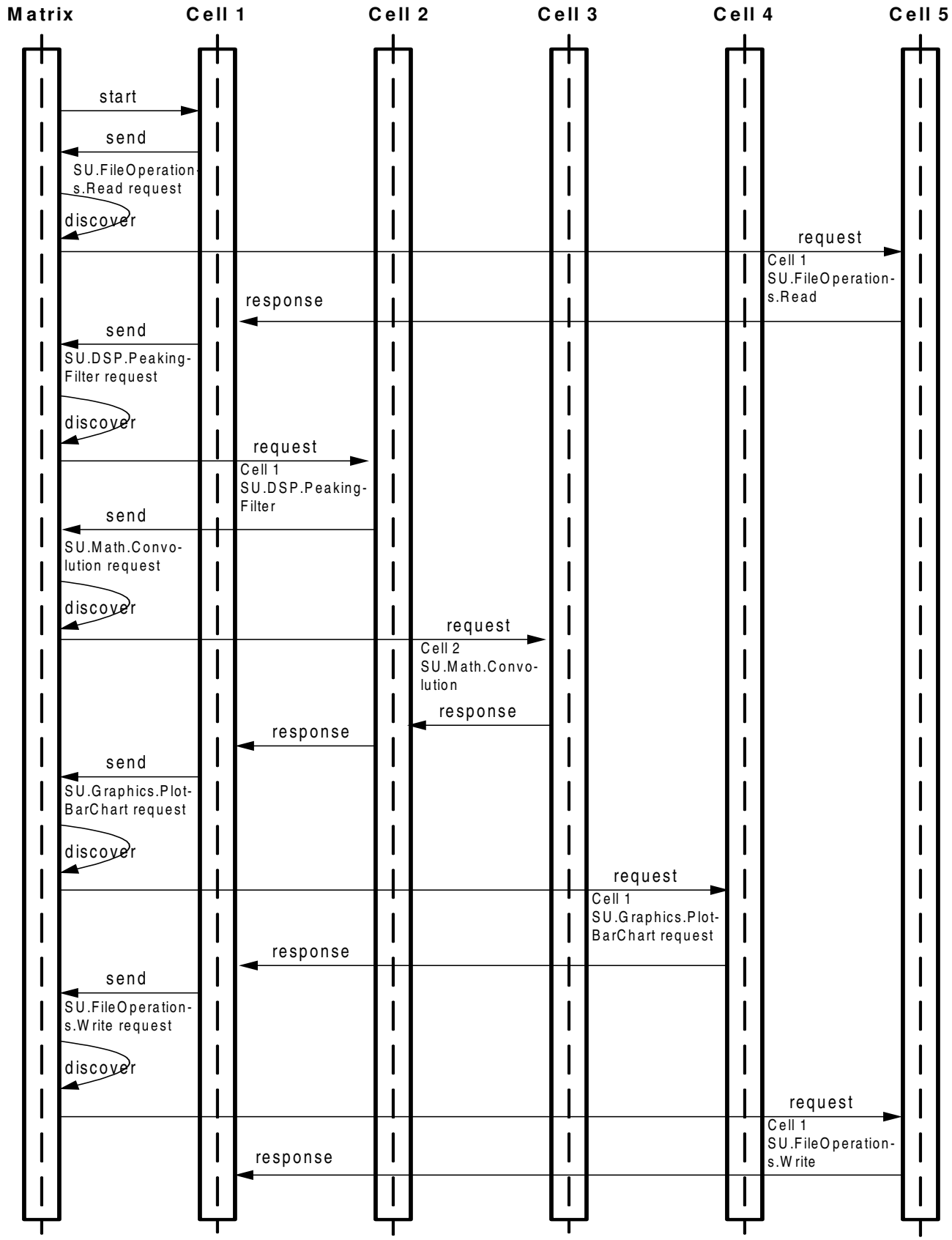

Figure 6: Message-passing collaboration between Cells

the message queues of the appropriate Cell. The Cell can then pull messages from this queue and process messages when it is free. This achieves temporal decoupling between the client and server Cells. In cases where the client cannot proceed further without having first received a response from the server Cell, synchronous behavior can be achieved. Cells possess facilities that allow them to wait efficiently while the response is being served. Another mode of operation is one-way operation typically envisaged for non-critical activities, where client Cells send out requests, but do not particularly care whether the request was successfully serviced or not, and hence no response is sent back to the client Cell.

Thus, the compositional aspects (as opposed to only computational aspects) of software are being taken care of by the Matrix; systems are being dynamically composed and constructed at runtime.

2.4 BUILDING SYSTEMS TO TAKE ADVANTAGE OF THE SOFTWARE MATRIX

The following section outlines the concrete steps needed to build systems by taking advantage of the Matrix.

In order to enable salvage, pieces of an application (at the time of writing it, or existing pieces) are wrapped in a Cell. All Cells conform to the ICell protocol. E.g., if we are writing a FileManager Cell responsible for common file

operations we create a FileManager class that inherits from ICell.

```
FileManager : ICell
{
    ...
}
```

The Capability List of this FileManager Cell is then populated to indicate the types of messages it is capable of handling.

```
CapabilityList.Add("SU.FileManager.Files.Read");
CapabilityList.Add("SU.FileManager.Files.Write");
CapabilityList.Add("SU.FileManager.Files.Search");
CapabilityList.Add("SU.FileManager.Files.Compression");
```

Next one needs to specify what is to be done in response to messages in the Capability List, i.e. we need to appropriately delegate calls to the code that actually implements functionality required to handle the message. This can be done by providing a suitable implementation for the process() method of ICell.

The resulting binary after compilation is then copied into the "plug-in" directory of the Matrix, from where it is automatically detected and registered, and is available for composition.

In the Matrix everything is a Cell——even a program executive is a Cell. This simplifies the model, since then everything can be treated the same way. A program executive however will most probably have an empty capability list, and

will use its entry-point (the start() method of ICell) to attempt to use the services of other Cells. The following statement is an example of a Cell trying to locate a Cell that can handle the SU.FileManager.Files.Search message type, with input parameters being specified by Params:

```
Result = syncSend("SU.FileManager.Files.Search", Params)
```

Cells that are solely meant for executive processing, and that handle no messages (Cells that have empty capability lists) serve no purpose lying around in the Matrix after they have completed execution, i.e. after its start() method has returned, and are therefore unplugged on completion.

2.5 FEATURES OF THE SOFTWARE MATRIX

We now discuss some of the key features of the Software Matrix. In addition to allowing the Matrix to effectively support salvage operations they also have other benefits as discussed.

- **Fine-grained message-passing**

  As discussed in Chapter 1, most of the problems associated with salvaging existing parts of a system and building new systems from them arise from the difficulty of extracting one of several tightly-coupled pieces from an existing application. By mandating the use of message-passing at a much finer level of granularity than is normally seen, i.e. by requiring that messages be the only mode of collaboration between different parts of an application, we are able to

decrease the degree of coupling, as compared to a system built using only the procedure-call model of collaboration. (While surely this decision will have performance implications, we are of the view that it is a tradeoff worth making, especially with additional hardware capability at our disposal due to the ever increase power of computing devices as dictated by Moore's Law). Explicitly inserting a message-bus also leads to a rigor in the development of Cells that help make them loosely coupled. This loose coupling is critical to the success of salvage operations.

- **Dynamic Composition**

  The Matrix takes care of the compositional aspects of software by automatically discovering and connecting the right pieces and building an application at runtime. By adding a few more Cells into the Matrix if needed we can build new applications with the Matrix composing the application for us by reusing existing Cells. This supports a growing environment.

- **Support for System Evolution**

  Since the appropriate Cell to serve a message of a particular type is selected at runtime, the Matrix supports evolution of software systems. Newer versions of Cells easily replace older versions. Evolving

requirements can be accommodated by easily unplugging or modifying only those Cells that represent the affected part of the application. It is easy to field replace components since all that has to be done is that the new Cell has to be copied over the old Cell in the plug-in directory of the Matrix. The Matrix will then register the newer version of the component. The Software Matrix is therefore an effective way to support program maintainability. Bug fixes/upgrades simply need to be copied into one of Matrix's known paths and they become instantly available to all applications. (This is the default case. However we imagine there may be a case where a Cell may still want to use a specific Cell instead of using the newest version. This hard-wired composition, though not recommended, is also supported by the Matrix. The discovery mechanism of the Matrix is by-passed and a Cell can choose to send messages and collaborate with a particular Cell, identified by its GUID.)

- **Simplicity**

  One of the goals of the Software Matrix was to provide a simple solution to make salvage a useful paradigm. It is a supporting infrastructure that is lightweight (the bare infrastructure is roughly only 2000 lines). The infrastructure implementation comes with full source code that is well documented and easy to understand for

those who would like to be familiar with the inner workings of the system. For others, sample applications and documentation make it easy to use the Software Matrix. In later chapters we will demonstrate this simplicity in leveraging the Matrix for salvage operations. Moreover the underlying component technology used is the .NET component model, which is considered superior [26] to other component models such as COM that have been know for their complexity (.NET components are considered easier to build and consume). Simplicity, as opposed to “high-ceremony” techniques prompted a lot of our design choices, which are elaborated in later chapters.

2.6 COMPARISON WITH PRIOR AND RELATED WORK

Some of the earlier attempts to promote the reuse of software assets have been discussed in Chapter 1. Object oriented technologies, that were expected to bring about a reuse revolution, did not deliver. They concentrated largely on programming language features such as inheritance and parameterization as enablers of reuse. The notion of an object had no direct mapping to the actual physical unit of code that was versioned, deployed and potentially reused. Component oriented technologies attempted to address this shortcoming. There exists today a combination of component and middleware technologies such as COM and CORBA that have reusability as one

of their goals. The Matrix differs from these in several key areas. Both COM and CORBA are considered to be highly complex, over-specified and require heavyweight supporting infrastructures. In comparison the Matrix was meant to address the specific problem of salvage, and is lightweight (the bare infrastructure implementation is only approximately 2000 lines of source code), is easy to use, and comes with full source code which is well documented and easily comprehensible, if required. Moreover most middleware technologies use the procedure-call model of collaboration. In our view having components bind to exact function signatures is not flexible enough, and leads to the very tightly coupled systems that make salvage operations difficult. The Matrix uses message-passing as the mode of collaboration between components, i.e. Cells——the unit of composition in our system. This leads to looser coupling of system elements, resulting in easier salvage in the future. Message oriented middleware (MOM) and to a certain extent Web Services also use message passing. (Most Web Services use XML-RPC, though they do support a message-passing abstraction over RPC). However the Matrix differs from them in scale and granularity of message-passing. The Matrix proposes to use message-passing at a much finer level of granularity than is used in Web Services and MOM applications, since we are focused on local compositions of Cells to build applications, rather than accessing remote functionality over the network.

In the Software Matrix a combination of techniques is being used to solve a problem that has not been solved before——

we are significantly managing the problems associated with software salvage.

Chapter 3

System Design

This chapter discusses the architecture, design and briefly considers the implementation details of the Software Matrix. Microsoft's .NET Framework and the C# programming language were used in this implementation.

3.1 PARITITIONING

The module diagram of the Software Matrix, showing the various partitions of the system, appears on the next page. A brief description of the different partitions follows:

- **Matrix Executive**

  This module serves as the executive module of the Software Matrix, and co-ordinates and uses the services of

all the helper modules in the system. It also provides the administrative console interface which is meant to display

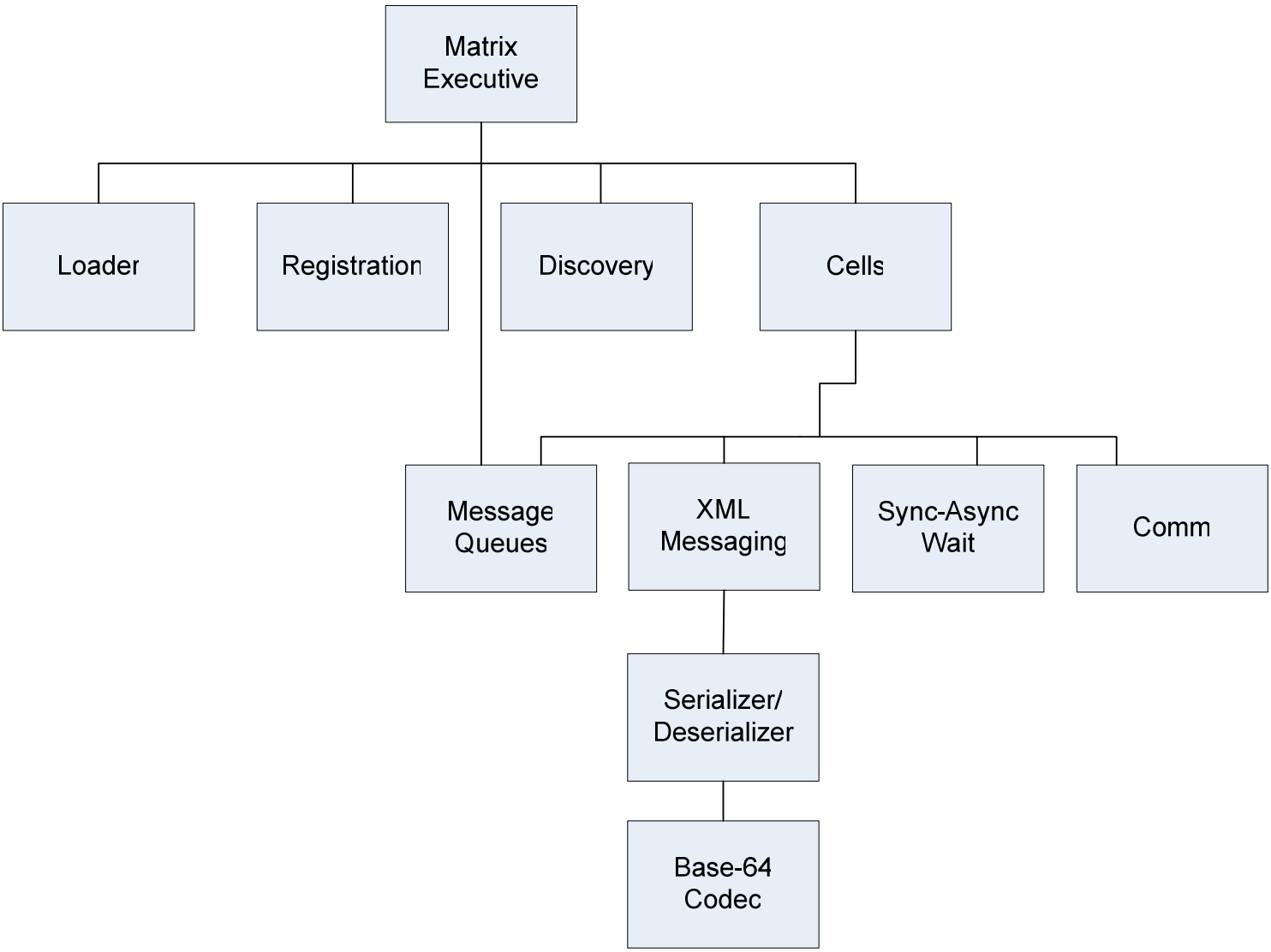

Figure 7: High-level partitioning of the Software Matrix

status and error messages relevant to the operation of the Software Matrix.

- **Loader**

The Loader module is responsible for identifying and loading valid Cells into the Software Matrix. At periodic intervals, the Matrix Executive calls the Loader module which interrogates a known location in order to discover if any new Cells are present. The Cells are physically deployed as plug-ins, and the Loader looks for them in a predetermined plug-in directory. It performs type reflection on code packages in order to decide whether a piece of code is a valid Cell or not (Here a code package implies a .NET assembly, which is the packaging, deployment and security unit in .NET [27]). It thus identifies Cells that are suitable to be plugged-in to the Matrix and used for system construction.

- **Registration**

The registration of valid Cells in the Software Matrix is done by this module. This involves associating a globally unique identifier with each Cell, so that it is reachable if needed, by the Matrix.

- **Cells**

This module represents the Cells in the system which are the actual unit of composition and reuse. They encapsulate the functionality that makes them potential candidates for reuse. All Cells subscribe to a common protocol that specifies:

- how to register with the Software Matrix, and associate itself with a unique identifier
- the entry point of execution of a Cell, that will be called when a Cell is first loaded, and gives Cells the opportunity to collaborate with other Cells.
- how to unregister the Cell from the Software Matrix so that it is no longer available to other Cells for system construction.
- how to enqueue request and response messages for processing within a Cell
- how to send execution request messages to another Cell, which is the means of collaboration is the Software Matrix.
- how to interrogate of a Cell what it is capable of, in terms of what message types it can handle. This will be used by the Matrix to discover and select the right Cells for system construction.

- how to return a clone of a Cell, to be used by other Cells that wish to have stateful collaboration
- what is to be done in order to handle different messages that arrive at a Cell

- **Discovery**

  This module is responsible for performing discovery operations to locate Cells of specified capability. The capability of Cells in the Matrix is described in terms of what message types they can handle, through their capability list. The Discovery module uses these capability lists in order to determine which Cell, if any, possesses the specified capability.

- **XML Messaging**

  The architecture of the Software Matrix defines the entire communication between cells be via XML formatted messages. The requests, responses or other data are all encapsulated in messages. All messages are sent in a pre-determined format, which allows the receiver to decode message contents. The XML Messaging module is responsible for generating (sender side) and interpreting (receiver side) the well-formed XML messages being passed in the Software Matrix. The figures on the next page show sample request and response messages in the Software Matrix.

This module uses the services of the Serializer/Deserializer and Base-64 Codec modules in order to encapsulate data such as binary objects within XML messages.

```
<?xml version="1.0" encoding="utf-8" ?>
<Message>
  <Type>Request</Type>
  <RequestID>f705da1e-4fc7-4c24-86b9-335540d0578d</RequestID>
  <RequestCellGUID>21c34674-6fe5-402d-892d-1fc7a0c34d60</RequestCellGUID>
  <ResponseCellGUID>NOT_SET</ResponseCellGUID>
  <MessageType>
    <Name>GUI.DrawBox</Name>
    <ReturnVal>
      <ReturnType>Not_Needed</ReturnType>
      <ReturnValue>NOT_SET</ReturnValue>
    </ReturnVal>
    <NoOfParams>1</NoOfParams>
    <Param>
      <ParamType>System.String</ParamType>
      <ParamValue>AAEAAAD/////AQAAAAAAAAAGAQAAAC5QZXJmb3JtaW
      5nIGNvbnZvbHV0aW9uIG9mIDIgZGlzY3JldGUgc2VxdWVuY2VzCw==</ParamValue>
    </Param>
  </MessageType>
</Message>
```


Figure 8: Sample Request Message

```
<?xml version="1.0" encoding="utf-8" ?>
<Message>
  <Type>Response</Type>
  <RequestID>79ca4626-2d44-4107-ae49-6b2b8ea89ccb</RequestID>
  <RequestCellGUID>7ce10f90-e984-4a79-b4b2-3dacdb38aee8</RequestCellGUID>
  <ResponseCellGUID>1e37b7dc-6007-46b2-9638-249e8068347c</ResponseCellGUID>
  <MessageType>
    <Name>Math.Add</Name>
    <ReturnVal>
      <ReturnType>Not_Needed</ReturnType>
      <ReturnValue>AAEAAAD/////AQAAAAAAAAAMAgAAAERNYXRyaXgsIFZlcnNpb249MS4wLjE3ODQuMzQ3Nj
      AsIEN1bHR1cmU9bmV1dHJhbCwgUHVibGljS2V5VG9rZW49bnVsbAUBAAAAE01hdHJpeC5
      SZXR1cm5PYmplY3QCAAAAC1JldHVybIZhbHVlCWV4Y2VwdGlvbgIDEFN5c3RlbS5FeGNlcHRpb2
      4CAAAACAa8V61M+F0xQAoL</ReturnValue>
    </ReturnVal>
    <NoOfParams>2</NoOfParams>
    <Param>
      <ParamType>System.Double</ParamType>
      <ParamValue>AAEAAAD/////AQAAAAAAAAAEAQAAAA1TeXN0ZW0uRG91YmxlAQAAAAdtX3ZhbHVlAAbOG
      VHaG3wUQAs=</ParamValue>
    </Param>
    <Param>
      <ParamType>System.Double</ParamType>
      <ParamValue>AAEAAAD/////AQAAAAAAAAAEAQAAAA1TeXN0ZW0uRG91YmxlAQAAAAdtX3ZhbHVlAAaSIj
      Ks4n0oQAs=</ParamValue>
    </Param>
  </MessageType>
</Message>
```


Figure 9: Sample Response Message

- **Serializer/Deserializer**

  This module is responsible for serializing data such as arguments in requests and return values in responses, so that the resulting persisted data can be encapsulated in XML messages to be passed around in the Software Matrix. The reverse process of deserialization in order to extract data from a persisted stream is also carried out by this module.

- **Base-64 Codec**

  This module converts binary serialized data into the Base-64 encoding so that it can be easily encapsulated within XML tags, and transmitted through the Software Matrix. The reverse process, of decoding base-64 encoded data to its corresponding binary format, is also carried out by this module.

- **MessageQueues**

  The MessageQueues module is responsible for maintaining queues used for messaging. Both the Matrix and the individual Cells use messages to interact with the outside world, and queues are used to store these messages (e.g. request or response messages). By introducing queues between a client Cell and a server Cell we achieve temporal decoupling between the two. The queue acts as a buffer for irregular or bursty generation of requests from the client and ensures that no requests are lost when the server is busy doing something

else. The queues implemented by the MessageQueues module are blocking queues, i.e. the threads servicing them are blocked in an efficient wait state whenever the queues are empty.

- **Sync-Async Wait**

  The collaboration between Cells in the Software Matrix can be synchronous, asynchronous or one-way. While collaboration between Cells is inherently asynchronous, we can have synchronous collaboration, where the client cell blocks until it receives a response from the server Cell can be implemented. This is done by the Sync-Async Wait module, by using the .NET event and notification mechanisms so that clients can wait efficiently until a response is received.

- **Comm**

  The Comm module is responsible for the transfer of messages between cells. Provided the right cells have been discovered it simply relays messages (request or response) from one message queue to the other. The fact that the Comm module is able to act as a mediator between 2 Cells makes possible other applications of the Software Matrix, as shall be discussed later.

3.2 ACTIVITIES

The activity diagrams below represents the major high-level activities in the Software Matrix and documents the overall flow of system.

3.2.1 LOADER ACTIVITY DIAGRAM

The activities involved in plugging-in valid Cells into the Matrix are shown below.

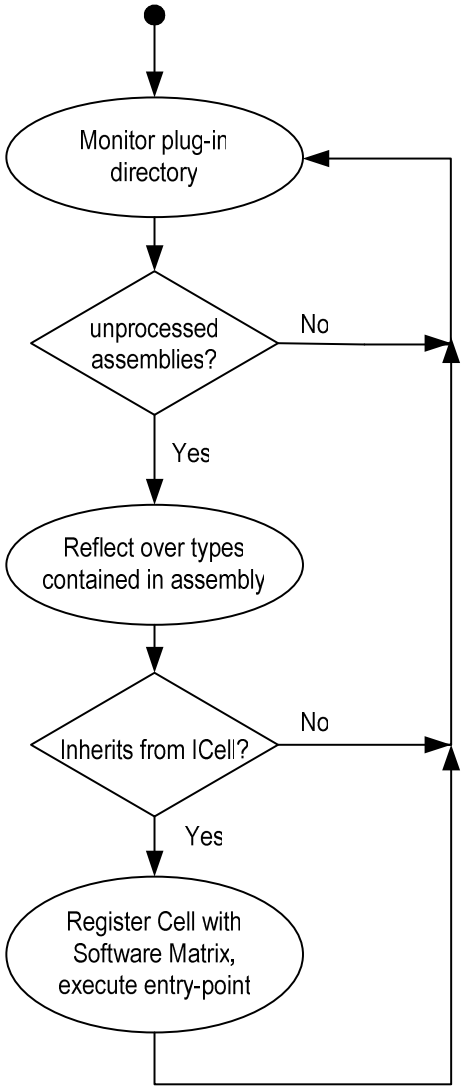

Figure 10: Loader Activity Diagram

- On startup, and at intervals throughout the lifetime of the Matrix, a known location——the plug-in directory——is monitored, for the presence of .NET assemblies, using Loader.monitor().
- Anytime a new assembly that has not already been processed by our system is seen, the Loader module reflects over all the types contained in the assembly to determine if it is a valid Cell. One could have identified valid Cells on the basis of a presence of an attribute, or a base type. We choose the existence of a parent interface to recognize valid Cells——every Cell derives from the ICell interface. We use the System.Type.IsAssignableFrom() method in .NET Framework Class Library for this purpose.
- If a valid Cell is indeed found, ICell.register() is called on that Cell, in order to complete the registration of the Cell with the Matrix, which associates with every Cell a globally unique identifier. Also the Cell is given a chance to execute, if it so wishes, by calling the ICell.start() method.
- If no new assembly is found, or if an assembly is found but that does not derive from ICell, the loader simply continues its monitoring process.

3.2.2 SEND MESSAGE ACTIVITY DIAGRAM

The activities involved in a message send operation, are described below:

- A send operation involves specifying the message type and the arguments that are to be delivered.
- A message in the correct XML format, with arguments serialized and base-64 encoded is then created.
- This well-formed message is dropped in message queue of the Matrix.
- The Matrix pulls out this message out of its queue, and performs a discovery operation based on the specified message type. The capability lists of active Cells in the Matrix are used to discover the appropriate Cell.
- If a suitable Cell is not found, a “message not supported” exception is generated. If a suitable Cell is found, the message is dropped into the request queue of the discovered Cell.
- The Cell pulls out messages from its request queue, for processing, in the order they were received. The contents of the message, such as its arguments are extracted after parsing the XML formatted message, including deserialization and base-64 decoding.
- The arguments are then processed by the Cell, according to the message type, i.e. a suitable handler for the message is executed.

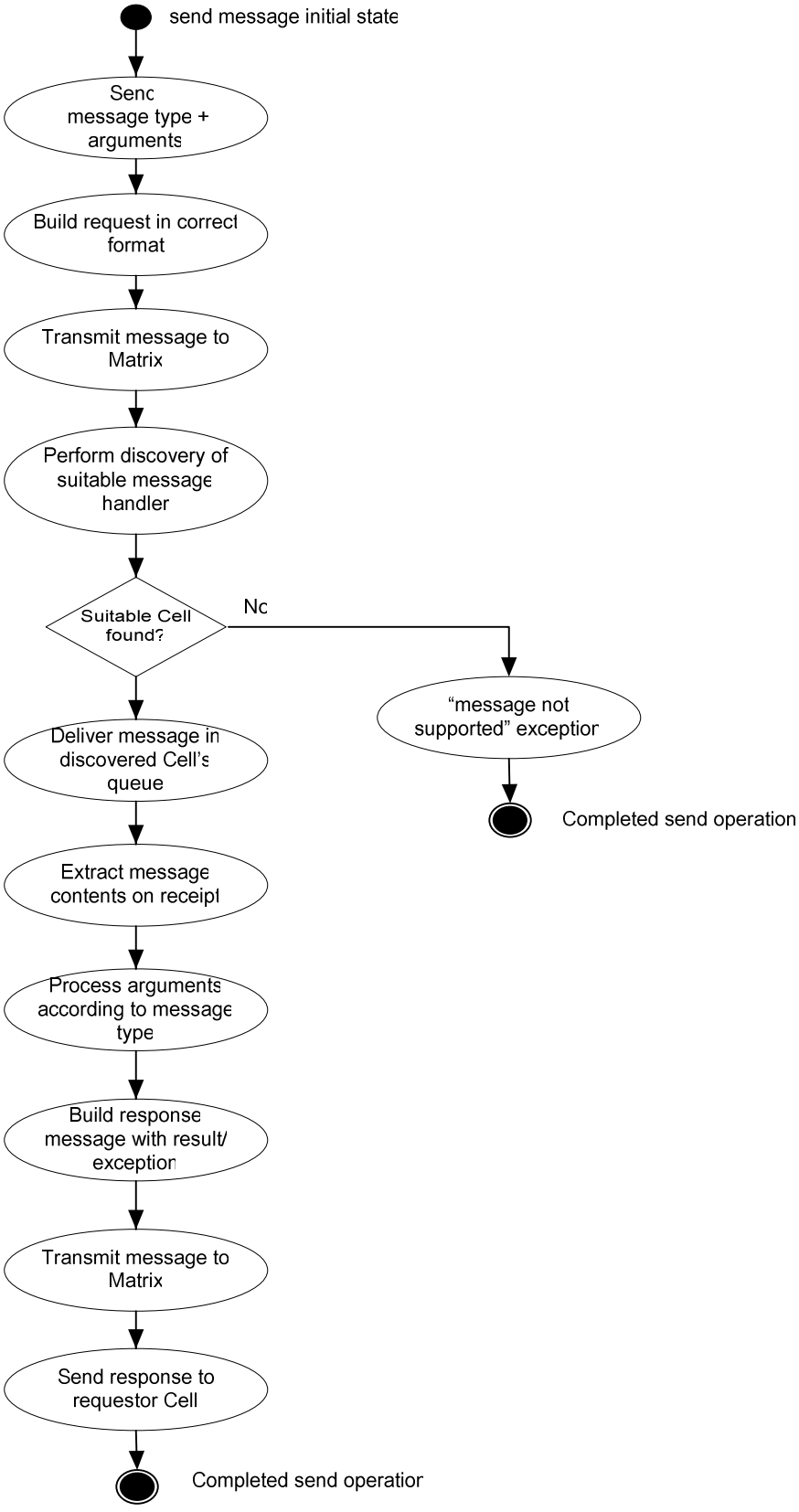

Figure 11: Send Message Activity Diagram

- A response message is then built in XML format with content such return values (or exceptions) being serialized and base-64 encoded.
- This message is then transmitted to the Matrix, which will drop the response in the requestor Cell’s message queue. The response is thus made available to the requestor.

3.2.3 CLASS RELATIONSHIPS

The class diagram (OMT) below shows the major classes used in the Software Matrix and gives us the relationship between them. It also gives an idea of the operations of a class.

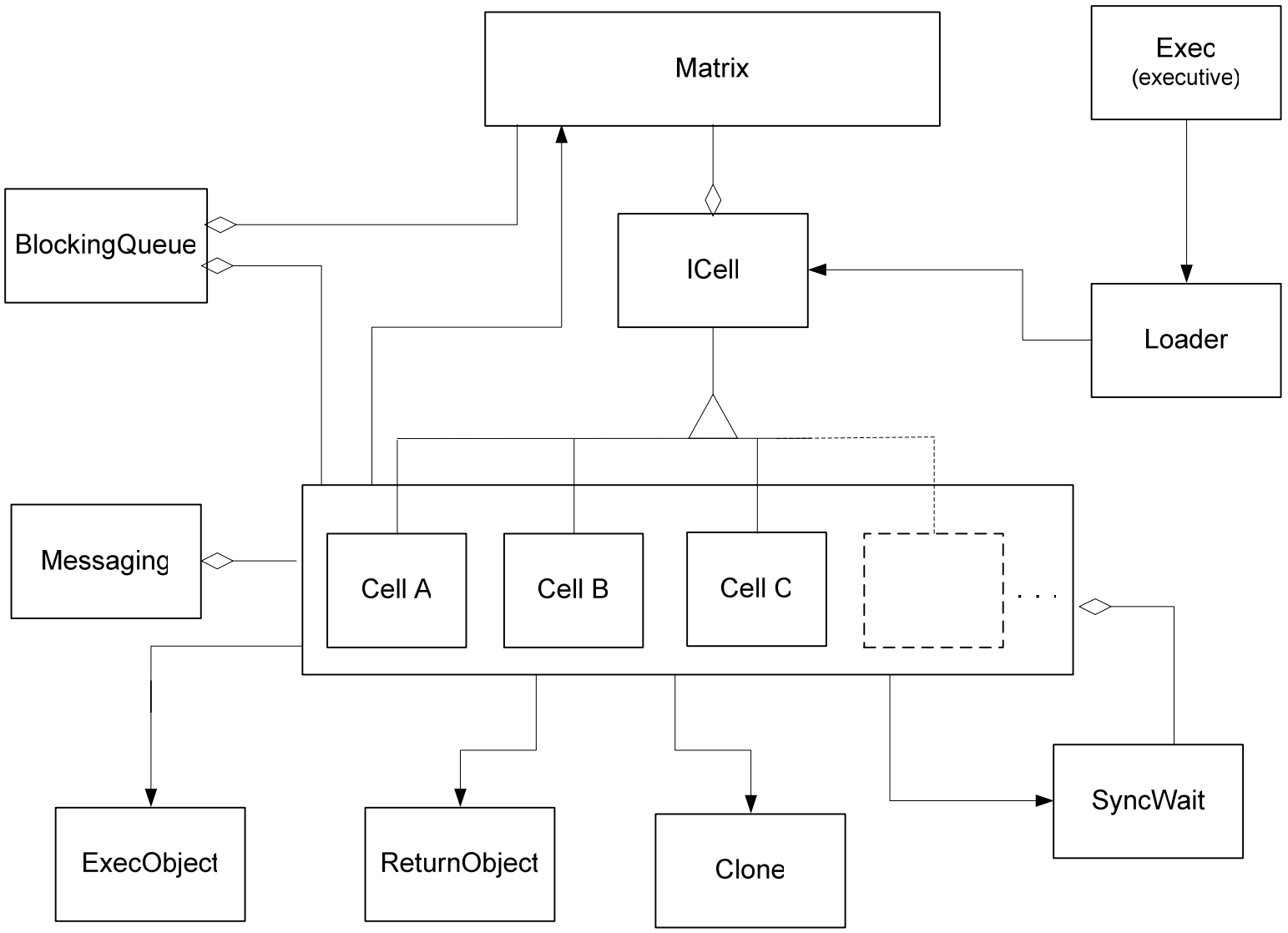

Figure 12: Software Matrix Class Diagram

The **Exec** class is the executive of the Software Matrix. It also manages the interface to the user, in the form of an administrative console interface which is meant to display status and error messages relevant to the operation of the Software Matrix. On start-up the executive uses the services of the Loader class.

The **Loader** class provides services that identify and plug-in valid Cells in the Software Matrix. At periodic intervals, the Loader.monitor() function checks the system “plug-in” directory for the presence of .NET assembly files. This function uses the operations provided by System.Type and in the System.Reflection namespace of the .NET FCL to reflect over the types contained in these assemblies, and identify valid Cells.

The **Matrix** class represents the substrate where all the cells plug-in to. The Matrix class therefore maintains the collection of all the active Cells in the system and provides as services many operations commonly performed in the system such as registration (Matrix.register()) and de-registration (Matrix.unregister())of Cells and also facilities their discovery (Matrix.discover()) and communication (Matrix.send()). The Matrix has as a data member an instance of the BlockingQueue class——Cells wanting to communicate with the Matrix do so by enqueuing messages (Matrix.transmit())in this queue.

The **BlockingQueue** class, as mentioned, is responsible for maintaining queues used for messaging. Both the Matrix and the individual Cells use messages to interact with the outside world, and queues are used to store these messages (e.g. request

or response messages). The queues implemented by this class are blocking queues, i.e. the threads servicing them are blocked in an efficient wait state whenever the queues are empty. The services provided by the AutoResetEvent class in the .NET framework is used in the implementation of these queues. The class provides a minimal interface in the form of BlockingQueue.enQ() and BlockingQueue.deQ() for enqueue and dequeue operations respectively.

The **ICell** interface is the one from which every valid Cell in the system must derive from. It defines the protocol that must be followed by all the Cells in the Software Matrix. The actual Cells that are concrete classes that derive from ICell are provided default implementations of all the common functionality required by the ICell interface by means of a wizard. The wizard generates the implementation code, and provides placeholders where specific implementation for a particular Cell is required.

The important methods of the ICell interface are as shown:

- **bool ICell.register()**

  This method registers a Cell with the Software Matrix and associates it with a GUID. Also gives Cells a chance to execute, if they so desire, by calling the cell entry-point on a new thread of execution.

- **bool ICell.unregister()**

  This method unregisters the Cell with the Software Matrix so that it is no longer available to other Cells for system construction.

- **void ICell.accept(string Message)**

  This method is used to enqueue request or response messages in the appropriate queue of a Cell, for processing.

- **Clone ICell.getClone(string MessageName)**

  This method returns a clone of a Cell, to be used by other Cells requiring stateful collaboration with this Cell.

- **Object ICell.syncSend(string Message, Object[] Params)**

  This method is used to send an execution request message to another cell. Blocks efficiently till a response corresponding to this request has arrived. This is the primary means of collaboration in the Software Matrix.

- **void ICell.start()**

This is the entry-point of execution of a Cell. It gives Cells the opportunity to collaborate with other Cells.

- **bool ICell.queryCapability(string Capability)**

  This method is used to interrogate of a Cell its capabilities, and used by the Matrix in the discovery process in order to locate and select the right Cells for system construction.

- **Object ICell.process(ExecObject obj)**

  This method is used by Cells to specify what is to be done in order to handle different messages that arrive at a Cell, i.e. this method appropriately delegates calls to the code that actually implements the functionality required to handle a particular message type.

Cells communicate with each other through message queues and therefore the composition relationship with BlockingQueue. In order to generate and interpret messages in the correct format, Cells compose an instance of the Messaging class. Cells also use the **ExecObject** and **ReturnObject** classes, which assist in packaging of arguments, return values and other information regarding requests responses between cooperating Cells in the Software Matrix. Cells use the services of the Matrix class which provide implementations of common tasks. Cells also use the services provided by the Clone and SyncWait classes, the purposes of which are described later in this section.

The **Messaging** class provides services for the generation of well formed XML messages according to a pre-determined message format(Messaging.buildRequest(), Messaging.buildResponse()), and also for interpreting messages in this format to extract the data contained within them (Messaging.extract()). It also provides the additional functionality of serializing data objects such as arguments in requests and return values in responses, so that the resulting data can be encapsulated in messages. The reverse process of deserialization in order to extract data from a serialized stream is also carried out by this module. In order to easily encapsulate binary data within XML tags, base-64 encoding and decoding functionality is also carried out by this class (Messaging.textSerialize() and Messaging.textDeserialize()).

A Cell can serve requests from several other Cells——a cell picks up requests from its message queue, and services them in the order they arrive. This can lead to an interleaving of requests from different Cells. However there may be situations where a Cell may wish to have stateful collaboration with other Cells; in other words the state of the server Cell serving a particular client needs to be maintained between multiple requests from the same client. To address this scenario and provide for stateful collaboration, the Matrix is capable for providing a “clone” of a particular Cell to a client Cell. The client can then collaborate with this clone. The **Clone** class is

used for this purpose and can be used to refer to a particular clone of a Cell.

The **SyncWait** class is used for synchronous collaboration between Cells.

The request-response exchange could be asynchronous. It could also be “one-way”, such as in non-critical cases where the client does not care about the particular result of a request operation. In synchronous collaboration the client Cell blocks until it receives a response from the server Cell. The SyncWait class is used to implement efficient blocking. Its methods SyncWait.subscribe(),SyncWait.unsubscribe(), SyncWait.Check() and SyncWait.wait() use .NET event and notification mechanisms such as AutoResetEvent, to ensure clients wait efficiently until a response is received.

Chapter 4

Assessment

This chapter provides an assessment of how effectively the Software Matrix supports salvage operations, when applied to the construction of a software system.

Apart from several “toy” examples, the Software Matrix was also used in the construction of real world applications. In this chapter we present two applications that were built by adopting the “Matrix” approach.

4.1 FILE SYNCHRONIZER

The File Synchronizer application is a useful tool that allows for remote directory synchronization. In other words, it allows the contents of a directory on one machine to match those

present on a specified directory on a remote machine, so that we always have the latest versions of a specified set of files. Either the remote or local machine directories can be brought up to date with respect to the other.

The File Synchronizer application has a peer-to-peer architecture——instances of File Synchronizer running on different machines over the network are exactly identical and can serve in both client and server roles.

Files that are present on the other side, but not on the side being synchronized, are copied. Newer versions of files, update older versions. Older files will not overwrite newer files, and explicit permission from the user is sought in this case.

4.1.1 BUILDING THE FILE SYNCHRONIZER APPLICATION

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:

- the file upload and download block, for transfer of files to and from remote machines on a network
- the file synchronization policy block, that decides what files to transfer, update, overwrite or delete, based on decided rules

- the user interface that allows the user to operate the application, and presents him with information such as status, error and confirmation messages.

These 3 blocks were each built into an independent Cell, containing the appropriate functionality. In order for the different Cells to collaborate with each other to construct the whole application, it was decided what the capability lists for each Cell would look like, i.e. what message types the different Cells would handle. The Cells were built by incorporating the wizard generated "boiler-plate" code for a generic Cell (in addition to the actual functionality the Cell was responsible for) and certain Cell-specific implementation code.

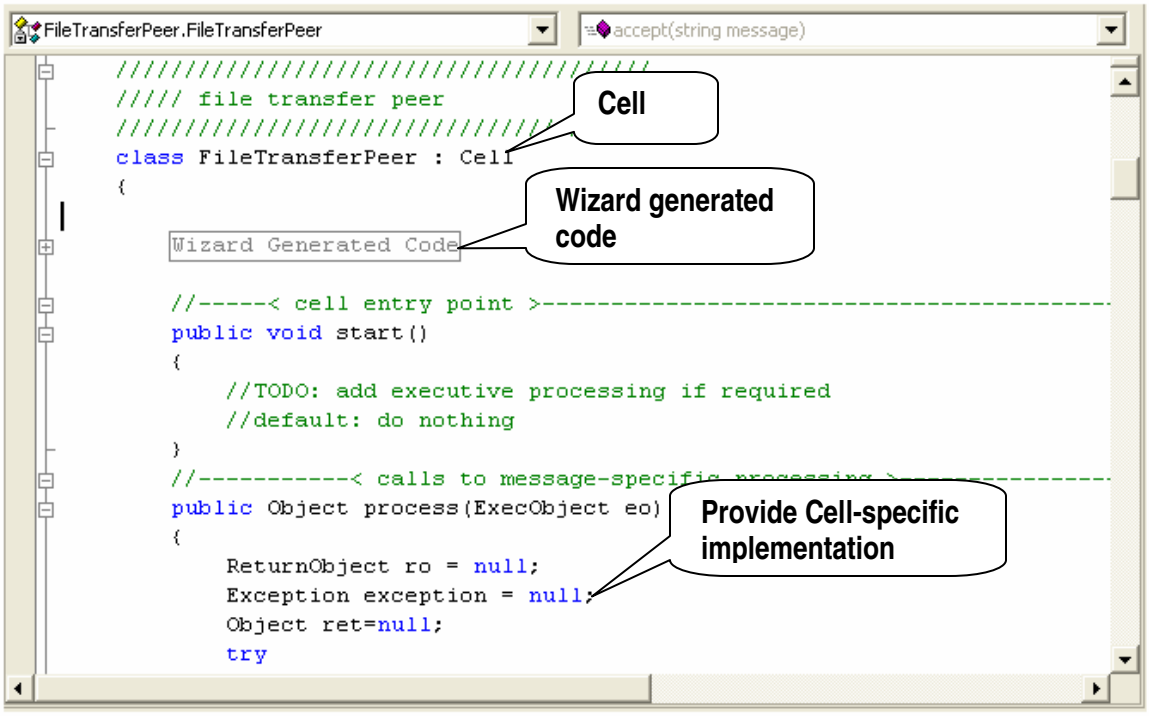

Figure 13: Developing Cells of File Synchronizer

The FileTransferPeer Cell provided file upload and download functionality between different machines on a network. Its capability list included the following message types related to remote connection and file transfer:

SU.FileTransfer.GetLocalDirsFiles

SU.FileTransfer.RunServer

SU.FileTransfer.Connect

SU.FileTransfer.GetRemoteDirsFiles

SU.FileTransfer.UploadLocalFile

SU.FileTransfer.DownloadRemoteFile

The Synchronizer Cell encapsulated the functionality implementing policies used during synchronization. This is what decides what files to copy, overwrite or update while comparing two directories. Its capability list indicated that it could handle the following message type:

SU.Synchronizer.Policy.Compare

The UI(User Interface) Cell was responsible for all of the implementation of the graphical user interface of the application. It also acted as the executive of File Synchronizer application. Program executives, or user interfaces are generally not intended to be reused in other applications. However in the Matrix, to keep the model simple, everything is a Cell, even program executives not meant for reuse. Therefore the

UI Cell was developed just like the above Cells, except that it has an empty capability list, indicating it is not capable of handling requests from other Cells.

The source files of the different Cells were compiled and the resulting binaries on compilation were then dropped into the plug-in directory of the Matrix.

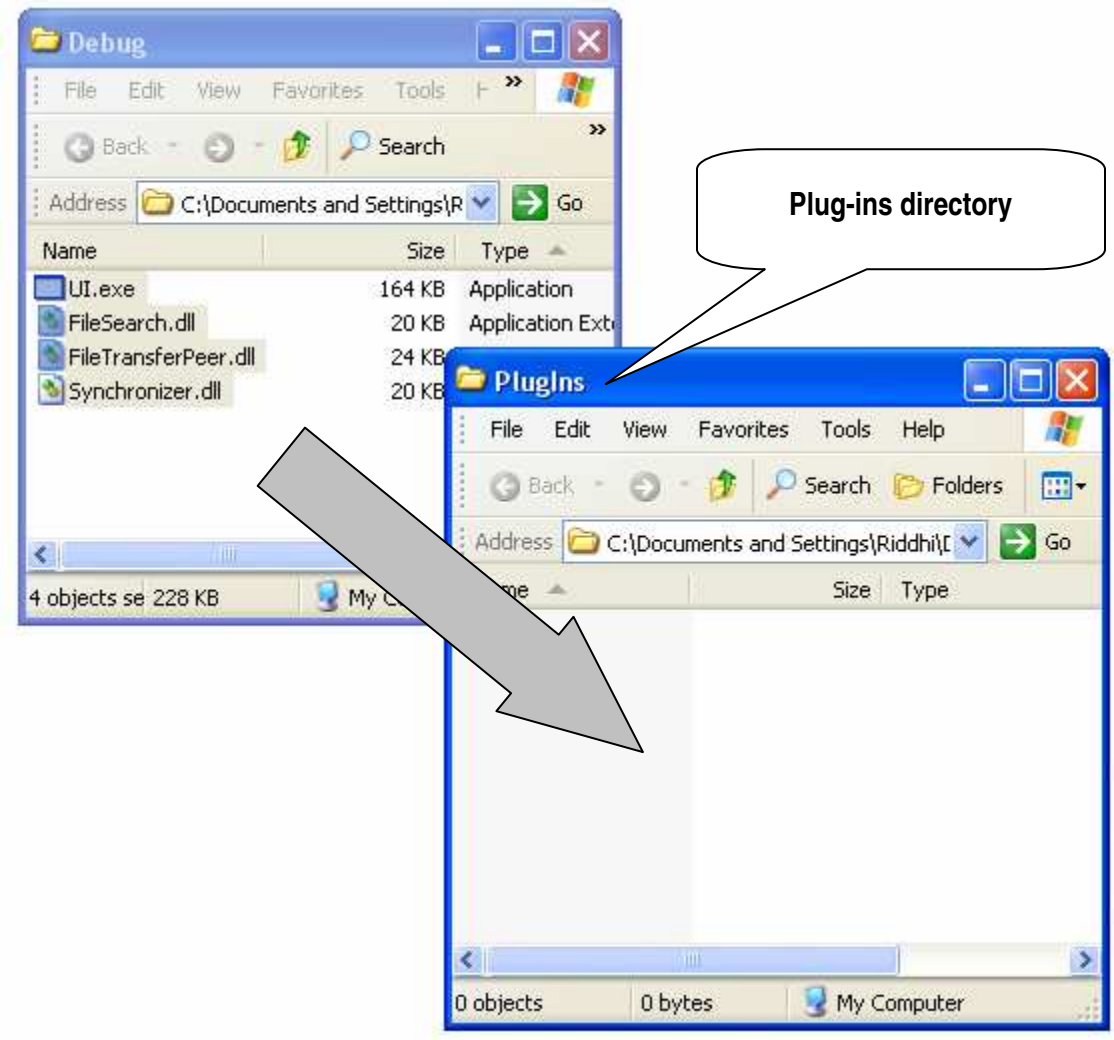

Figure 14: Deploying Cells in the Software Matrix

The Matrix then discovered and loaded up the available Cells, and composed the right pieces together to form the File Synchronizer application.

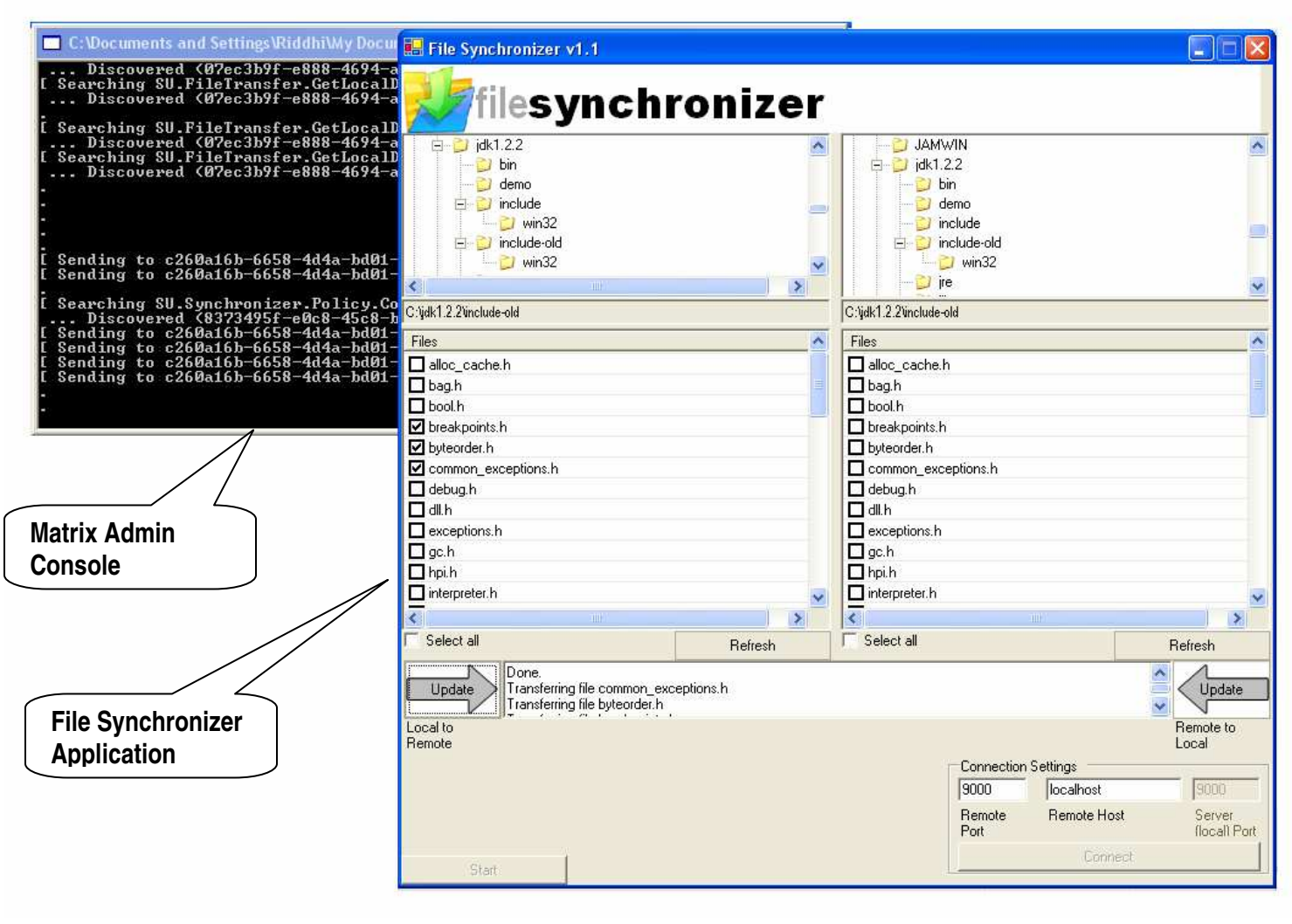

Figure 15: Screenshot of the sample application – File Synchronizer—built using the Software Matrix

4.1.2 EVALUATION

An evaluation of the File synchronizer application, built using the Software Matrix is done in this section.

4.1.2.1 COGNITIVE DISTANCE

Cognitive distance, i.e. the intellectual effort needed to understand and adopt a new methodology or technique is a qualitative metric that has been used before [28] to assess among other things, approaches to reuse.

We feel that the steps taken above to deploy an application on the Matrix via Cells are neither too numerous nor complicated for the average developer with a fair understanding of the platform / language being used. It is appropriate to say that the cognitive distance of the Software Matrix approach is moderate.

4.1.2.2 SOURCE LINES OF CODE

If one compares the File Synchronizer application built in the “traditional” way, with the Matrix version, in terms of number of lines of source code, we notice an increase in latter, as shown in the tables below. However the wizard generated code for Cells, is really infrastructure code——the developer does not write this

code, and this can be hidden from him if required. Therefore by not counting the Cell "boiler-plate" code, we observe that the source code values are comparable, with only a marginal increase in the Software Matrix File Synchronizer (1729 vs. 1526).

Software Matrix File Synchronizer

Source File	Total SLOC	SLOC less wizard code
FileTransferPeer.cs	452	257
FileTransfer.cs	168	168
TreeViewExplorer.cs	79	79
UI.cs	985	790
ChangeConfirmation.cs	234	234
Node.cs	80	80
Synchronizer.cs	316	121
Total	2314	**1729**

Table 1: SLOC for Matrix File Synchronizer Files

Traditional File Synchronizer

Source File	Total SLOC
ChangeConfirmation.cs	**261**
FileTransfer.cs	**368**
TreeViewExplorer.cs	**75**
UI.cs	**822**
Total	**1526**

Table 2: SLOC for Traditional Matrix File Synchronizer Files

4.1.2.3 PERFORMANCE

This section discusses the performance of applications built using the Software Matrix vis-à-vis their “traditional” counterparts.

In employing the Software Matrix supporting infrastructures for system construction, including its enforcement of XML-message based loosely-coupled collaboration, we definitely anticipate an impact on the performance characteristics of applications. Since the File Synchronizer application was built both using traditional means, and then the Matrix, it is possible to compare and determine how the two versions of the same application fare in terms of their performance.

On building and running both versions of the applications, there was no perceptible difference from a user’s perspective. Both versions of the application seemed to be equally responsive and fast. An attempt nevertheless was made to determine the performance from a quantitative perspective.

Commonly performed activities using the File Synchronizer, such as browsing and traversing the file system of a remote machine across the network, and transferring files from one machine to another, were timed in both versions of the application. For instance the table that follows presents the readings of the remote file transfer operation.

(Measurement was done using the high-resolution performance counters provided by the Windows operating system.

The Windows 32 API provides QueryPerformanceCounter() and QueryPerformanceFrequency(), with a timing resolution in the range of a few microseconds [29]. .NET Interop techniques were used in order to access unmanaged/native operating system services from managed code [30]).

The time values of the Matrix approach to File Synchronizer are marginally higher than the traditional version in most cases. The average figure in this case comes to around 20% higher. However as the length of the operation increases, the overhead contribution due to the Matrix becomes more and more negligible.

The performance impact of using the Software Matrix should normally not be prohibitive. The trade-off here is between slightly better performance and increased productivity and ease of salvage. With the ever increasing power of computational devices, adopting the latter is feasible.

File Size	Application	Micro-seconds	Micro-seconds	Micro-seconds	Micro-seconds	Micro-seconds	Average (Milli seconds)
512 B	Traditional File Synchronizer	23749	23186	24089	25085	24511	24
	Software Matrix File Synchronizer	33510	31345	29040	29036	28933	30
1 KB	Traditional File Synchronizer	30546	25439	26900	24625	26008	27
	Software Matrix File Synchronizer	27400	31043	28744	30657	31091	30
2 KB	Traditional File Synchronizer	27383	25942	25083	28363	27874	27
	Software Matrix File Synchronizer	32156	31245	42399	32025	29613	33
4 KB	Traditional File Synchronizer	28971	33569	26922	26085	27051	29
	Software Matrix File Synchronizer	33407	29847	34121	36220	33198	33
8 KB	Traditional File Synchronizer	28050	28037	29482	29644	28997	29
	Software Matrix File Synchronizer	37271	37373	36055	35443	36722	37
16 KB	Traditional File Synchronizer	27364	27792	31073	31185	28414	29
	Software Matrix File Synchronizer	36086	36154	40061	40438	38444	38
32 KB	Traditional File Synchronizer	30263	34870	32966	33374	35854	33
	Software Matrix File Synchronizer	42243	40392	40052	40197	40192	41
64 KB	Traditional File Synchronizer	33401	31481	35427	33863	30715	33
	Software Matrix File Synchronizer	44499	41803	44969	43757	42634	44
128 KB	Traditional File Synchronizer	36313	34334	38538	35821	38726	37
	Software Matrix File Synchronizer	50476	48506	52670	52060	47213	50
256 KB	Traditional File Synchronizer	41855	40845	42391	45852	49748	44
	Software Matrix File Synchronizer	61082	61340	66317	68880	52989	62
512 KB	Traditional File Synchronizer	63793	66424	63992	63604	67101	65
	Software Matrix File Synchronizer	76199	74739	74177	74044	75412	75
1 MB	Traditional File Synchronizer	90762	90185	101821	116057	116019	103
	Software Matrix File Synchronizer	110680	113260	128165	106651	111242	114
2 MB	Traditional File Synchronizer	212977	219673	209435	207322	202039	210
	Software Matrix File Synchronizer	226795	231194	205266	251471	202615	223
4 MB	Traditional File Synchronizer	407272	390905	383886	390538	398694	394
	Software Matrix File Synchronizer	391203	415990	407826	411143	419071	409
8 MB	Traditional File Synchronizer	857204	817551	853566	806387	866713	840
	Software Matrix File Synchronizer	810735	1170234	838999	851975	902344	915

Table 3: Comparing File Synchronizer application performance

4.1.2.4 EASE OF SALVAGE

In order to demonstrate that pieces of the application we've built using the Software Matrix can be salvaged, we "pull-out" approximately one-third of the functionality of this application, and use it through a minimalist interface. The exact functionality of this minimal interface is to perform upload and download file operations between 2 machines, connected to each other over the Internet or a network connection. On specifying the address of the remote machine and appropriate paths, the file transfer operation is carried out.

The steps that were taken to extract the core of the File Synchronizer application and use it are as follows:

- The Matrix infrastructure was copied to a suitable directory location. The file upload and download Cell from the File Synchronizer application was then copied and dropped into the Matrix "plug-in" directory at this location.

- Next, a Cell was developed in order to collaborate with the above Cell. The source code of this Cell is shown in Table 4, and is a combination of the wizard-generated "boiler-plate" code and Cell specific implementation

which in this case, allows it to consume the services of the upload / download Cell.

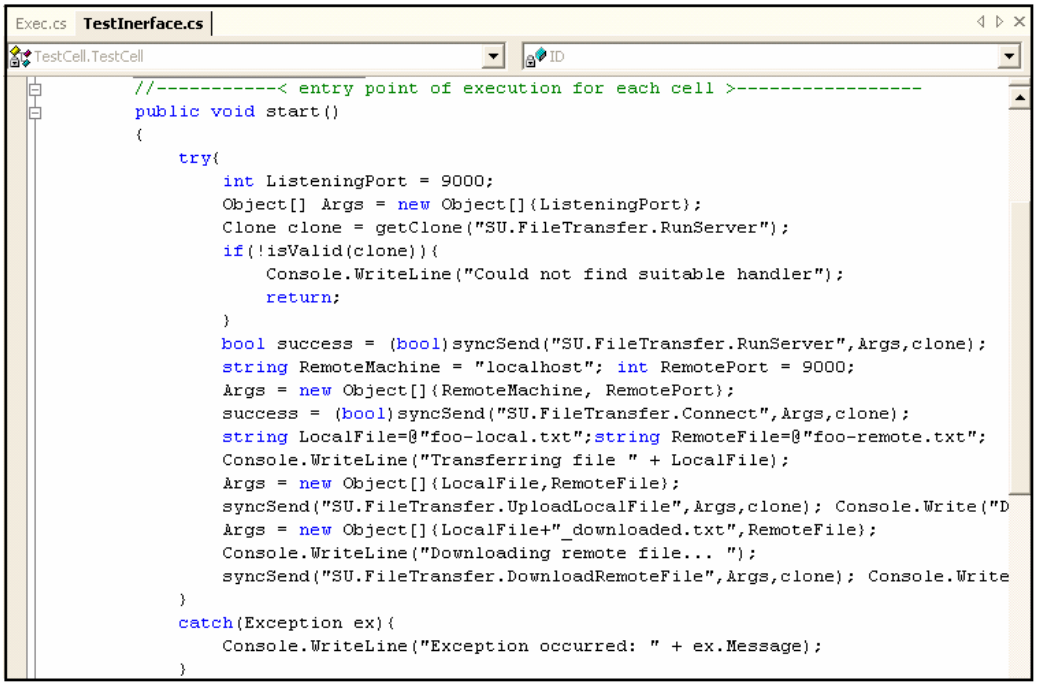

Figure 16: Developing the TestInterface Cell

```
using System;
using Matrix;
namespace TestCell
{
  class TestCell : Cell
  {
    Wizard Generated Code
    //-----------< entry point of execution for each cell >------------
    public void start()
    {
      try{
           int ListeningPort = 9000;
           Object[] Args = new Object[]{ListeningPort};
           Clone clone = getClone("SU.FileTransfer.RunServer");
           if(!isValid(clone)){
               Console.WriteLine("Could not find suitable handler");
               return;
           }
           bool success = (bool)syncSend("SU.FileTransfer.RunServer",Args,clone);
           string RemoteMachine = "localhost"; int RemotePort = 9000;
           Args = new Object[]{RemoteMachine, RemotePort};
```

```
            success = (bool)syncSend("SU.FileTransfer.Connect",Args,clone);
            string LocalFile=@"foo-local.txt";string RemoteFile=@"foo-remote.txt";
            Console.WriteLine("Transferring file " + LocalFile);
            Args = new Object[]{LocalFile,RemoteFile};
            syncSend("SU.FileTransfer.UploadLocalFile",Args,clone); Console.Write("Done.\n");
            Args = new Object[]{LocalFile+"_downloaded.txt",RemoteFile};
            Console.WriteLine("Downloading remote file... ");
            syncSend("SU.FileTransfer.DownloadRemoteFile",Args,clone); Console.Write("Done.\n");
        }
        catch(Exception ex){
            Console.WriteLine("Exception occurred: " + ex.Message);
        }
    }
    //-----------< constructor >----------------------------------------
    public TestCell()
    {
        initializeCell();
    }
  }
}
```

Table 4: Source code of minimal TestInterface Cell

- On compilation, the resulting binary of this Cell was dropped into the Matrix “plug-in” directory, from where it was discovered by the Matrix and then plugged in, and thus the application was composed.

Thus, with a mere 25 lines of developer code, we were able to extract a major chunk of an existing application, in this case approximately 30%, without terminal “arterial bleeding” —— a successful salvage operation.

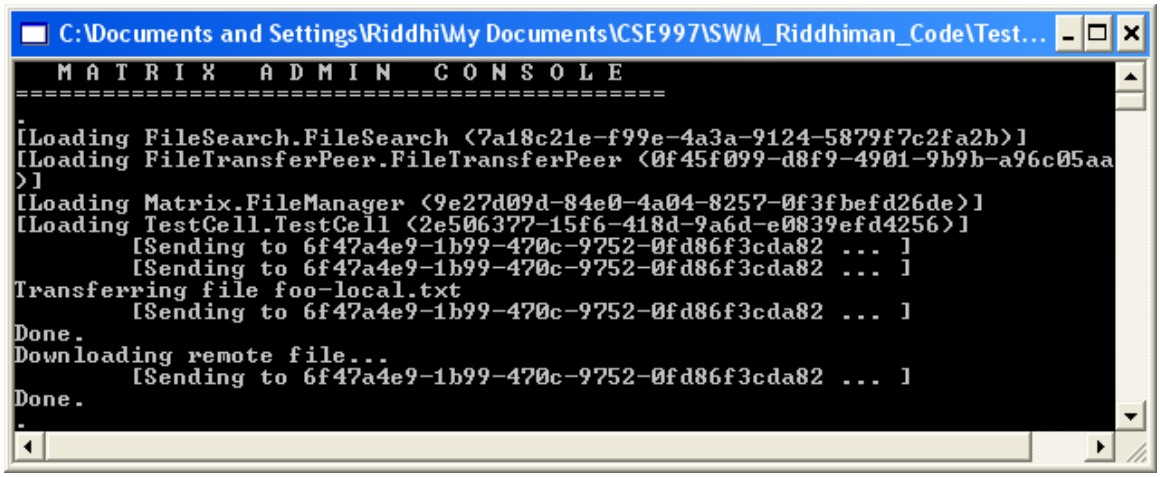

Figure 17: Salvage operation

4.3 NETVIEW

Let us consider another application, built using the Software Matrix. NetView is a simple conferencing tool that allows users drive presentations, slide shows or other similar content from their machines, on other remotely connected computers over the network or the Internet.

Instances of NetView running on machines can either behave in client or server, i.e. master or slave roles. The “master” is able to load slides/images/content from his machine for a slide show. As the master navigates through the presentation, the

content from his machine, in the correct sequence is instantaneously delivered to the remotely connected computers.

A goal during the building of NetView was to be able to salvage a major block of an already existing system. In this case, the file upload and download Cell from the File Synchronizer application is pulled out, and is used in the

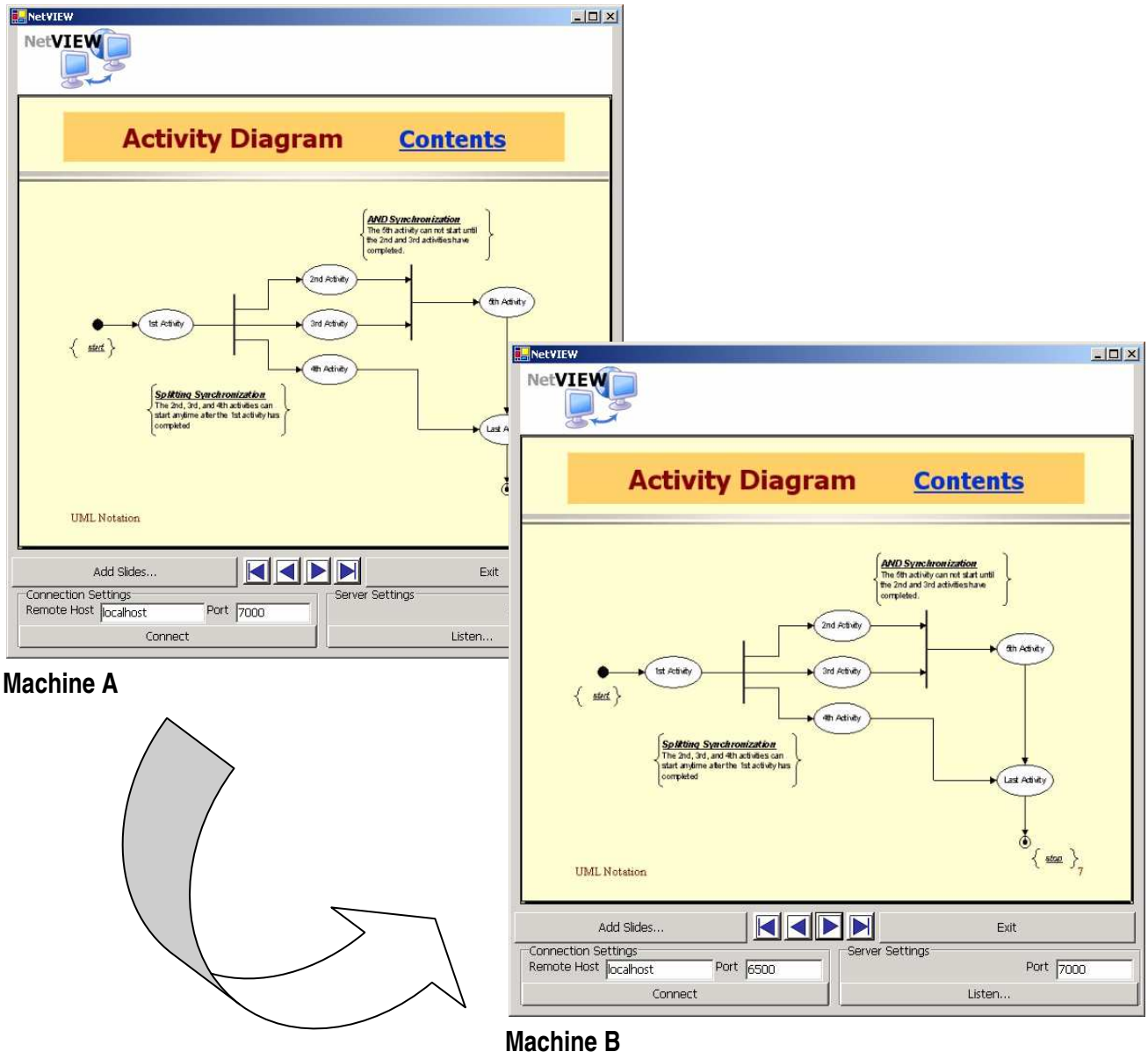

Figure 18: NetVIEW sample application

NetView application. Therefore the Cells that will be the building blocks of this application would be:

- the Cell pulled out of the File Synchronizer application that will allow us to upload and download files between two machines over a network.
- a Cell that handles conversion and serialization of the content being displayed on the master machine so that it can be transmitted to the remote machine.
- the interface responsible for displaying presentation slides/images/other content, and also for loading and navigating through the content.

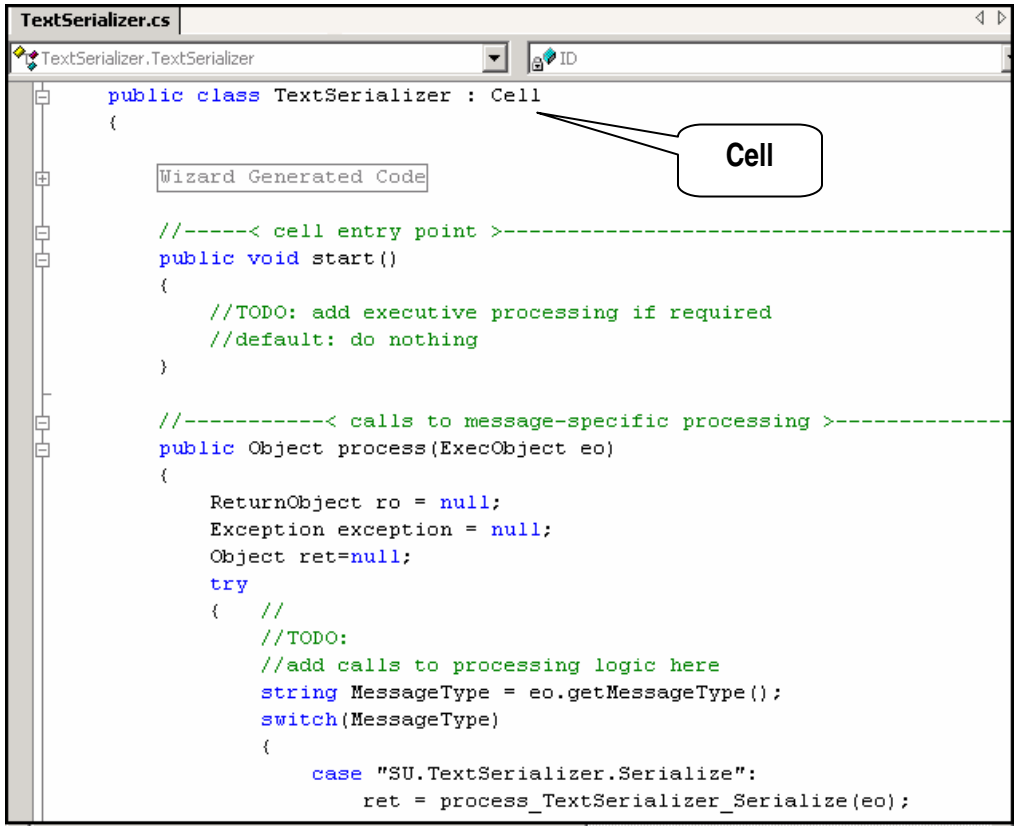

Figure 19: Developing Cells of NetVIEW

The first Cell is simply copied into the Matrix “plug-in” directory. As required by the Matrix, the last two blocks of the NetView application were developed as Cells by following the steps outlined earlier——a combination of the wizard generated Cell “boiler-plate” code and the Cell-specific implementation code. The capability lists of the different Cells, defining what message types different Cells would handle were populated.

In order for the different Cells to collaborate with each other to construct the whole application, it was decided what the capability lists for each Cell would look like, i.e. what message types the different Cells would handle. The Cells were built by incorporating the wizard generated “boiler-plate” code for a generic Cell (in addition to the actual functionality the Cell was responsible for) and certain Cell-specific implementation code. The compiled binaries were then “dropped” into the Matrix “plug-in” directory. These Cells are discovered by the Matrix and then plugged in.

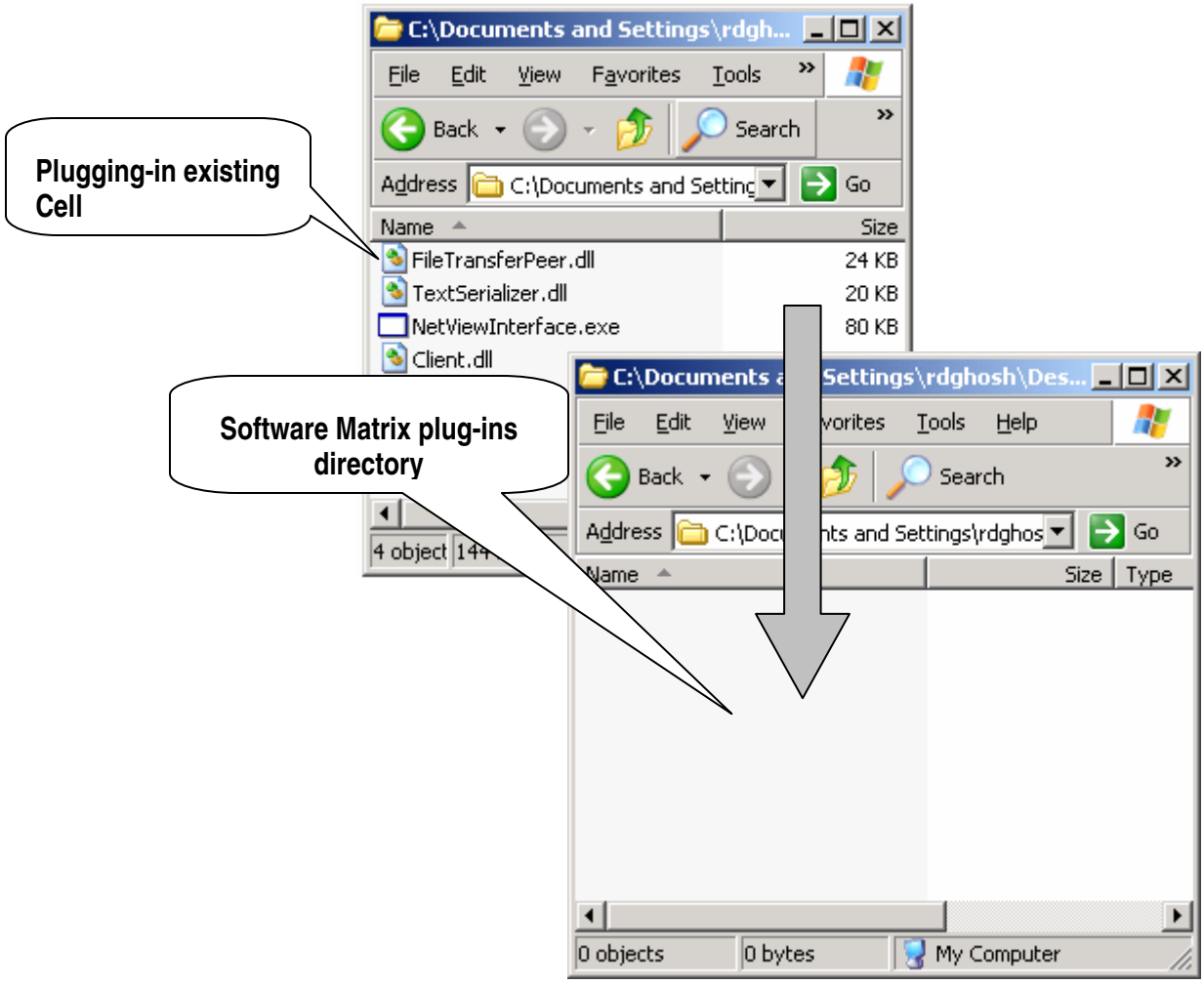

Figure 20: Deploying Cells of NetVIEW; salvaging through existing Cells

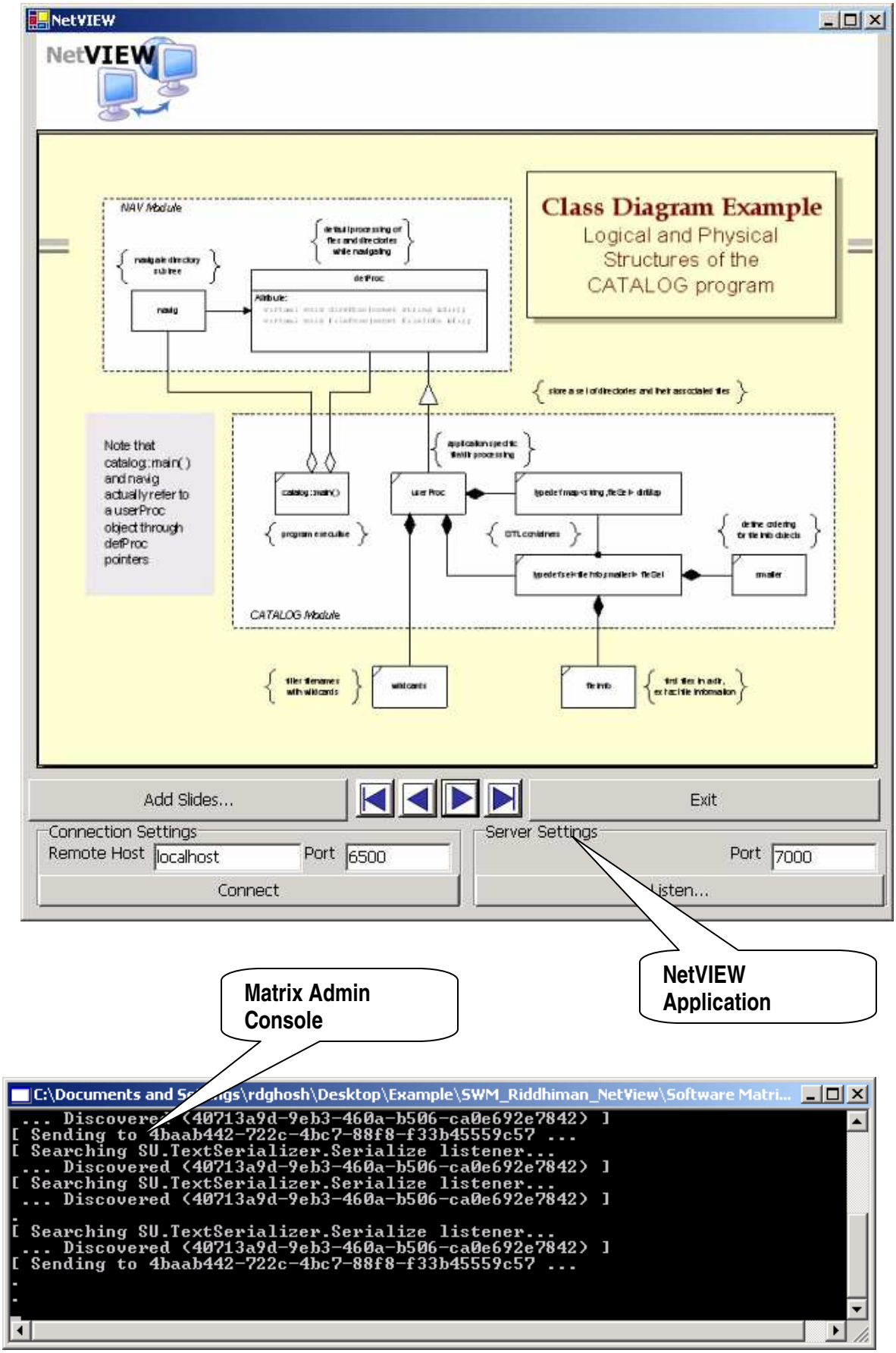

Figure 21: NetVIEW application built using the Software Matrix

The loose-coupling that was enforced by the Software Matrix allowed us to easily pull-out a major chunk of the existing File Synchronizer application in order to build the new NetView application. The messages that flow between the Cells here act like self-describing command and state carriers, and without really wiring together parts at design time the different Cells find each other using the Software Matrix. The effort required in “surgery” in order to extricate pieces from conventional systems that typically have a lot of tight binding between classes, is avoided.

Chapter 5

Conclusion

This chapter summarizes the goals of the Software Matrix, our contributions towards simplifying software salvage, the results of our endeavor and future work.

5.1 ADDRESSING SOFTWARE SALVAGE

Effective recycling of software assets has been an important, yet partially unfulfilled goal of the software engineering discipline; it holds out the promises of reduced development and maintenance costs, gains in development schedule and quicker time-to-market, and robustness and quality over software developed without reuse. While these benefits are widely recognized, the practice of systematic reuse is certainly

not widespread in the software industry. As discussed earlier, there has also been a reluctance to adopt the “high-ceremony” solutions put forth by the reuse research community, especially among the small and medium sized software shops.

In this thesis we have been concerned with the more pragmatic practice of software salvage——of lifting significant blocks of existing systems and inserting them into a newly developed system. While it may seem easy to lift major blocks of code from one system for use within another, our experience (Fawcett) with organizations that routinely practice software salvage, such as the Radar Systems Department, General Electric Company, tells us that the truth is quite different. The large pieces we want to recycle often have many dependencies on parts we don’t want, which may result in expensive modifications to the part being salvaged.

5.2 CONTRIBUTIONS OF THIS THESIS

Our contribution in this thesis centered on significantly managing the problem of salvage.

- We proposed an architecture for system construction, the Software Matrix, with a view to simplify the paradigm of software salvage. Rather than concentrating only on programming languages as in object-oriented theory, or only on packaging techniques as in component-oriented technologies (both to a certain extent meant to address reuse), our approach went a

step further by viewing applications solely as compositions of different pieces——of Cells in a Matrix——and having a framework that actively supports the dynamic composition of applications from the collaboration of these different pieces. In particular the Software Matrix is a runtime infrastructure into which individual pieces of an application can plug. Applications are then composed from these different pieces by having them collaborate by passing messages to each other.

- By enforcing a separation of concern between pieces of an application by inserting a message bus between them we are insisting on loose-coupling, which makes salvaging these pieces for use in future applications much easier. This addresses one of the main problems of salvaging which has been the problem of extracting parts of monolithic (or very tightly-coupled) applications.

- The mediator is one of the key strategies that makes the Software Matrix work. It promotes loose coupling by keeping system entities from referring to each other explicitly. Due to the introduction of a mediator the different entities need only to know the mediator and the minimalist message-passing interface needed to

communicate with it, thereby reducing the number of interconnections. Salvage is much easier because Cells can find each other through the Matrix, and a designer does not have to wire together the parts.

- The Matrix infrastructure also facilitates the discovery of the right pieces needed for system construction and dynamically connects those pieces and composes them in order to build an application. Therefore, one part of an application does not bind to another at compile time, nor does it bind to a particular interface signature. All it does is specify the types of messages they need handled, and correct application component is located at runtime. In this sense, the message-passing interface is a universal interface.

- The ideas presented here were implemented and the Matrix runtime infrastructure was built using Microsoft’s .NET framework and the C# programming language. This infrastructure was used to construct two real-world tools in order to understand the issues involved. Our techniques were evaluated on the basis of cognitive distance, performance, code size and ease of salvage, and the results obtained were encouraging. As an experiment these tools were developed both using

traditional development, and using the Software Matrix method of system construction. For the examples we examined the penalties in terms of performance or code size were not serious with the advantage obtained being systems that were amenable to salvage operations.

5.3 FUTURE WORK

During the process of working on the Software Matrix directions for further research and development were identified. Pursuing these areas of improvement would help in the adoption of the Software Matrix paradigm of system construction, in developing systems much larger than we have considered in our experiments.

5.3.1 MESSAGE CATALOGS

Since the capabilities (and hence the potential for salvage) of application building blocks (Cells), are identified by message types, it would be useful if there existed catalogs that listed message types available and being used in a particular domain, vertical or organization. This catalog would then aid developers in salvaging existing pieces of applications from their current software assets, and could be used in conjunction with the Software Matrix.

5.3.2 VERSIONING SUPPORT SCHEMES

In the Software Matrix message types are significant. Cells handling exactly the same message types, if such Cells exist, can be thought to be providing equivalent message handling functionality, which may be true within an organization's or department's code base. However associating versioning information with message types could add more flexibility for graceful evolution of systems. The current implementation of the Software Matrix lacks this support. It would be interesting to add this feature to the Matrix, leveraging inherent .NET assembly versioning features. Cells could then specify versions if they so wish in addition to the message types they need handled.

5.3.3 REMOTE CELL COLLABORATION

Our mental model of the Software Matrix has been for it to aid in the composition of systems from available Cells, and in most cases it makes sense to have Cells available locally (note however that Cells may themselves access remote functionality if they so wish). However it could be envisaged to have system composition include Cells on a remote machine in addition to locally available Cells. It would be interesting to investigate further the new use cases that can be supported due to this improvement, as

also the issues that this gives rise to (such as performance).

5.3.4 APPLICATIONS OF THE SOFTWARE MATRIX

The Matrix infrastructure, apart from supporting salvage operations could also be used in other scenarios. For instance, the fact that the Matrix can act as an interceptor can be exploited for the automated testing (regression testing) of systems. When a modified cell is added to the system, the newly generated pattern and nature of message exchanges can be compared with a known pattern from a previous log, in order to determine if the most recent changes “broke” the system.

5.3.5 USABILITY STUDIES

As with any new methodology, unfamiliarity with the new techniques is a stumbling block in the way of its adoption. While the Software Matrix aims to be simple, it would be very instructive to conduct even a straightforward usability study amongst developers to obtain feedback on how the Matrix infrastructure and the services offered by it could be exposed in a more convenient way. This would go a long way in ensuring that the use of the Matrix is intuitive and simple.

References

[1] Jacobson, I., Griss M. and Jonsson P., *Software Reuse: Architecture Process and Organization for Business Success*, Addison-Wesley, 1997.

[2] Frank W. L., What Limits to Software Gains, *ComputerWorld,* pp. 65-70, May 1981.

[3] Jones T. C., Reusability in Programming: A survey of the state of the art, *IEEE Transactions on Software Engineering,* pp. 488-494, September 1984.

[4] Brooks, F.P., *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition,* Addison-Wesley, Reading, MA, 1998.

[5] McIlroy, D., Mass-produced software components, in Buxton, J.M., Naur, P., and Randell, B., editors, *Software Engineering Concepts and Techniques, 1968 Nato Conference on Software Engineering,* pp. 138-155, January 1969.

[6] Cox, B.J., *Object-Oriented Programming: An Evolutionary Approach,* Addison-Wesley, Reading, MA, 1986.

[7] Gaffney, J. and Cruickschank, R., A General Economics Model of Software Reuse, in *Proceedings, International Conference on Software Engineering,* pp. 327-332, May 1992.

[8] Griss, M., Software Reuse: Objects and Frameworks are not Enough, *Object Magazine,* February 1995.

[9] Frakes, W., and Terry, C., Software Reuse: Metrics and Models, *ACM Computing Surveys,* Vol. 28, No. 2, pp. 415-435, June 1996.

[10] Zand, M., Basili, V., Baxter, I., Griss, M., Karlsson, E., Perry, D., Reuse R&D: Gap Between Theory and Practice, *Proceedings of the Symposium on Software Reusability,* pp. 172-177, 1999.

[11] Ibid.

[12] Booch, G., and Vilot, M., The Design of the C++ Booch Components, *Proceedings of the ECOOP on OOPSLA,* pp. 1-11, 1990.

[13] Microsoft Corporation, *The Microsoft Foundation Class Library.* (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vcmfc98/html/mfchm.asp)

[14] Szyperski, C., *Component Software: Beyond Object-Oriented Programming,* ACM Press, New York, 1997.

[15] Ibid.

[16] D'Souza, D.F., and Will, A.C., Objects, Components and Frameworks with UML: The Catalysis Approach, Addison-Wesley, Reading, MA, 1998.

[17] Mili H., Mili A., Yacoub S., Addy E., *Reuse Based Software Engineering: Techniques, Organization, and Controls*, John Wiley & Sons, New York, 2002.

[18] Microsoft Corporation, *The Component Object Model Specification,* Version 0.9, October 1995. (http://www.microsoft.com/com/resources/comdocs.asp)

[19] Vinoski, S., Introduction to CORBA, *Proceedings of the 22nd International Conference on Software Engineering,* pp. 822, 2000.

[20] Sun Microsystems, Inc., *Enterprise JavaBeans Specification*, Version 2.1, November 2003. (http://java.sun.com/products/ejb/docs.html)

[21] Cox, B.J., *Object-Oriented Programming: An Evolutionary Approach,* Addison-Wesley, Reading, MA, 1986.

[22] *The Matrix,* Directors: Wachowski, A. and Wachowski, L., Warner Bros., 1999.

[23] Boost.org, The Boost C++ Library, 2004 (http://www.boost.org)

[24] Gamma, E., Helm, R., Johnson, R. and Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley, Boston, MA, 1995.

[25] Ibid.

[26] Box, D. and Sells, C., *Essential .NET, Volume I: The Common Language Runtime,* Addison-Wesley, 2002.

[27] Löwy, J., *Programming .NET Components,* O'Reilly & Associates, Sebastopol, CA, 2003.

[28] Krueger, C.W., Software Reuse, *ACM Computing Surveys,* Vol. 24, No. 2, June 1992.

[29] Microsoft Corporation, How To Use QueryPerformanceCounter to Time Code, *Microsoft Developer Network Article 172338,* Revision 1.0, July, 2004. (http://support.microsoft.com/kb/q172338/)

[30] Microsoft Corporation, Interoperating with Unmanaged Code, *.NET Framework Developer's Guide*, 2004. (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconInteroperatingWithUnmanagedCode.asp)

Appendix

Source Code Listings

Software Matrix Source Code

```
/////////////////////////////////////////////////////////////////////
//  Loader.cs      The loader continously monitors a specifiec     //
//                 "PlugIns" directory for any new "Cells",        //
//                 and if appropriate loads them into the          //
//                 Software Matrix.                                //
//  ver 1.0                                                        //
//                                                                 //
//  Language:      C#                                              //
//  Platform:      Dell Dimension Pentium 4, Windows 2000          //
//  Course:        CSE997 Master's Thesis                          //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>               //
//                                                                 //
/////////////////////////////////////////////////////////////////////
using System;
using System.IO;
using System.Threading;


using System.Collections;
using System.Reflection;

namespace MatrixManager
{
    //monitor plug-in location and load valid cells
    public class Loader
    {
        string SearchPath;
        Hashtable PlugInNames;

        //-----------< Loader constructor >-----------------------------------
        public Loader(string _SearchPath)
        {
            //_SearchPath is the relative path from the Matrix executable
            PlugInNames = new Hashtable();
            SearchPath = _SearchPath;
            Thread MonitorThread = new Thread(new ThreadStart(monitor));
            MonitorThread.Start();
        }

        //-----------< function to monitor plug-in directory >------------------
        public void monitor()
        {
            while(true)
            {
                //use a "directory watcher" here instead of sleep/loop
                Console.WriteLine(".");
                string d = Directory.GetCurrentDirectory();
                String[] Files1 = Directory.GetFiles(SearchPath,"*.dll");
                String[] Files2 = Directory.GetFiles(SearchPath,"*.exe");
                ArrayList merge = new ArrayList();
                foreach(string f in Files1)
                    merge.Add(f);
                foreach(string f in Files2)
                    merge.Add(f);
                String[] Files = (String[])merge.ToArray(typeof(String));
                foreach(string f in Files)
                {
                    if(!PlugInNames.ContainsKey(f))
                    {
                        PlugInNames.Add(f,f);
                        FileInfo fi = new FileInfo(f);
                        AppDomain.CurrentDomain.AppendPrivatePath(SearchPath);
                        Assembly a = Assembly.Load(
                            Path.GetFileNameWithoutExtension(fi.FullName));
                        Type[] types = a.GetTypes();
                        Type compare = typeof(Matrix.Cell);
```

```
                        foreach(Type t in types)
                        {
                            if(t.Equals(compare))
                                continue;
                            if(compare.IsAssignableFrom(t))
                            {
                                Type[] NoParams = new Type[0];
                                Object[] Args = new Object[0];
                                Object o = Activator.CreateInstance(t,Args);
                                ((Matrix.Cell)o).register();
                                Console.WriteLine("[Loading " + t.ToString()
                                    + " (" + ((Matrix.Cell)o).getID()+ ")" +  "]");
                                break;
                            }
                        }
                    }
                }
                Thread.Sleep(5000);
            }
        }


    }
}
```

```
////////////////////////////////////////////////////////////////
//  Exec.cs         The program executive.                    //
//  ver 1.0                                                   //
//                                                            //
//  Language:      C#                                         //
//  Platform:      Dell Dimension Pentium 4, Windows 2000     //
//  Course:        CSE997 Master's Thesis                     //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>          //
//                                                            //
////////////////////////////////////////////////////////////////
using System;
using System.IO;

namespace MatrixManager
{
    public class Exec
    {
        [STAThread]
        static void Main(string[] args)
        {
            Console.WriteLine("   M A T R I X   A D M I N   C O N S O L E   ");
            Console.WriteLine("=============================================");
        #if RUN
...
        #else
            Loader l = new Loader("PlugIns");
        #endif

        }
        public Exec()
        {

        }
    }
}
```

```
/////////////////////////////////////////////////////////////////
//  SyncWait.cs      Class that provides services to enable    //
//                   synchronous request/receive communication //
//                   between cooperating Cells. Implements     //
//                   efficient waiting through events.         //
//  ver 1.0                                                    //
//                                                             //
//  Language:      C#                                          //
//  Platform:      Dell Dimension Pentium 4, Windows 2000      //
//  Course:        CSE997 Master's Thesis                      //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>           //
//                                                             //
/////////////////////////////////////////////////////////////////

using System;
using System.Threading;


namespace Matrix
{
    //class to enable synchronous collaboration
    public class SyncWait
    {
        Object result;
        AutoResetEvent Received;
        Cell cell;
        Guid RequestID;
        ReceiveDelegate rdel;

        //-----< contructor >-------------------------------------------------
        public SyncWait(Cell c, Guid _RequestID)
        {
            cell = c;
            RequestID = _RequestID;
            subscribe();
            Received = new AutoResetEvent(false);
        }

        //-----< subscribe to response receive notification >-----------------
        private void subscribe()
        {
            rdel = new ReceiveDelegate(Check);
            cell.OnReceive += rdel;
        }

        //-----< unsubscribe from response receive notification >-------------
        private void unsubscribe()
        {
            cell.OnReceive -= rdel;
        }

        //-----< check response >---------------------------------------------
        private void Check(Object queue, ReceiveInfoEventArgs rargs)
        {
            result = rargs.getResult();
            cell.addResponse(rargs.getMessageID());
            Received.Set();
        }

        //-----< wait for response >------------------------------------------
        public Object wait()
        {
            if(cell.checkAndRetrieveResponse(RequestID))
            {
                cell.removeResponse(RequestID);
                if(((ReturnObject)result).hasException())
```

```
                    throw ((ReturnObject)result).getException();
                else
                    return result;
            }
            Received.WaitOne();
            Received.Reset();
            unsubscribe();
            if(((ReturnObject)result).hasException())
                throw ((ReturnObject)result).getException();
            else
                return ((ReturnObject)result).getReturnValue();
        }
    }

    ////////////////////////////////////////////////////////////////////////
    //           A class derived from EventArgs to hold information       //
    //           related to a OnReceive event                             //
    ////////////////////////////////////////////////////////////////////////
    public class ReceiveInfoEventArgs : EventArgs
    {
        private Guid MessageID;
        private Object Result;

        //-----< cosntructor >-------------------------------------------------
        public ReceiveInfoEventArgs(Guid _MessageID, Object _Result)
        {
            MessageID = _MessageID;
            Result = _Result;
        }

        //-----< get message id >----------------------------------------------
        public Guid getMessageID()
        {
            return MessageID;
        }

        //-----< get result >--------------------------------------------------
        public Object getResult()
        {
            return Result;
        }
    }
}
```

```
/////////////////////////////////////////////////////////////////
//  ReturnObject.cs  Implements the ReturnObject class that    //
//                   assists in packaging of return values     //
//                   along with any thrown exceptions, used    //
//                   during the collaboratio of different      //
//                   Cells in the Matrix.                      //
//  ver 1.0                                                    //
//                                                             //
//  Language:      C#                                          //
//  Platform:      Dell Dimension Pentium 4, Windows 2000      //
//  Course:        CSE997 Master's Thesis                      //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>           //
//                                                             //
/////////////////////////////////////////////////////////////////
using System;

namespace Matrix
{
    [Serializable]
    public class ReturnObject
    {
        Object ReturnValue;
        Exception exception;

        //-----< constructor >-------------------------------------------------
        public ReturnObject(Object _ReturnValue, Exception _exception)
        {
            ReturnValue = _ReturnValue;
            exception = _exception;
        }

        //-----< get return value of response >--------------------------------
        public Object getReturnValue()
        {
            return ReturnValue;
        }

        //-----< get the encapsulated exception object >-----------------------
        public Exception getException()
        {
            return exception;
        }

        //-----< indicates the presence of an exception >----------------------
        public bool hasException()
        {
            if(exception == null)
                return false;
            else
                return true;
        }
    }
}
```

```
/////////////////////////////////////////////////////////////////
//  Messaging.cs     Utility class that helps in the           //
//                   composition and decomposition of XML      //
//                   messages used in the Matrix.              //
//  ver 1.0                                                    //
//                                                             //
//  Language:      C#                                         //
//  Platform:      Dell Dimension Pentium 4, Windows 2000     //
//  Course:        CSE997 Master's Thesis                     //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>          //
//                                                             //
/////////////////////////////////////////////////////////////////

using System;
using System.Xml;
using System.IO;
using System.Text;
using System.Collections;
using System.Reflection;
using System.Threading;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using System.Runtime.Serialization.Formatters.Soap;

namespace Matrix
{

    //class to support XML message composition and decomposition
    public class Messaging
    {
        XmlTextWriter wr;
        MemoryStream stream;

        //-----------< default constructor >-----------------------------------
        public Messaging()
        {
        }

        //-----------< method to build request messages >----------------------
        public string buildRequest(string RequestCellGuid, string MessageName,
            Object ReturnVal, Object[] Params, out Guid RequestID, Guid ResponseGuid/*
    clone*/)
        {
            RequestID = Guid.Empty;
            string message=null;
            try
            {
                stream = new MemoryStream();
                wr = new XmlTextWriter(stream, Encoding.UTF8);
                wr.Formatting = Formatting.Indented;
                RequestID = Guid.NewGuid();
                wr.WriteStartDocument();
                wr.WriteStartElement("Message");
                wr.WriteElementString("Type", "Request");
                wr.WriteElementString("RequestID",RequestID.ToString());
                wr.WriteElementString("RequestCellGUID",RequestCellGuid);
                if(ResponseGuid.CompareTo(Guid.Empty)==0)
                    wr.WriteElementString("ResponseCellGUID","NOT_SET");
                else
                    wr.WriteElementString("ResponseCellGUID",ResponseGuid.ToString());
                wr.WriteStartElement("Method");
                wr.WriteElementString("Name",MessageName);
                wr.WriteStartElement("ReturnVal");
                wr.WriteElementString("ReturnType",
                    "Not_Needed_3_21_04"/*ReturnVal.GetType().ToString()*/);
                wr.WriteElementString("ReturnValue","NOT_SET");
```

```
            wr.WriteEndElement();
            if(Params==null){
                wr.WriteElementString("NoOfParams","0");
            }
            else{
                wr.WriteElementString("NoOfParams",Params.Length.ToString());

                foreach(Object o in Params){
                    wr.WriteStartElement("Param");
                    wr.WriteElementString("ParamType", o.GetType().ToString());
                    wr.WriteElementString("ParamValue", textSerialize(o));
                    wr.WriteEndElement();
                }
            }
            wr.WriteEndElement();
            wr.WriteEndElement();
            wr.Flush();
            message = toString(stream);
        }
        catch(Exception e){
            Console.WriteLine(e.Message);
        }
        finally{
            wr.Close();
        }
        return message;
    }


    //-----< return a string representation of message >--------------------
    public  string toString(MemoryStream _stream){

        if(_stream.CanRead == true)
        {
            _stream.Position=0;
            StreamReader sr = new StreamReader(_stream);
            return sr.ReadToEnd();
        }
        else
            return "";
    }

    //-----------< method to extract request or response messages >---------
    public ExecObject extract(object msg)
    {
        Exception e = new Exception("XML message is not in expected format");
        string message = (string)msg;
        try
        {
            XmlDocument doc = new XmlDocument();
            doc.LoadXml(message);
            XmlNode root = doc.DocumentElement;
            if(root.Name.Equals("Message")== false)
                throw e ;
            XmlNode Type = root.FirstChild;
            switch(Type.InnerText)
            {
                case "Request":
                    return extractRequest(doc);

                case "Response":
                    return extractResponse(doc);
            }
        }
        catch(Exception ex)
        {
```

```
                Console.WriteLine(ex.Message);
            }
            return null; //should not be reached
        }


        //-----------< method to build CellToCell response messages >----------------
        public string buildResponse(Object msg, Object Ret, out Guid RequestCellGUID)
        {
            Exception e = new Exception("XML message is not in expected format");

            string message = (string)msg;
            XmlDocument doc = new XmlDocument();
            doc.LoadXml(message);
            XmlNode root = doc.DocumentElement;
            if(root.Name.Equals("Message")== false)
                throw e ;
            XmlNode Type = root.FirstChild;
            if(Type.InnerText == "Request")
                Type.InnerText = "Response";
            else
                throw e;

            XmlNodeList RequestCellGUIDList = doc.GetElementsByTagName("RequestCellGUID")↙
    ;
            RequestCellGUID = new Guid(RequestCellGUIDList[0].InnerText);

            XmlNodeList ReturnValue = doc.GetElementsByTagName("ReturnValue");
            ReturnValue[0].InnerText = textSerialize(Ret);

            MemoryStream mstr = new MemoryStream();
            XmlTextWriter xtw = new XmlTextWriter(mstr, Encoding.UTF8);
            xtw.Formatting = Formatting.Indented;
            xtw.Flush();
            doc.Save(mstr);
            message = toString(mstr);
            return message;
        }

        //-----------< method to extract Request messages >----------------
        public ExecObject extractRequest(XmlDocument doc)
        {
            XmlNode root = doc.DocumentElement;
            XmlNodeList Children = root.ChildNodes;
            XmlNodeList RequestID = doc.GetElementsByTagName("RequestID");
            XmlNodeList RequestCellGUID =
                doc.GetElementsByTagName("RequestCellGUID");
            XmlNodeList ResponseCellGUID =
                doc.GetElementsByTagName("ResponseCellGUID");
            Guid Resp = new Guid(ResponseCellGUID[0].InnerText);
            XmlNodeList MessageName = doc.GetElementsByTagName("Name");
            XmlNodeList Param = doc.GetElementsByTagName("Param");
            Object Ret = null;
            Object[] Args = new Object[Param.Count];
            for(int i=0; i<Param.Count; i++)
            {
                string val = (Param[i]).ChildNodes[1].InnerText;
                Object o = textDeserialize(val);
                Args[i]=o;
            }

            ExecObject eo = new ExecObject(true/*Request*/,RequestCellGUID[0].InnerText,
                                           ResponseCellGUID[0].InnerText,
                                           RequestID[0].InnerText,
                                           MessageName[0].InnerText,
                                           Ret, Args);
```

```
        return eo;
    }

    //-----------< method to extract Response messages >----------------
    public ExecObject extractResponse(XmlDocument doc)
    {
        XmlNode root = doc.DocumentElement;
        XmlNodeList Children = root.ChildNodes;
        XmlNodeList RequestIDList = doc.GetElementsByTagName("RequestID");
        XmlNodeList ReturnValueList = doc.GetElementsByTagName("ReturnValue");
        string RequestID = RequestIDList[0].InnerText;
        string val = ReturnValueList[0].InnerText;
        ExecObject eo = new ExecObject(false, RequestID, textDeserialize(val));
        return eo;
    }

    //----------< method to forward messages from one cell to another >-----
    public void forwardRequest(object msg)
    {
        string message = (string)msg;
        Exception e =
            new Exception("XML message is not in expected format");
        XmlDocument doc = new XmlDocument();
        doc.LoadXml(message);
        XmlNode root = doc.DocumentElement;
        if(root.Name.Equals("Message")== false)
            throw e ;
        XmlNode Type = root.FirstChild;
        if((Type.InnerText.Equals("Request"))== false)
            return;
        else
        {
            XmlNodeList ResponseCellGUID =
                doc.GetElementsByTagName("ResponseCellGUID");
            string CloneGuid = ResponseCellGUID[0].InnerText;
            Guid Resp;
            if(CloneGuid == "NOT_SET")
            {
                XmlNodeList MessageTypeName = doc.GetElementsByTagName("Name");
                string MessageType = MessageTypeName[0].InnerText;
                string guid = null;
                Console.WriteLine("[ Searching " + MessageType + " listener... ");
                while(guid==null)
                {
                    guid=Matrix.discover(MessageType);
                }
                Console.WriteLine(" ... Discovered " + "("+guid+") ]");
                Resp = new Guid(guid);
                ResponseCellGUID[0].InnerText = Resp.ToString();
                MemoryStream mstream = new MemoryStream();
                XmlTextWriter x = new XmlTextWriter(mstream, Encoding.UTF8);
                doc.Save(x);
                x.Flush(); mstream.Position=0;
                StreamReader sr = new StreamReader(mstream);
                string mesg = sr.ReadToEnd();
                //sends message to a specific cell
                Matrix.send(mesg, Resp);
            }
            else{
                Resp = new Guid(CloneGuid);
                Matrix.sendClone(message,Resp);
            }
        }
    }
```

```
        //-----< method to serialize an object in text (base64) format >--------

        public string textSerialize(Object obj)
        {
            if(obj == null)
                return "";
            MemoryStream ms = new MemoryStream();
            BinaryFormatter b = new BinaryFormatter();
            b.Serialize(ms,obj);
            byte[] buffer = ms.ToArray();
            return Convert.ToBase64String(buffer);
        }


        //----------< method to deserialize a base64 encoded text string >------
        public Object textDeserialize(string ser)
        {
            if(ser.Equals(""))
                return null;
            byte[] buffer = Convert.FromBase64String(ser);
            MemoryStream ms = new MemoryStream(buffer);
            BinaryFormatter b = new BinaryFormatter();
            return (b.Deserialize(ms));
        }
    }


    //----------< TEST STUB >-------------------------------------------------
    #if(TEST_MESSAGING)
...
    #endif
}
```

```
/////////////////////////////////////////////////////////////////
//  Matrix.cs        Implements the Matrix class. Provides     //
//                   implementations for operations commonly   //
//                   performed in the Matrix via static        //
//                   functions. Also maintains a collection of //
//                   all active cells in the Matrix and        //
//                   facilitates their discovery and communic- //
//                   ation.
//  ver 1.0                                                    //
//                                                             //
//  Language:      C#                                         //
//  Platform:      Dell Dimension Pentium 4, Windows 2000     //
//  Course:        CSE997 Master's Thesis                     //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>          //
//                                                             //
/////////////////////////////////////////////////////////////////

using System;
using System.Collections;

namespace Matrix
{
    public class Matrix
    {
        internal static Hashtable Cells=new Hashtable(20);
        internal static Hashtable Clones=new Hashtable(20);
        static BlockingQueue MatrixQueue = new BlockingQueue(
            new BlockingQueue.DequeueDelegate(new Messaging().forwardRequest));

        //-----------< cell registration >----------------
        public static bool register(Cell c)
        {
            lock(Cells)
            {
                Cells.Add(c.getID(),c);
            }
            return true;
        }

        //-----------< cell deregistration >----------------
        public static bool unregister(Cell c)
        {
            lock(Cells)
            {
                Cells.Remove(c.getID());
            }
            return true;
        }

        //-----------< clone registration >----------------
        public static bool registerClone(Cell c)
        {
            lock(Clones)
            {
                Clones.Add(c.getID(),c);
            }
            return true;
        }

        //-----------< clone deregistration >----------------
        public static bool unregisterClone(Cell c)
        {
            lock(Clones)
            {
                Clones.Remove(c.getID());
            }
```

```
            return true;
        }


        //-----< create clone of a Cell >--------------------------------------
        public static Guid createClone(string MessageName)
        {
            string guid = null;
            System.Threading.Thread.Sleep(500);
            guid=Matrix.discover(MessageName);
            if(guid==null)
                return Guid.Empty;
            //creating clone of existing type
            Cell c = (Cell)Activator.CreateInstance((Cells[new Guid(guid)]).GetType());
            registerClone(c);
            return c.getID();
        }

        //-----------< called by cells wanting to forward message to other cells >-------↙
    ---------
        public static void transmit(string message)
        {
            MatrixQueue.enQ(message);
            //this enqueue will call Messaging::forwardRequest for the dequeue operation
        }

        //-----< is this cell plugged-in >-------------------------------------
        private static bool checkCells(Guid CellID, out Cell c)
        {
            Object ID = CellID;
            lock(Cells)
            {
                c = (Cell)Cells[ID];
            }
            if(c==null)
                return false;
            else
                return true;
        }

        //-----< does this clone exist >---------------------------------------
        private static bool checkClones(Guid CellID, out Cell c)
        {
            Object ID = CellID;
            lock(Clones)
            {
                c = (Cell)Clones[ID];
            }
            if(c==null)
                return false;
            else
                return true;
        }

        //-----------< cell notification >-------------------------------------------
        public static void send(Object message, Guid CellID)
        {
            Cell c;
            if(checkCells(CellID, out c))
            {
                c.accept((string)message);
            }
            else if(checkClones(CellID, out c))
            {
                c.accept((string)message);
            }
```

```
            else
            {
                throw new Exception("Cell not found in send operation ");
            }
        }

        //-----------< cell notification >-----------------------------------------
        public static void sendResponse(Object message, Guid CellID)
        {
            Cell c;
            if(checkCells(CellID, out c))
            {
                c.accept((string)message,true);
            }
            else if(checkClones(CellID, out c))
            {
                c.accept((string)message,true);
            }
            else
            {
                throw new Exception("Cell not found in send operation ");
            }
        }


        //-----------< clone cell notification >---------------------------------
        public static void sendClone(Object message, Guid CellID)
        {
            Object ID = CellID;
            lock(Clones)
            {
                Cell c = (Cell)Clones[ID];
                if(c!=null)
                {
                    Console.WriteLine("[ Sending to " + CellID.ToString() + " ... ");
                    c.accept((string)message);
                }
                else
                    throw new Exception("Cell not found in send operation " + "(");
            }
        }


        //-----------< cell discovery based on capability >-----------------------------↙
    -------------
        public static string discover(string MessageType)
        {
            lock(Cells)
            {
                IDictionaryEnumerator ide = Cells.GetEnumerator();
                while(ide.MoveNext())
                {
                    if(((Cell)ide.Value).queryCapability(MessageType))
                    {
                        return ((Guid)ide.Key).ToString();
                    }
                }
            }
            return null; //Message not Supported
        }
    }
}
```

```
////////////////////////////////////////////////////////////////
//  ExecObject.cs    Implements an "execution object" that     //
//                   assists in packaging of arguments, return //
//                   values and other information related to   //
//                   related to requests and responses between //
//                   cooperating Cells in the Matrix.          //
//  ver 1.0                                                    //
//                                                             //
//  Language:      C#                                          //
//  Platform:      Dell Dimension Pentium 4, Windows 2000      //
//  Course:        CSE997 Master's Thesis                      //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>           //
//                                                             //
////////////////////////////////////////////////////////////////
using System;

namespace Matrix
{

    public class ExecObject
    {
        bool Request;
        string RequestCellGuid;
        string ResponseCellGuid;
        string RequestID;
        string MessageName;
        Object ReturnVal;
        Object[] Params;

        //-----------< constructor >-------------------------------------------
        public ExecObject(bool _Request, string _RequestCellGuid, string
    _ResponseCellGuid, string  _RequestID, string _MessageName, Object _ReturnVal, Object
    [] _Params)
        {
            Request = _Request;
            RequestCellGuid = _RequestCellGuid;
            ResponseCellGuid =  _ResponseCellGuid;
            RequestID = _RequestID;
            MessageName =  _MessageName;
            ReturnVal =  _ReturnVal;
            Params =  _Params;
        }

        //-----------< constructor >-------------------------------------------
        public ExecObject(bool _Request, string _RequestID, Object _ReturnVal)
        {
            if(_Request == true)
                throw new Exception("Insufficient number of arguments for request
    construction");

            else
            Request = _Request;
            ReturnVal =  _ReturnVal;
            RequestID = _RequestID;
        }

        //-----------< does the execution object represent a request >-------------------
    -------------------------
        public bool isRequest()
        {
            return Request;
        }

        //-----------< get return value of response >---------------------------------
    --------
        public Object getReturnValue()
```

```
        {
            return ReturnVal;
        }


        //-----------< return MessageType of Execution Object >-----------------
        public string getMessageType()
        {
            return MessageName;
        }

        //-----------< return RequestCellGuid of Execution Object >-----------------
        public string getRequestCellGuid()
        {
            return RequestCellGuid;
        }

        //-----------< return ResponseCellGuid of Execution Object >-----------------
        public string getResponseCellGuid()
        {
            return ResponseCellGuid;
        }

        //-----------< return RequestID of Execution Object >-----------------
        public string getRequestID()
        {
            return RequestID;
        }

        //-----------< return Params of Execution Object >-----------------
        public Object[] getParams()
        {
            return Params;
        }

    }
}
```

```
/////////////////////////////////////////////////////////////////////
//  Cell.cs          Specifies an the type "Clone" to be       //
//                   used by cells requiring stateful          //
//                   collaboration with other Cells.           //
//  ver 1.0                                                    //
//                                                             //
//  Language:      C#                                          //
//  Platform:      Dell Dimension Pentium 4, Windows 2000      //
//  Course:        CSE997 Master's Thesis                      //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>           //
//                                                             //
/////////////////////////////////////////////////////////////////////
using System;

namespace Matrix
{
    public class Clone
    {
        Guid _guid;

        //-----< constructor >-------------------------------------------------
        public Clone(Guid g)
        {
            _guid = g;
        }

        //-----< return guid >-------------------------------------------------
        public Guid guid()
        {
            return _guid;
        }
    }
}
```

```
/////////////////////////////////////////////////////////////////
//  Cell.cs          Specifies an interface "Cell" that        //
//                   defines the protocol to be followed by    //
//                   all valid Cells in the Software Matrix.   //
//  ver 1.0                                                    //
//                                                             //
//  Language:      C#                                          //
//  Platform:      Dell Dimension Pentium 4, Windows 2000      //
//  Course:        CSE997 Master's Thesis                      //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>           //
//                                                             //
/////////////////////////////////////////////////////////////////
using System;
using System.Threading;
using System.Diagnostics;
using System.Collections;

namespace Matrix
{
    public delegate void ReceiveDelegate(Object queue, ReceiveInfoEventArgs args);

    public interface Cell
    {
        void extract(object msg);   //extract request and response messages
        bool register();            //register with the Matrix
        bool unregister();          //unregister with the Matrix
        void accept(string message);//accept queue a request message
        void accept(string message, bool Response); //accept and queue a response message

        bool send(string message, Guid ID, int argc, Object[] args);//send a message to a
     different Cell
        void start();               //entry point
        Guid getID();               //return Cell ID
        bool queryCapability(string capability);//query message handling capabilities

        Clone getClone(string MessageName);//collaborate with Cell clone
        Object syncSend(string MessageName, Object[] Params);//synchronous send
        Object syncSend(string MessageName, Object[] Params, Clone clone);//synchronous
    send to clone
        void addResponse(Guid RequestID);//return response
        void removeResponse(Guid RequestID);//extract response
        bool checkAndRetrieveResponse(Guid RequestID);//check for response
        Object process(ExecObject obj);//process requests
        event ReceiveDelegate OnReceive;
    }
}
```

```
/////////////////////////////////////////////////////////////////////
//  BlockingQueue.cs File implementing a reusable                  //
//                   blocking queue.                               //
//  ver 1.0                                                        //
//                                                                 //
//  Language:      C#                                              //
//  Platform:      Dell Dimension Pentium 4, Windows 2000          //
//  Course:        CSE997 Master's Thesis                          //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>               //
//                                                                 //
/////////////////////////////////////////////////////////////////////

using System;
using System.Collections;
using System.Threading;

namespace Matrix
{
    ///////////////////////////////////////////////////////////////////////////
    //           A reusable class that encapsulates a Blocking Queue         //
    //                                                                       //
    ///////////////////////////////////////////////////////////////////////////
    public class BlockingQueue
    {
        Queue queue;
        AutoResetEvent are;
        Thread DequeueThread;
        public delegate void DequeueDelegate(Object Obj);
        DequeueDelegate dequeue;

        //-----------< constructor >---------------
        public BlockingQueue(DequeueDelegate _dequeue)
        {
            dequeue = _dequeue;
            queue = new Queue(20);
            are = new AutoResetEvent(true);
            DequeueThread = new Thread(new ThreadStart(deQ));
            //DequeueThread.IsBackground = true;
            DequeueThread.Start();
        }

        //-----------< function to perform dequeue operation >---------------
        public void deQ()
        {
            Object o;
            while(true)
            {
                are.WaitOne();
                while(queue.Count > 0)
                {
                    lock(queue)
                    {
                        o = queue.Dequeue();
                    }
                    //calling target function through delegate
                    dequeue(o);
                }
                are.Reset();
            }
        }

        //-----------< function to perform enqueue operation >---------------
        public void enQ(Object o)
        {
            lock(queue)
            {
```

```
                queue.Enqueue(o);
                are.Set();
            }
        }
    }



    //----------< TEST STUB >------------------------------------------
   #if(TEST_BLOCKINGQUEUE)
...
   #endif
}
```

Sample Cells

```
/////////////////////////////////////////////////////////////////////
//  FileManager.cs   File implementing a "Cell".                   //
//                   Implements common file operations.            //
//                   Possesses the capability to handle the        //
//                   following Message Types:                      //
//                   "FileManager.ReadText"                        //
//                   "FileManager.ReadBinary"                      //
//                   "FileManager.WriteText"                       //
//                   "FileManager.WriteBinary"                     //
//  ver 1.0                                                        //
//                                                                 //
//  Language:      C#                                              //
//  Platform:      Dell Dimension Pentium 4, Windows 2000          //
//  Course:        CSE997 Master's Thesis                          //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>               //
//                                                                 //
/////////////////////////////////////////////////////////////////////

using System;
using System.IO;
using System.Text;

namespace Matrix
{
    public class FileManager : Cell
    {
        #region Wizard Generated Code
        private Guid ID;
        private Messaging messager;
        private BlockingQueue queue;
        private BlockingQueue ResponseQueue;
        private System.Threading.Thread StartThread;
        private System.Collections.Hashtable ResponseList;
        protected System.Collections.Hashtable CapabilityList;
        public event ReceiveDelegate OnReceive;

        //-----------< process messages pulled out of the queue >---------------
        public void extract(object msg)
        {
            ExecObject eo = messager.extract(msg);
            if(eo==null)
                return;
            if(eo.isRequest())
            {
                Object result = process(eo);
                Guid RequestCellGUID;
                String response = messager.buildResponse(msg,result,out RequestCellGUID);
                Matrix.send(response, RequestCellGUID);
            }
                //Response Received
            else
            {
                ReceiveInfoEventArgs rargs = new ReceiveInfoEventArgs(new Guid(eo.
    getRequestID()), eo.getReturnValue());
                if(OnReceive!=null)
                    OnReceive(this, rargs);
            }
        }

        //-----------< method to register cell in matrix >----------------------
        public bool register()
        {
            Matrix.register((Cell)this);
            StartThread = new System.Threading.Thread(new System.Threading.ThreadStart(
    start));
            StartThread.Start();
```

```
        return true;
    }

    //-----------< method to unregister cell in matrix >--------------------
    public bool unregister()
    {
        Matrix.unregister(this);
        StartThread.Abort();
        return true;
    }

    //-----------< accept a request message >-------------------------------------
    public void accept(string message)
    {
        accept(message,false);
    }

    //-----< accept a response message >-----------------------------------------
    public void accept(string message, bool Response)
    {
        Object msg = message;
        if(Response == false)
            queue.enQ(msg);
        else
            ResponseQueue.enQ(msg);
    }


    //-----------< send message to another registered cell >----------------
    public bool send(string message, Guid ID, int argc, Object[] args)
    {
        return true;
    }

    //-----------< get unique cell ID >-------------------------------------
    public Guid getID()
    {
        return ID;
    }

    //-----------< capability interrogation >-------------------------------------
    public bool queryCapability(string capability)
    {
        if(CapabilityList.ContainsKey(capability))
            return true;
        else
            return false;
    }

    //-----------< method to send execution messages to other cells >-------
    public Object syncSend(string MessageName, Object[] Params)
    {
        string RequestCellGuid = ID.ToString();
        Object Result = null;
        Messaging msg = new Messaging();
        Guid RequestID;
        string xml = msg.buildRequest(RequestCellGuid,MessageName,
            Result, Params, out RequestID, Guid.Empty);
        Matrix.transmit(xml);
        SyncWait sync = new SyncWait(this, RequestID);
        Result = sync.wait();
        return Result;
    }

    //-----------< method to send execution messages to clone cells >-------
    public Object syncSend(string MessageName, Object[] Params, Clone clone)
```

```
        {
            Guid guid = clone.guid();
            string RequestCellGuid = ID.ToString();
            Object Result = null;
            Messaging msg = new Messaging();
            Guid RequestID;
            string xml = msg.buildRequest(RequestCellGuid,MessageName,
                Result, Params, out RequestID,guid);
            Matrix.transmit(xml);
            SyncWait sync = new SyncWait(this, RequestID);
            Result = sync.wait();
            return Result;
        }

        //-----------< method to obtain a Cell clone for stateful operation >---
        public Clone getClone(string MessageName)
        {
            Guid g = Matrix.createClone(MessageName);
            if(g.CompareTo(Guid.Empty) == 0)
                throw new Exception("Could not locate cell to handle: "+
                    MessageName);
            else
                return new Clone(g);
        }

        //-----< check if a Cell ID is valid >-----------------------------------
        bool isValid(Clone clone)
        {
            Guid CellID = clone.guid();
            if(CellID.CompareTo(Guid.Empty)==0)
                return false;
            else
                return true;
        }

        //-----------<add response received flag to ResponseList >--------------
        public void addResponse(Guid RequestID)
        {
            ResponseList.Add(RequestID, null);
        }

        //-----------<remove response received flag to ResponseList >-----------
        public void removeResponse(Guid RequestID)
        {
            if(ResponseList.Contains(RequestID))
            {
                ResponseList.Remove(RequestID);
            }
        }

        //-----------< retrieve response >----------------------------------------
        public bool checkAndRetrieveResponse(Guid RequestID)
        {

            if(ResponseList.Contains(RequestID))
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        //-----< initialize cell >-----------------------------------------------
        private void initializeCell()
```

```
        {
            if(ID.CompareTo(Guid.Empty)==0)
            {
                ID = Guid.NewGuid(); //ID//
            }
            messager = new Messaging();
            queue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extract));
            ResponseQueue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extract));
            CapabilityList = new System.Collections.Hashtable();
            ResponseList = new System.Collections.Hashtable();
        }
        #endregion

        //-----< cell entry point >------------------------------------------
        public void start()
        {
            //TODO: add executive processing if required
            //default: do nothing
        }

        //-----------< constructor >-----------------------------------------
        public FileManager()
        {
            //
            initializeCell();
            //TODO:
            //Add constructor logic here
            CapabilityList.Add("FileManager.ReadText", null);
            CapabilityList.Add("FileManager.ReadBinary", null);
            CapabilityList.Add("FileManager.WriteText", null);
            CapabilityList.Add("FileManager.WriteBinary", null);
        }

        //-----------< calls to message-specific processing >----------------
        public Object process(ExecObject eo)
        {
            ReturnObject ro = null;
            Exception exception = null;
            Object ret=null;
            try
            {   //
                //TODO:
                //add calls to processing logic here
                string MessageType = eo.getMessageType();
                switch(MessageType)
                {
                    case "FileManager.ReadText":
                        ret = process_FileManager_ReadText(eo);
                        break;
                    case "FileManager.ReadBinary":
                        ret = process_FileManager_ReadBinary(eo);
                        break;
                    case "FileManager.WriteText":
                        ret = process_FileManager_WriteText(eo);
                        break;
                    case "FileManager.WriteBinary":
                        ret = process_FileManager_WriteBinary(eo);
                        break;
                }
            }
            catch(Exception ex)
            {
                exception = ex;
            }
```

```
            ro = new ReturnObject(ret,exception);
            return ro;
        }


        //-----------< method to process "FileManager.ReadText" messages >---------------
        private Object process_FileManager_ReadText(ExecObject obj)
        {
            string Result = null;
            Object[] Params = obj.getParams();
            if(Params.Length!=1)
                throw new Exception("Incorrect Input for FileManager.ReadText message
    processing");
            else
            {
                Result = readTextFile((string)Params[0]);
            }

            return Result;
        }


        //-----------< method to process "FileManager.ReadBinary" messages >-------------
    --
        private Object process_FileManager_ReadBinary(ExecObject obj)
        {
            byte[] Result = null;
            Object[] Params = obj.getParams();
            if(Params.Length!=1)
                throw new Exception("Incorrect Input for FileManager.ReadBinary message
    processing");
            else
            {
                Result = readBinaryFile((string)Params[0]);
            }
            return Result;
        }


        //-----------< method to process "FileManager.WriteBinary" messages >------------
    ---
        private Object process_FileManager_WriteText(ExecObject obj)
        {
            bool Result = false;
            Object[] Params = obj.getParams();
            if(Params.Length!=2)
                throw new Exception("Incorrect Input for FileManager.WriteText message
    processing");
            else
            {
                Result = writeTextFile((string)Params[0], (string)Params[1]);
            }
            return Result;
        }


        //-----------< method to process "FileManager.WriteBinary" messages >------------
    ---
        private Object process_FileManager_WriteBinary(ExecObject obj)
        {
            bool Result = false;
            Object[] Params = obj.getParams();
            if(Params.Length!=2)
                throw new Exception("Incorrect Input for FileManager.WriteBinary message
    processing");
            else
```

```
            {
                Result = writeTextFile((string)Params[0], (string)Params[1]);
            }
            return Result;
        }

        //-----------< method to read a text file >----------------------------
        public string readTextFile(string FilePath)
        {
            try
            {
                StreamReader sr = new StreamReader(FilePath);
                return sr.ReadToEnd();
            }
            catch(Exception e)
            {
                Console.WriteLine(e.Message);
            }
            return "";
        }

        //-----------< method to read a binary file >---------------------------
        public byte[] readBinaryFile(string FilePath)
        {
            byte[] buffer;
            try
            {
                FileStream file = File.OpenRead(FilePath);
                FileInfo f = new FileInfo(FilePath);
                buffer = new byte[f.Length];
                file.Read(buffer,0,(int)f.Length);
                file.Close();
            }
            catch(Exception e)
            {
                Console.WriteLine(e.Message);
                buffer=null;
            }
            return buffer;
        }

        //-----------< method to write a text file >----------------------------
        public bool writeTextFile(string FilePath, string contents)
        {
            UnicodeEncoding ue = new UnicodeEncoding();
            byte[] buffer = ue.GetBytes(contents);
            return writeBinaryFile(FilePath, buffer);
        }

        //-----------< method to write a binary file >---------------------------
        public bool writeBinaryFile(string FilePath, byte[] contents)
        {
            FileStream file;
            file = File.Create(FilePath);
            file.Write(contents,0,contents.Length);
            file.Close();
            return true;
        }
    }


    //----------< TEST STUB >-------------------------------------------
    #if(TEST_FILEMANAGER)
...
    #endif
```

```
}
```

```
////////////////////////////////////////////////////////////////
//  Convolution.cs   File implementing a "Cell".              //
//                   Possesses the capability to handle the   //
//                   following Message Types:                 //
//                   "DSP.Convolution"                        //
//  ver 1.0                                                   //
//                                                            //
//  Language:      C#                                         //
//  Platform:      Dell Dimension Pentium 4, Windows 2000     //
//  Course:        CSE997 Master's Thesis                     //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>          //
//                                                            //
////////////////////////////////////////////////////////////////

using System;

namespace Matrix
{
    public class Convolution : Cell
    {
        #region Wizard Generated Code
        private Guid ID;
        private Messaging messager;
        private BlockingQueue queue;
        private BlockingQueue ResponseQueue;
        private System.Threading.Thread StartThread;
        private System.Collections.Hashtable ResponseList;
        protected System.Collections.Hashtable CapabilityList;
        public event ReceiveDelegate OnReceive;

        //-----------< process messages pulled out of the queue >---------------
        public void extract(object msg)
        {
            ExecObject eo = messager.extract(msg);
            if(eo==null)
                return;
            if(eo.isRequest())
            {
                Object result = process(eo);
                Guid RequestCellGUID;
                String response = messager.buildResponse(msg,result,out RequestCellGUID);
                Matrix.send(response, RequestCellGUID);
            }
                //Response Received
            else
            {
                ReceiveInfoEventArgs rargs = new ReceiveInfoEventArgs(new Guid(eo.
    getRequestID()), eo.getReturnValue());
                if(OnReceive!=null)
                    OnReceive(this, rargs);
            }
        }

        //-----------< method to register cell in matrix >----------------------
        public bool register()
        {
            Matrix.register((Cell)this);
            StartThread = new System.Threading.Thread(new System.Threading.ThreadStart(
    start));
            StartThread.Start();
            return true;
        }

        //-----------< method to unregister cell in matrix >--------------------
        public bool unregister()
        {
```

```
            Matrix.unregister(this);
            StartThread.Abort();
            return true;
        }

        //-----------< accept a request message >-------------------------------------
        public void accept(string message)
        {
            accept(message,false);
        }

        //-----< accept a response message >-----------------------------------------
        public void accept(string message, bool Response)
        {
            Object msg = message;
            if(Response == false)
                queue.enQ(msg);
            else
                ResponseQueue.enQ(msg);
        }

        //-----------< send message to another registered cell >----------------
        public bool send(string message, Guid ID, int argc, Object[] args)
        {
            return true;
        }

        //-----------< get unique cell ID >------------------------------------
        public Guid getID()
        {
            return ID;
        }

        //-----------< capability interrogation >-------------------------------------
        public bool queryCapability(string capability)
        {
            if(CapabilityList.ContainsKey(capability))
                return true;
            else
                return false;
        }

        //-----------< method to send execution messages to other cells >-------
        public Object syncSend(string MessageName, Object[] Params)
        {
            string RequestCellGuid = ID.ToString();
            Object Result = null;
            Messaging msg = new Messaging();
            Guid RequestID;
            string xml = msg.buildRequest(RequestCellGuid,MessageName,
                Result, Params, out RequestID, Guid.Empty);
            SyncWait sync = new SyncWait(this, RequestID);
            Result = sync.wait();
            return Result;
        }

        //-----------< method to send execution messages to clone cells >-------
        public Object syncSend(string MessageName, Object[] Params, Clone clone)
        {
            Guid guid = clone.guid();
            string RequestCellGuid = ID.ToString();
            Object Result = null;
            Messaging msg = new Messaging();
            Guid RequestID;
            string xml = msg.buildRequest(RequestCellGuid,MessageName,
                Result, Params, out RequestID,guid);
```

```
            Matrix.transmit(xml);
            SyncWait sync = new SyncWait(this, RequestID);
            Result = sync.wait();
            return Result;
        }

        //-----------< method to obtain a Cell clone for stateful operation >---
        public Clone getClone(string MessageName)
        {
            Guid g = Matrix.createClone(MessageName);
            if(g.CompareTo(Guid.Empty) == 0)
                throw new Exception("Could not locate cell to handle: "+
                    MessageName);
            else
                return new Clone(g);
        }

        //-----< check if a Cell ID is valid >-----------------------------------
        bool isValid(Clone clone)
        {
            Guid CellID = clone.guid();
            if(CellID.CompareTo(Guid.Empty)==0)
                return false;
            else
                return true;
        }

        //-----------<add response received flag to ResponseList >--------------
        public void addResponse(Guid RequestID)
        {
            ResponseList.Add(RequestID, null);
        }

        //-----------<remove response received flag to ResponseList >-----------
        public void removeResponse(Guid RequestID)
        {
            if(ResponseList.Contains(RequestID))
            {
                ResponseList.Remove(RequestID);
            }
        }

        //-----------< retrieve response >---------------------------------------
        public bool checkAndRetrieveResponse(Guid RequestID)
        {

            if(ResponseList.Contains(RequestID))
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        //-----< initialize cell >-----------------------------------------------
        private void initializeCell()
        {

            if(ID.CompareTo(Guid.Empty)==0)
            {
                ID = Guid.NewGuid(); //ID//
            }
            messager = new Messaging();
            queue = new BlockingQueue(
```

```
                new BlockingQueue.DequeueDelegate(extract));
            ResponseQueue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extract));
            CapabilityList = new System.Collections.Hashtable();
            ResponseList = new System.Collections.Hashtable();
        }
        #endregion

        //-----< cell entry point >-----------------------------------------
        public void start()
        {
            //TODO: add executive processing if required
            //default: do nothing
        }

        //-----------< constructor >-----------------------------------------
        public Convolution()
        {
            //
            initializeCell();
            //TODO:
            //Add constructor logic here
            CapabilityList.Add("DSP.Convolution", null);
        }

        //-----------< calls to message-specific processing >-------------------
        public Object process(ExecObject eo)
        {
            ReturnObject ro = null;
            Exception exception = null;
            Object ret=null;
            try
            {   //
                //TODO:
                //add calls to processing logic here
                string MessageType = eo.getMessageType();
                switch(MessageType)
                {
                    case "DSP.Convolution":
                        ret = process_DSP_Convolution(eo);
                        break;
                }
            }
            catch(Exception ex)
            {
                exception = ex;
            }
            ro = new ReturnObject(ret,exception);
            return ro;
        }

        //-----------< method to process "DSP.Convolution" messages >--------------------↙
    ---
        private Object process_DSP_Convolution(ExecObject obj)
        {
            Object[] Params = obj.getParams();
            if(Params.Length!=2)
                throw new Exception("Incorrect Input for Add message processing");
            Double[] Sequence1 = (Double[])Params[0];
            Double[] Sequence2 = (Double[])Params[1];
            Double[] Result = convolve(Sequence1, Sequence2);
            return Result;

        }

        //-------------< convolution of 2 sequences  >-------------------------
```

```
        private static Double[] convolve(Double[] x,  Double[] h)
        {
            Double[] Result = new Double[(x.Length+h.Length)-1];

            Double[] PaddedH = new Double[(x.Length+h.Length)];
            for(int i=0; i<h.Length; i++)
            {
                PaddedH[(PaddedH.Length-i)-1]=h[(h.Length-i)-1];
            }

            Double[] Flip = new Double[(x.Length+h.Length)];
            for(int i=0; i<x.Length; i++)
            {
                Flip[i]=x[((x.Length)-i)-1];
            }

            //Performing convolution
            ShiftLeft(ref PaddedH);
            for(int i=0; i<Result.Length; i++)
            {
                Result[i] = productAndSum(PaddedH, Flip);
                ShiftLeft(ref PaddedH);
            }
            return Result;
        }


        //---------------< helper function for product and sum >-----------
        private static Double productAndSum(Double[] x, Double[] y)
        {
            Double result = 0;
            //check the sizes of the arrays to see if they are the same size
            if(x.Length != x.Length)
                throw(new Exception("Arrays have to be padded to be of the same size"));
            for(int i=0; i<x.Length; i++)
            {
                result+= x[i]*y[i];
            }
            return result;
        }


        //---------------< helper function to shift >-----------
        private static void ShiftLeft(ref Double[] Arr)
        {
            //Double temp = Arr[0];
            for(int i=0; i<(Arr.Length-1); i++)
            {
                Arr[i]=Arr[i+1];
            }
            Arr[Arr.Length-1]=0;
        }
    }
}
```

```
/////////////////////////////////////////////////////////////////////
//  Client.cs     File implementing a "Cell". Acts as a client     //
//                by using the services of other cells in the      //
//                Matrix.                                          //
//  ver 1.0                                                        //
//                                                                 //
//  Language:      C#                                              //
//  Platform:      Dell Dimension Pentium 4, Windows 2000          //
//  Course:        CSE997 Master's Thesis                          //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>               //
//                                                                 //
/////////////////////////////////////////////////////////////////////
using System;
using System.Globalization;

namespace Matrix
{
    public class Client : Cell
    {
        #region Wizard Generated Code
        private Guid ID;
        private Messaging messager;
        private BlockingQueue queue;
        private BlockingQueue ResponseQueue;
        private System.Threading.Thread StartThread;
        private System.Collections.Hashtable ResponseList;
        protected System.Collections.Hashtable CapabilityList;
        public event ReceiveDelegate OnReceive;

        //-----------< process messages pulled out of the queue >---------------
        public void extract(object msg)
        {
            ExecObject eo = messager.extract(msg);
            if(eo==null)
                return;
            if(eo.isRequest())
            {
                Object result = process(eo);
                Guid RequestCellGUID;
                String response = messager.buildResponse(msg,result,out RequestCellGUID);
                Matrix.send(response, RequestCellGUID);
            }
                //Response Received
            else
            {
                ReceiveInfoEventArgs rargs = new ReceiveInfoEventArgs(new Guid(eo.
    getRequestID()), eo.getReturnValue());
                if(OnReceive!=null)
                    OnReceive(this, rargs);
            }
        }

        //-----------< method to register cell in matrix >----------------------
        public bool register()
        {
            Matrix.register((Cell)this);
            StartThread = new System.Threading.Thread(new System.Threading.ThreadStart(
    start));
            StartThread.Start();
            return true;
        }

        //-----------< method to unregister cell in matrix >--------------------
        public bool unregister()
        {
            Matrix.unregister(this);
```

```
            StartThread.Abort();
            return true;
        }

        //-----------< accept a request message >-------------------------------------
        public void accept(string message)
        {
             accept(message,false);
        }

        //-----< accept a response message >-----------------------------------------
        public void accept(string message, bool Response)
        {
            Object msg = message;
            if(Response == false)
                queue.enQ(msg);
            else
                ResponseQueue.enQ(msg);
        }

        //-----------< send message to another registered cell >----------------
        public bool send(string message, Guid ID, int argc, Object[] args)
        {
            return true;
        }

        //-----------< get unique cell ID >------------------------------------
        public Guid getID()
        {
            return ID;
        }

        //-----------< capability interrogation >------------------------------------
        public bool queryCapability(string capability)
        {
            if(CapabilityList.ContainsKey(capability))
                return true;
            else
                return false;
        }

        //-----------< method to send execution messages to other cells >-------
        public Object syncSend(string MessageName, Object[] Params)
        {
            return syncSend(MessageName,Params,new Clone(Guid.Empty));
        }

        //-----------< method to send execution messages to clone cells >-------
        public Object syncSend(string MessageName, Object[] Params, Clone clone)
        {
            Guid guid = clone.guid();
            string RequestCellGuid = ID.ToString();
            Object Result = null;
            Messaging msg = new Messaging();
            Guid RequestID;
            string xml = msg.buildRequest(RequestCellGuid,MessageName,
                Result, Params, out RequestID,guid);
            Matrix.transmit(xml);
            SyncWait sync = new SyncWait(this, RequestID);
            Result = sync.wait();
            return Result;
        }

        //-----------< method to obtain a Cell clone for stateful operation >---
        public Clone getClone(string MessageName)
        {
```

```
            Guid g = Matrix.createClone(MessageName);
            if(g.CompareTo(Guid.Empty) == 0)
                throw new Exception("Could not locate cell to handle: "+
                    MessageName);
            else
                return new Clone(g);
        }

        //-----< check if a Cell ID is valid >---------------------------------
        bool isValid(Clone clone)
        {
            Guid CellID = clone.guid();
            if(CellID.CompareTo(Guid.Empty)==0)
                return false;
            else
                return true;
        }

        //-----------<add response received flag to ResponseList >--------------
        public void addResponse(Guid RequestID)
        {
            ResponseList.Add(RequestID, null);
        }

        //-----------<remove response received flag to ResponseList >-----------
        public void removeResponse(Guid RequestID)
        {
            if(ResponseList.Contains(RequestID))
            {
                ResponseList.Remove(RequestID);
            }
        }

        //-----------< retrieve response >-------------------------------------
        public bool checkAndRetrieveResponse(Guid RequestID)
        {

            if(ResponseList.Contains(RequestID))
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        //-----< initialize cell >---------------------------------------------
        private void initializeCell()
        {
            if(ID.CompareTo(Guid.Empty)==0)
            {
                ID = Guid.NewGuid(); //ID//
            }
            messager = new Messaging();
            queue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extract));
            ResponseQueue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extract));
            CapabilityList = new System.Collections.Hashtable();
            ResponseList = new System.Collections.Hashtable();
        }
        #endregion

        //-----------< entry point of execution for each cell >----------------
        public void start()
```

```
        {
            //TODO: add executive processing if required
            //default: do nothing

            Object[] Params; Object Return;
            Console.WriteLine("\t[starting " + this.ToString() +" execution]");
            Messaging msg = new Messaging();


            string DispParam = "Performing convolution of 2 discrete sequences";
            Params = new Object[]{DispParam};
            Return = syncSend("GUI.DrawBox",Params);
            Console.WriteLine("\n");
            Console.WriteLine(Return);

            string FilePath = "DSP.Convolution.txt";
            Params = new Object[]{FilePath};
            Params = new Object[]{syncSend("FileManager.ReadText",Params)};
            Console.WriteLine(syncSend("GUI.DrawBox",Params));


            Double[] seq1 = new Double[]{1,2,5,4,7,8,3};
            Double[] seq2 = new Double[]{2,3,6,978,52,1.3665,2,0,1,0,3};
            Params = new Object[]{seq1, seq2};
            Object Res = syncSend("DSP.Convolution",Params);
            Console.WriteLine("\n\n");
            Console.WriteLine("Sequence A:\t" + arrayToString(seq1));
            Console.WriteLine("Sequence B:\t" + arrayToString(seq2));
            Console.Write("Convolved:\t");
            Console.WriteLine(arrayToString((Double[])Res));
            Console.WriteLine("\n\n");


            Double d = 5.1212, e = 12.24587;
            Object[] Args = new Object[]{d,e};
            Console.WriteLine("Adding:\t" + d + "  " + e);
            Object Result = syncSend("Math.Add",Args);
            Console.WriteLine("Result:\t" + Result);
        }



        //-----------< calls to message-specific processing >-------------------
        public Object process(ExecObject eo)
        {
            ReturnObject ro = null;
            Exception exception = null;
            Object ret=null;
            try
            {
                //TODO:
                //add calls to processing logic here
            }
            catch(Exception ex)
            {
                exception = ex;
            }
            ro = new ReturnObject(ret,exception);
            return ro;
        }

        //-----------< constructor >---------------------------------------------
        public Client()
        {
            //
            initializeCell();
```

```
            //TODO:
            //Add constructor logic here
        }

        //-----------------< helper function to print distribution >--------------
        private static string arrayToString(Double[] d)
        {
            string s = null;
            for(int i=0; i<d.Length; i++)
            {
                NumberFormatInfo nfi = new CultureInfo( "en-US", false ).NumberFormat;
                s=s+ "   " + d[i].ToString("N", nfi)  ;
            }
            return s;
        }
    }
}
```

```
/////////////////////////////////////////////////////////////////////
//  FileSearch.cs    File implementing a "Cell".                   //
//                   Possesses the capability to handle the        //
//                   following Message Types:                      //
//                   "SU.FileSearch.SearchDirsFiles"               //
//  ver 1.0                                                        //
//                                                                 //
//  Language:      C#                                              //
//  Platform:      Dell Dimension Pentium 4, Windows 2000          //
//  Course:        CSE997 Master's Thesis                          //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>               //
//                                                                 //
/////////////////////////////////////////////////////////////////////
using System;
using Matrix;

namespace FileSearch
{
    public class FileSearch : Cell
    {

        #region Wizard Generated Code
        private Guid ID;
        private Messaging messager;
        private BlockingQueue queue;
        private System.Threading.Thread StartThread;
        private System.Collections.Hashtable ResponseList;
        protected System.Collections.Hashtable CapabilityList;
        public event ReceiveDelegate OnReceive;
        private BlockingQueue ResponseQueue;

        //-----------< process messages pulled out of the queue >---------------
        public void extract(object msg)
        {
            ExecObject eo = messager.extract(msg);
            if(eo==null)
                return;
            if(eo.isRequest())
            {
                Object result = process(eo);
                Guid RequestCellGUID;
                String response = messager.buildResponse(msg,result,out RequestCellGUID);
                Matrix.Matrix.sendResponse(response, RequestCellGUID); //SEND RESPONSE
            }
                //Response Received
            else
            {
                ReceiveInfoEventArgs rargs = new ReceiveInfoEventArgs(new Guid(eo.
    getRequestID()), eo.getReturnValue());
                if(OnReceive!=null)
                    OnReceive(this, rargs);
            }
        }


        //-----< process responses out of the response queue >-------------------------
        public void extractResponse(object msg)
        {
            ExecObject eo = messager.extract(msg);
            if(eo==null)
                return;
            if(!eo.isRequest())
            {
                ReceiveInfoEventArgs rargs = new ReceiveInfoEventArgs(new Guid(eo.
    getRequestID()), eo.getReturnValue());
                if(OnReceive!=null)
```

```
                    OnReceive(this, rargs);
                }
            }

            //-----------< method to register cell in matrix >----------------------
            public bool register()
            {
                Matrix.Matrix.register((Cell)this);
                StartThread = new System.Threading.Thread(new System.Threading.ThreadStart(
        start));
                StartThread.Start();
                return true;
            }

            //-----------< method to unregister cell in matrix >--------------------
            public bool unregister()
            {
                Matrix.Matrix.unregister(this);
                StartThread.Abort();
                return true;
            }

            //-----------< accept a request message >-------------------------------------
            public void accept(string message)
            {
                accept(message,false);
            }

            //-----< accept a response message >-----------------------------------------
            public void accept(string message, bool Response)
            {
                Object msg = message;
                if(Response == false)
                    queue.enQ(msg);
                else
                    ResponseQueue.enQ(msg);
            }


            //-----------< send message to another registered cell >----------------
            public bool send(string message, Guid ID, int argc, Object[] args)
            {
                return true;
            }

            //-----------< get unique cell ID >-------------------------------------
            public Guid getID()
            {
                return ID;
            }

            //-----------< capability interrogation >-------------------------------------
            public bool queryCapability(string capability)
            {
                if(CapabilityList.ContainsKey(capability))
                    return true;
                else
                    return false;
            }

            //-----------< method to send execution messages to other cells >-------
            public Object syncSend(string MessageName, Object[] Params)
            {
                string RequestCellGuid = ID.ToString();
                Object Result = null;
                Messaging msg = new Messaging();
```

```
        Guid RequestID;
        string xml = msg.buildRequest(RequestCellGuid,MessageName,
            Result, Params, out RequestID, Guid.Empty);
        Matrix.Matrix.transmit(xml);
        SyncWait sync = new SyncWait(this, RequestID);
        Result = sync.wait();
        return Result;
    }

    //-----------< method to send execution messages to clone cells >-------
    public Object syncSend(string MessageName, Object[] Params, Clone clone)
    {
        Guid guid = clone.guid();
        string RequestCellGuid = ID.ToString();
        Object Result = null;
        Messaging msg = new Messaging();
        Guid RequestID;
        string xml = msg.buildRequest(RequestCellGuid,MessageName,
            Result, Params, out RequestID,guid);
        Matrix.Matrix.transmit(xml);
        SyncWait sync = new SyncWait(this, RequestID);
        Result = sync.wait();
        return Result;
    }

    //-----------< method to obtain a Cell clone for stateful operation >---
    public Clone getClone(string MessageName)
    {
        Guid g = Matrix.Matrix.createClone(MessageName);
        if(g.CompareTo(Guid.Empty) == 0)
            throw new Exception("Could not locate cell to handle: "+
                MessageName);
        else
            return new Clone(g);
    }

    //-----< check if a Cell ID is valid >-----------------------------------
    bool isValid(Clone clone)
    {
        Guid CellID = clone.guid();
        if(CellID.CompareTo(Guid.Empty)==0)
            return false;
        else
            return true;
    }

    //-----------<add response received flag to ResponseList >--------------
    public void addResponse(Guid RequestID)
    {
        ResponseList.Add(RequestID, null);
    }

    //-----------<remove response received flag to ResponseList >-----------
    public void removeResponse(Guid RequestID)
    {
        if(ResponseList.Contains(RequestID))
        {
            ResponseList.Remove(RequestID);
        }
    }

    //-----------< retrieve response >---------------------------------------
    public bool checkAndRetrieveResponse(Guid RequestID)
    {

        if(ResponseList.Contains(RequestID))
```

```
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        //-----< initialize cell >-------------------------------------------
        private void initializeCell()
        {
            if(ID.CompareTo(Guid.Empty)==0)
            {
                ID = Guid.NewGuid(); //ID//
            }
            messager = new Messaging();
            queue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extract));
            ResponseQueue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extractResponse));
            CapabilityList = new System.Collections.Hashtable();
            ResponseList = new System.Collections.Hashtable();
        }
        #endregion

        //-----< cell entry point >------------------------------------------
        public void start()
        {
            //TODO: add executive processing if required
            //default: do nothing
        }
        //-----------< calls to message-specific processing >-----------------
        public Object process(ExecObject eo)
        {
            ReturnObject ro = null;
            Exception exception = null;
            Object ret=null;
            try
            {   //
                //TODO:
                //add calls to processing logic here
                string MessageType = eo.getMessageType();
                switch(MessageType)
                {
                    case "SU.FileSearch.SearchDirsFiles":
                        ret = process_SU_FileSearch_SearchDirsFiles(eo);
                        break;

                }
            }
            catch(Exception ex)
            {
                exception = ex;
            }
            ro = new ReturnObject(ret,exception);
            return ro;
        }

        //-----< method to process SU.FileSearch.SearchDirsFiles messages >----------
        private Object process_SU_FileSearch_SearchDirsFiles(ExecObject obj)
        {
            return null;
        }

        //-----------< constructor >-----------------------------------------
```

```
        public FileSearch()
        {
            //
            initializeCell();
            //TODO:
            //Add constructor logic here
            CapabilityList.Add("SU.FileSearch.SearchDirsFiles", null);
        }
    }
}
```

```
/////////////////////////////////////////////////////////////////////
//  GUI.cs           File implementing a "Cell".                   //
//                   Implements common windowing operations        //
//                   for console applications.                     //
//                   Possesses the capability to handle the        //
//                   following Message Types:                      //
//                   "GUI.DrawBox"                                 //
//                   "GUI.DrawTitleBox"                            //
//  ver 1.0                                                        //
//                                                                 //
//  Language:      C#                                              //
//  Platform:      Dell Dimension Pentium 4, Windows 2000          //
//  Course:        CSE997 Master's Thesis                          //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>               //
//                                                                 //
/////////////////////////////////////////////////////////////////////
using System;

namespace Matrix
{
    public class GUI : Cell
    {
        #region Wizard Generated Code
        private Guid ID;
        private Messaging messager;
        private BlockingQueue queue;
        private BlockingQueue ResponseQueue;
        private System.Threading.Thread StartThread;
        private System.Collections.Hashtable ResponseList;
        protected System.Collections.Hashtable CapabilityList;
        public event ReceiveDelegate OnReceive;

        //-----------< process messages pulled out of the queue >---------------
        public void extract(object msg)
        {
            ExecObject eo = messager.extract(msg);
            if(eo==null)
                return;
            if(eo.isRequest())
            {
                Object result = process(eo);
                Guid RequestCellGUID;
                String response = messager.buildResponse(msg,result,out RequestCellGUID);
                Matrix.send(response, RequestCellGUID);
            }
                //Response Received
            else
            {
                ReceiveInfoEventArgs rargs = new ReceiveInfoEventArgs(new Guid(eo.
    getRequestID()), eo.getReturnValue());
                if(OnReceive!=null)
                    OnReceive(this, rargs);
            }
        }

        //-----------< method to register cell in matrix >----------------------
        public bool register()
        {
            Matrix.register((Cell)this);
            StartThread = new System.Threading.Thread(new System.Threading.ThreadStart(
    start));
            StartThread.Start();
            return true;
        }

        //-----------< method to unregister cell in matrix >--------------------
```

```
        public bool unregister()
        {
            Matrix.unregister(this);
            StartThread.Abort();
            return true;
        }

        //-----------< accept a request message >-----------------------------------
        public void accept(string message)
        {
            accept(message,false);
        }

        //-----< accept a response message >----------------------------------------
        public void accept(string message, bool Response)
        {
            Object msg = message;
            if(Response == false)
                queue.enQ(msg);
            else
                ResponseQueue.enQ(msg);
        }


        //-----------< send message to another registered cell >----------------
        public bool send(string message, Guid ID, int argc, Object[] args)
        {
            return true;
        }

        //-----------< get unique cell ID >-----------------------------------
        public Guid getID()
        {
            return ID;
        }

        //-----------< capability interrogation >-----------------------------------
        public bool queryCapability(string capability)
        {
            if(CapabilityList.ContainsKey(capability))
                return true;
            else
                return false;
        }

        //-----------< method to send execution messages to other cells >-------
        public Object syncSend(string MessageName, Object[] Params)
        {
            string RequestCellGuid = ID.ToString();
            Object Result = null;
            Messaging msg = new Messaging();
            Guid RequestID;
            string xml = msg.buildRequest(RequestCellGuid,MessageName,
                Result, Params, out RequestID, Guid.Empty);
            Matrix.transmit(xml);
            SyncWait sync = new SyncWait(this, RequestID);
            Result = sync.wait();
            return Result;
        }

        //-----------< method to send execution messages to clone cells >-------
        public Object syncSend(string MessageName, Object[] Params, Clone clone)
        {
            Guid guid = clone.guid();
            string RequestCellGuid = ID.ToString();
            Object Result = null;
```

```
            Messaging msg = new Messaging();
            Guid RequestID;
            string xml = msg.buildRequest(RequestCellGuid,MessageName,
                Result, Params, out RequestID,guid);
            Matrix.transmit(xml);
            SyncWait sync = new SyncWait(this, RequestID);
            Result = sync.wait();
            return Result;
        }

        //-----------< method to obtain a Cell clone for stateful operation >---
        public Clone getClone(string MessageName)
        {
            Guid g = Matrix.createClone(MessageName);
            if(g.CompareTo(Guid.Empty) == 0)
                throw new Exception("Could not locate cell to handle: "+
                    MessageName);
            else
                return new Clone(g);
        }

        //-----< check if a Cell ID is valid >-----------------------------------
        bool isValid(Clone clone)
        {
            Guid CellID = clone.guid();
            if(CellID.CompareTo(Guid.Empty)==0)
                return false;
            else
                return true;
        }

        //-----------<add response received flag to ResponseList >--------------
        public void addResponse(Guid RequestID)
        {
            ResponseList.Add(RequestID, null);
        }

        //-----------<remove response received flag to ResponseList >-----------
        public void removeResponse(Guid RequestID)
        {
            if(ResponseList.Contains(RequestID))
            {
                ResponseList.Remove(RequestID);
            }
        }

        //-----------< retrieve response >----------------------------------------
        public bool checkAndRetrieveResponse(Guid RequestID)
        {

            if(ResponseList.Contains(RequestID))
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        //-----< initialize cell >-----------------------------------------------
        private void initializeCell()
        {
            if(ID.CompareTo(Guid.Empty)==0)
            {
                ID = Guid.NewGuid(); //ID//
```

```
            }
            messager = new Messaging();
            queue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extract));
            ResponseQueue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extract));
            //Debug.WriteLine(ID);
            CapabilityList = new System.Collections.Hashtable();
            ResponseList = new System.Collections.Hashtable();
        }
        #endregion

        //-----< cell entry point >-------------------------------------------------
        public void start()
        {
            //TODO: add executive processing if required
            //default: do nothing
        }

        //-----------< constructor >-----------------
        public GUI()
        {
            //
            initializeCell();
            //TODO:
            //Add constructor logic here
            CapabilityList.Add("GUI.DrawBox", null);
            CapabilityList.Add("GUI.DrawTitleBox", null);
        }

        //-----------< calls to message-specific processing >-------------------
        public Object process(ExecObject eo)
        {
            ReturnObject ro = null;
            Exception exception = null;
            Object ret=null;
            try
            {   //
                //TODO:
                //add calls to processing logic here
                string MessageType = eo.getMessageType();
                switch(MessageType)
                {
                    case "GUI.DrawBox":
                        ret = process_GUI_DrawBox(eo);
                        break;
                    case "GUI.DrawTitleBox":
                        ret = process_GUI_DrawTitleBox(eo);
                        break;
                }
            }
            catch(Exception ex)
            {
                exception = ex;
            }
            ro = new ReturnObject(ret,exception);
            return ro;
        }


        //-----------< method to process "GUI.DrawBox" messages >---------------
        private Object process_GUI_DrawBox(ExecObject obj)
        {
            string Result = null;
            ConsoleBox b = new ConsoleBox();
            Object[] Params = obj.getParams();
```

```
            if(Params.Length==1)
                Result = b.drawBox((string)Params[0]);
            else if(Params.Length==3)
                Result = b.drawBox((string)Params[0],(int)Params[1],(int)Params[2]);
            else
                throw new Exception("Incorrect Input for GUI.DrawBox message processing")↙
    ;
            return Result;
        }

        //-----------< method to process "GUI.DrawBox" messages >---------------
        private Object process_GUI_DrawTitleBox(ExecObject obj)
        {
            string Result = null;
            ConsoleBox b = new ConsoleBox();
            Object[] Params = obj.getParams();
            if(Params.Length==1)
                Result = b.drawTitleBox((string)Params[0]);
            else if(Params.Length==3)
                Result = b.drawTitleBox((string)Params[0],(int)Params[1],(int)Params[2]);
            else
                throw new Exception("Incorrect Input for GUI.DrawBox message processing")↙
    ;
            return Result;
        }



        ////////////////////////////////////////////////////////////////////////
        ///                                                                  ///
        ///        Class to model a console message box                      ///
        ///                                                                  ///
        ////////////////////////////////////////////////////////////////////////
        private class ConsoleBox
        {
            private class Box
            {
                public char HorzLine = '-';//196
                public char VertLine = '¦';//179
                public char TopLeftCorner = '+';//218
                public char TopRightCorner = '+';//191
                public char BottomLeftCorner = '+';//192
                public char BottomRightCorner = '+';//217
            }

            private class TitleBox : Box
            {
                public  new char HorzLine = '-';//205
                public  new char VertLine = '¦';//186
                public  new char TopLeftCorner = '+';//201
                public  new char TopRightCorner = '+';//187
                public  new char BottomLeftCorner = '+';//200
                public  new char BottomRightCorner = '+';//188
            }

            //-----------< ConsoleBox constructor >-----------------------------
            public ConsoleBox()
            {
            }

            //-----------< overloaded method to draw a title box >--------------
            public string drawTitleBox(string str)
            {
                return drawBox(str, 70, 5, new TitleBox());
            }
```

```
        //-----------< overloaded method to draw a title box >--------------
        public string drawTitleBox(string str, int Width, int TextMargin)
        {
            TitleBox tb = new TitleBox();
            return drawBox( str,  Width,  TextMargin, tb );
        }

        //-----------< overloaded method to draw a message box >------------
        public string drawBox(string str)
        {
            return drawBox(str, 70, 5, new Box());
        }

        //-----------< overloaded method to draw a message box >------------
        public string drawBox(string Text, int Width, int TextMargin)
        {
            return drawBox( Text,  Width,  TextMargin, new Box() );
        }


        //-----------< overloaded method to draw a message box >------------
        private string drawBox(string Text, int Width, int TextMargin, Box box)
        {
            string Display = null;
            Display+="\n";
            Text = Text.Replace("\r","");
            Text = justifyLine(Text,Width-10);
            String[] Lines = Text.Split(new char[]{'\n'});
            Display+="  " + box.TopLeftCorner;
            for(int i=0; i<(Width-2); i++)
                Display+=box.HorzLine;
            Display+=box.TopRightCorner;
            Display+="\n";
            for(int i=0; i<Lines.Length; i++)
            {
                Display+=("  " + box.VertLine);
                for(int j=0; j<TextMargin-1; j++)
                    Display+=" ";
                Display+=(Lines[i].PadRight(Width-10));
                for(int j=0; j<TextMargin-1; j++)
                    Display+=" ";
                Display+= box.VertLine + "\n" ;
            }
            Display+=("  " + box.BottomLeftCorner);
            for(int i=0; i<(Width-2); i++)
                Display+=box.HorzLine;
            Display+=box.BottomRightCorner + "\n";
            return Display;
        }


        //-----------< helper function for text justification  >------------
        private string justifyLine(string str, int margin)
        {
            string line = null;
            if(str.Length <= margin)
            {
                return str;
            }
            //char[] Delim = { '.', ',', '-', ' ', ';', '=', ':', '"'};
            char[] Delim = { ' '};
            int index = str.LastIndexOfAny(Delim,margin-1,margin-1);
            if(index == -1)
                index=margin;
            line = str.Substring(0,index) + "\n";
            str = (str.Remove(0,index)).TrimStart();
```

```
                line+= justifyLine(str, margin);
                return line;
            }
        }
    }
}
```

```
/////////////////////////////////////////////////////////////////
//  Math.cs          File implementing a "Cell".               //
//                   Implements common math operations.        //
//                   Possesses the capability to handle the    //
//                   following Message Types:                  //
//                   "Math.Add"                                //
//  ver 1.0                                                    //
//                                                             //
//  Language:      C#                                         //
//  Platform:      Dell Dimension Pentium 4, Windows 2000     //
//  Course:        CSE997 Master's Thesis                     //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>          //
//                                                             //
/////////////////////////////////////////////////////////////////
using System;

namespace Matrix
{
    public class Math : Cell
    {
        private int stateful = 10;
        #region Wizard Generated Code
        private Guid ID;
        private Messaging messager;
        private BlockingQueue queue;
        private System.Threading.Thread StartThread;
        private System.Collections.Hashtable ResponseList;
        protected System.Collections.Hashtable CapabilityList;
        public event ReceiveDelegate OnReceive;
        private BlockingQueue ResponseQueue;

        //-----------< process messages pulled out of the queue >---------------
        public void extract(object msg)
        {
            ExecObject eo = messager.extract(msg);
            if(eo==null)
                return;
            if(eo.isRequest())
            {
                Object result = process(eo);
                Guid RequestCellGUID;
                String response = messager.buildResponse(msg,result,out RequestCellGUID);
                Matrix.sendResponse(response, RequestCellGUID); //SEND RESPONSE
            }
                //Response Received
            else
            {
                ReceiveInfoEventArgs rargs = new ReceiveInfoEventArgs(new Guid(eo.
    getRequestID()), eo.getReturnValue());
                if(OnReceive!=null)
                    OnReceive(this, rargs);
            }
        }


        //-----< process responses out of the response queue >-------------------------
        public void extractResponse(object msg)
        {
            ExecObject eo = messager.extract(msg);
            if(eo==null)
                return;
            if(!eo.isRequest())
            {
                ReceiveInfoEventArgs rargs = new ReceiveInfoEventArgs(new Guid(eo.
    getRequestID()), eo.getReturnValue());
                if(OnReceive!=null)
```

```
                    OnReceive(this, rargs);
                }
            }

            //-----------< method to register cell in matrix >----------------------
            public bool register()
            {
                Matrix.register((Cell)this);
                StartThread = new System.Threading.Thread(new System.Threading.ThreadStart(
        start));
                StartThread.Start();
                return true;
            }

            //-----------< method to unregister cell in matrix >--------------------
            public bool unregister()
            {
                Matrix.unregister(this);
                StartThread.Abort();
                return true;
            }

            //-----------< accept a request message >-------------------------------------
            public void accept(string message)
            {
                accept(message,false);
            }

            //-----< accept a response message >-----------------------------------------
            public void accept(string message, bool Response)
            {
                Object msg = message;
                if(Response == false)
                    queue.enQ(msg);
                else
                    ResponseQueue.enQ(msg);
            }


            //-----------< send message to another registered cell >----------------
            public bool send(string message, Guid ID, int argc, Object[] args)
            {
                return true;
            }

            //-----------< get unique cell ID >-------------------------------------
            public Guid getID()
            {
                return ID;
            }

            //-----------< capability interrogation >-------------------------------------
            public bool queryCapability(string capability)
            {
                if(CapabilityList.ContainsKey(capability))
                    return true;
                else
                    return false;
            }

            //-----------< method to send execution messages to other cells >-------
            public Object syncSend(string MessageName, Object[] Params)
            {
                string RequestCellGuid = ID.ToString();
                Object Result = null;
                Messaging msg = new Messaging();
```

```
        Guid RequestID;
        string xml = msg.buildRequest(RequestCellGuid,MessageName,
            Result, Params, out RequestID, Guid.Empty);
        Matrix.transmit(xml);
        SyncWait sync = new SyncWait(this, RequestID);
        Result = sync.wait();
        return Result;
    }

    //-----------< method to send execution messages to clone cells >-------
    public Object syncSend(string MessageName, Object[] Params, Clone clone)
    {
        Guid guid = clone.guid();
        string RequestCellGuid = ID.ToString();
        Object Result = null;
        Messaging msg = new Messaging();
        Guid RequestID;
        string xml = msg.buildRequest(RequestCellGuid,MessageName,
            Result, Params, out RequestID,guid);
        Matrix.transmit(xml);
        SyncWait sync = new SyncWait(this, RequestID);
        Result = sync.wait();
        return Result;
    }


    //-----------< method to obtain a Cell clone for stateful operation >---
    public Clone getClone(string MessageName)
    {
        Guid g = Matrix.createClone(MessageName);
        if(g.CompareTo(Guid.Empty) == 0)
            throw new Exception("Could not locate cell to handle: "+
                MessageName);
        else
            return new Clone(g);
    }

    //-----< check if a Cell ID is valid >-----------------------------------
    bool isValid(Clone clone)
    {
        Guid CellID = clone.guid();
        if(CellID.CompareTo(Guid.Empty)==0)
            return false;
        else
            return true;
    }

    //-----------<add response received flag to ResponseList >--------------
    public void addResponse(Guid RequestID)
    {
        ResponseList.Add(RequestID, null);
    }

    //-----------<remove response received flag to ResponseList >-----------
    public void removeResponse(Guid RequestID)
    {
        if(ResponseList.Contains(RequestID))
        {
            ResponseList.Remove(RequestID);
        }
    }

    //-----------< retrieve response >---------------------------------------
    public bool checkAndRetrieveResponse(Guid RequestID)
    {
```

```
            if(ResponseList.Contains(RequestID))
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        //-----< initialize cell >-------------------------------------------
        private void initializeCell()
        {
            if(ID.CompareTo(Guid.Empty)==0)
            {
                ID = Guid.NewGuid(); //ID//
            }
            messager = new Messaging();
            queue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extract));
            ResponseQueue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extractResponse));
            //Debug.WriteLine(ID);
            CapabilityList = new System.Collections.Hashtable();
            ResponseList = new System.Collections.Hashtable();
        }
        #endregion

        //-----< cell entry point >------------------------------------------------
        public void start()
        {
            //TODO: add executive processing if required
            //default: do nothing
        }

        //-----------< calls to message-specific processing >-------------------
        public Object process(ExecObject eo)
        {
            ReturnObject ro = null;
            Exception exception = null;
            Object ret=null;
            try
            {   //
                //TODO:
                //add calls to processing logic here
                string MessageType = eo.getMessageType();
                switch(MessageType)
                {
                    case "Math.Add":
                        ret = process_Math_Add(eo);
                        break;
                    case "Math.Modify":
                        ret = process_Math_Modify(eo);
                        break;
                }
            }
            catch(Exception ex)
            {
                exception = ex;
            }
            ro = new ReturnObject(ret,exception);
            return ro;
        }

        //-----------< constructor >-----------------------------------------
        public Math()
```

```
        {
            //
            initializeCell();
            //TODO:
            //Add constructor logic here
            CapabilityList.Add("Math.Add", null);
            CapabilityList.Add("Math.Modify", null);
        }

        //-----------< method to add 2 double values >-------------------------
        public static double add(double first, double second)
        {
            return (first+second);
        }

        //-----------< method to multiply 2 double values >--------------------
        public static double multiply(double first, double second)
        {
            return (first*second);
        }

        //-----< test stateful operation >-------------------------------------
        public double modify()
        {
            stateful+=5;
            return stateful;
        }

        //-----------< method to process "Modify" messages >-----------------------
        public Object process_Math_Modify(ExecObject obj)
        {
            //a test message/function
            Object[] Params = obj.getParams();
            if(Params.Length!=0)
                throw new Exception("Incorrect Input for Add message processing");
            else
            {
                Object o = modify();
                return o;
            }
        }

        //-----------< method to process "Add" messages >-----------------------
        public Object process_Math_Add(ExecObject obj)
        {
            Object[] Params = obj.getParams();
            if(Params.Length!=2)
                throw new Exception("Incorrect Input for Add message processing");
            else
            {
                Object o = add((double)Params[0],(double)Params[1]);
                return o;
            }
        }
    }

}
```

NetView Matrix Application

```
//////////////////////////////////////////////////////////////////
// FileTransferPeer.cs File implementing a "Cell".              //
//                     Possesses the capability to handle the   //
//                     following Message Types:                 //
//                      "SU.FileTransfer.GetLocalDirsFiles"     //
//                      "SU.FileTransfer.RunServer"             //
//                      "SU.FileTransfer.Connect"               //
//                      "SU.FileTransfer.GetRemoteDirsFiles"    //
//                      "SU.FileTransfer.UploadLocalFile"       //
//                      "SU.FileTransfer.DownloadRemoteFile"    //
//                                                              //
//  ver 1.0                                                     //
//                                                              //
//  Language:      C#                                          //
//  Platform:      Dell Dimension Pentium 4, Windows 2000      //
//  Course:        CSE997 Master's Thesis                      //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>           //
//                                                              //
//////////////////////////////////////////////////////////////////
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
using System.IO;
using Matrix;


namespace FileTransferPeer
{
    ////////////////////////////////////
    ///// file transfer/synchronizer peer
    ////////////////////////////////////
    class FileTransferPeer : Cell
    {
        private FileTransfer.FileTransfer RemoteFileTransfer;
        private FileTransfer.FileSystemTraversal LocalFSTraversal;
        private FileTransfer.FileTransfer LocalFileTransfer;
        private FileTransfer.FileSystemTraversal RemoteFSTraversal;
        #region Wizard Generated Code
        private Guid ID;
        private Messaging messager;
        private BlockingQueue queue;
        private BlockingQueue ResponseQueue;
        private System.Threading.Thread StartThread;
        private System.Collections.Hashtable ResponseList;
        protected System.Collections.Hashtable CapabilityList;
        public event ReceiveDelegate OnReceive;

        //-----------< process messages pulled out of the queue >---------------
        public void extract(object msg)
        {
            ExecObject eo = messager.extract(msg);
            if(eo==null)
                return;
            if(eo.isRequest())
            {
                Object result = process(eo);
                Guid RequestCellGUID;
                String response = messager.buildResponse(msg,result,out RequestCellGUID);
                Matrix.Matrix.send(response, RequestCellGUID);
            }
                //Response Received
            else
            {
                ReceiveInfoEventArgs rargs = new ReceiveInfoEventArgs(new Guid(eo.
    getRequestID()), eo.getReturnValue());
```

```
                if(OnReceive!=null)
                    OnReceive(this, rargs);
            }
        }

        //-----------< method to register cell in matrix >----------------------
        public bool register()
        {
            Matrix.Matrix.register((Cell)this);
            StartThread = new System.Threading.Thread(new System.Threading.ThreadStart(
    start));
            StartThread.Start();
            return true;
        }

        //-----------< method to unregister cell in matrix >--------------------
        public bool unregister()
        {
            Matrix.Matrix.unregister(this);
            StartThread.Abort();
            return true;
        }

        //-----------< accept a request message >-------------------------------------
        public void accept(string message)
        {
            accept(message,false);
        }

        //-----< accept a response message >-----------------------------------------
        public void accept(string message, bool Response)
        {
            Object msg = message;
            if(Response == false)
                queue.enQ(msg);
            else
                ResponseQueue.enQ(msg);
        }


        //-----------< send message to another registered cell >----------------
        public bool send(string message, Guid ID, int argc, Object[] args)
        {
            return true;
        }

        //-----------< get unique cell ID >-------------------------------------
        public Guid getID()
        {
            return ID;
        }

        //-----------< capability interrogation >-------------------------------------
        public bool queryCapability(string capability)
        {
            if(CapabilityList.ContainsKey(capability))
                return true;
            else
                return false;
        }

        //-----------< method to send execution messages to other cells >-------
        public Object syncSend(string MessageName, Object[] Params)
        {
            string RequestCellGuid = ID.ToString();
            Object Result = null;
```

```
        Messaging msg = new Messaging();
        Guid RequestID;
        string xml = msg.buildRequest(RequestCellGuid,MessageName,
            Result, Params, out RequestID, Guid.Empty);
        Matrix.Matrix.transmit(xml);
        SyncWait sync = new SyncWait(this, RequestID);
        Result = sync.wait();
        return Result;
    }

    //-----------< method to send execution messages to clone cells >-------
    public Object syncSend(string MessageName, Object[] Params, Clone clone)
    {
        Guid guid = clone.guid();
        string RequestCellGuid = ID.ToString();
        Object Result = null;
        Messaging msg = new Messaging();
        Guid RequestID;
        string xml = msg.buildRequest(RequestCellGuid,MessageName,
            Result, Params, out RequestID,guid);
        Matrix.Matrix.transmit(xml);
        SyncWait sync = new SyncWait(this, RequestID);
        Result = sync.wait();
        return Result;
    }

    //-----------< method to obtain a Cell clone for stateful operation >---
    public Clone getClone(string MessageName)
    {
        Guid g = Matrix.Matrix.createClone(MessageName);
        if(g.CompareTo(Guid.Empty) == 0)
            throw new Exception("Could not locate cell to handle: "+
                MessageName);
        else
            return new Clone(g);
    }

    //-----< check if a Cell ID is valid >-----------------------------------
    bool isValid(Clone clone)
    {
        Guid CellID = clone.guid();
        if(CellID.CompareTo(Guid.Empty)==0)
            return false;
        else
            return true;
    }

    //-----------<add response received flag to ResponseList >--------------
    public void addResponse(Guid RequestID)
    {
        ResponseList.Add(RequestID, null);
    }

    //-----------<remove response received flag to ResponseList >-----------
    public void removeResponse(Guid RequestID)
    {
        if(ResponseList.Contains(RequestID))
        {
            ResponseList.Remove(RequestID);
        }
    }

    //-----------< retrieve response >---------------------------------------
    public bool checkAndRetrieveResponse(Guid RequestID)
    {
```

```
            if(ResponseList.Contains(RequestID))
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        //-----< initialize cell >-------------------------------------------------
        private void initializeCell()
        {
            if(ID.CompareTo(Guid.Empty)==0)
            {
                ID = Guid.NewGuid(); //ID//
            }
            messager = new Messaging();
            queue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extract));
            ResponseQueue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extract));
            CapabilityList = new System.Collections.Hashtable();
            ResponseList = new System.Collections.Hashtable();
        }
        #endregion

        //-----< cell entry point >------------------------------------------------
        public void start()
        {
            //TODO: add executive processing if required
            //default: do nothing
        }
        //-----------< calls to message-specific processing >--------------------
        public Object process(ExecObject eo)
        {
            ReturnObject ro = null;
            Exception exception = null;
            Object ret=null;
            try
            {   //
                //TODO:
                //add calls to processing logic here
                string MessageType = eo.getMessageType();
                switch(MessageType)
                {
                    case "SU.FileTransfer.GetLocalDirsFiles":
                        ret = process_FileTransfer_GetLocalDirsFiles(eo);
                        break;
                    case "SU.FileTransfer.GetRemoteDirsFiles":
                        ret = process_FileTransfer_GetRemoteDirsFiles(eo);
                        break;
                    case "SU.FileTransfer.RunServer":
                        ret = process_FileTransfer_RunServer(eo);
                        break;
                    case "SU.FileTransfer.Connect":
                        ret = process_FileTransfer_Connect(eo);
                        break;
                    case "SU.FileTransfer.UploadLocalFile":
                        ret = process_FileTransfer_UploadLocalFile(eo);
                        break;
                    case "SU.FileTransfer.DownloadRemoteFile":
                        ret = process_FileTransfer_DownloadRemoteFile(eo);
                        break;
                }
            }
```

```
            catch(Exception ex)
            {
                exception = ex;
            }
            ro = new ReturnObject(ret,exception);
            return ro;
        }


        //-----------< method to process GetLocalFiles messages >-----------------------
        public Object process_FileTransfer_GetLocalDirsFiles(ExecObject obj)
        {
            Object[] Params = obj.getParams();
            if(Params.Length>2)
            {
                throw new Exception("Incorrect Input for GetLocalDirsFiles message processing");
            }
            else if(Params.Length == 0)
            {
                Object Nodes = LocalFSTraversal.getDirsFiles();
                return Nodes;
            }
            else
            {
                Object Nodes = LocalFSTraversal.getDirsFiles((string)Params[0]);
                return Nodes;
            }
        }

        //-----------< method to process GetRemoteFiles messages >-----------------------
        public Object process_FileTransfer_GetRemoteDirsFiles(ExecObject obj)
        {
            Object[] Params = obj.getParams();
            if(Params.Length>2)
            {
                throw new Exception("Incorrect Input for GetRemoteDirsFiles message processing");
            }
            else if(Params.Length == 0)
            {
                Object Nodes = RemoteFSTraversal.getDirsFiles();
                return Nodes;
            }
            else
            {
                Object Nodes = RemoteFSTraversal.getDirsFiles((string)Params[0]);
                return Nodes;
            }
        }

        //-----------< method to process Connect messages >-----------------------
        public Object process_FileTransfer_Connect(ExecObject obj)
        {
            Object[] Params = obj.getParams();
            if(Params.Length!=2)
            {
                throw new Exception("Incorrect Input for Connect message processing");
            }
            bool success = connect((string)Params[0],(int)Params[1]);
            return success;
        }

        //-----------< method to process RunServer messages >-----------------------
        public Object process_FileTransfer_RunServer(ExecObject obj)
        {
```

```
            bool success = false;
            Object[] Params = obj.getParams();
            if(Params.Length!=1)
            {
                throw new Exception("Incorrect Input for RunServer message processing");
            }
            else
            {
                int ListeningPort = (int)Params[0];
                success = initServer(ListeningPort);
            }
            return success;
        }

        //-----------< method to process UploadLocalFile messages >----------------------
        public Object process_FileTransfer_UploadLocalFile(ExecObject obj)
        {
            Object[] Params = obj.getParams();
            if(Params.Length!=2)
            {
                throw new Exception("Incorrect Input for UploadLocalFile message processing");
            }
            bool success = sendLocalFile((string)Params[0],(string)Params[1]);
            return success;
        }

        //-----------< method to process DownloadRemoteFile messages >-------------------
        public Object process_FileTransfer_DownloadRemoteFile(ExecObject obj)
        {
            Object[] Params = obj.getParams();
            if(Params.Length!=2)
            {
                throw new Exception("Incorrect Input for DownloadRemoteFile message processing");
            }
            bool success = getRemoteFile((string)Params[0],(string)Params[1]);
            return success;
        }

        //-----------< constructor >-----------------------------------------
        public FileTransferPeer()
        {
            //
            initializeCell();
            //TODO:
            //Add constructor logic here
            LocalFSTraversal = new FileTransfer.FileSystemTraversal();
            LocalFileTransfer = new FileTransfer.FileTransfer();
            CapabilityList.Add("SU.FileTransfer.GetLocalDirsFiles",null);
            CapabilityList.Add("SU.FileTransfer.RunServer",null);
            CapabilityList.Add("SU.FileTransfer.Connect",null);
            CapabilityList.Add("SU.FileTransfer.GetRemoteDirsFiles",null);
            CapabilityList.Add("SU.FileTransfer.UploadLocalFile",null);
            CapabilityList.Add("SU.FileTransfer.DownloadRemoteFile",null);
        }


        //-----< init client >-------------------------------------------------
        public bool initClient(string uri)
        {
            BinaryClientFormatterSinkProvider cp = new BinaryClientFormatterSinkProvider();
            BinaryServerFormatterSinkProvider sp = new BinaryServerFormatterSinkProvider
```

```
();
        sp.TypeFilterLevel = System.Runtime.Serialization.Formatters.TypeFilterLevel.
Full;
        System.Collections.IDictionary props = new System.Collections.Hashtable();
        props["typeFilterLevel"] = System.Runtime.Serialization.Formatters.
TypeFilterLevel.Full;
        props["name"] = "client";
        TcpChannel ClientChannel = new TcpChannel(props,cp,sp);
        ChannelServices.RegisterChannel(ClientChannel);
        return true;
    }

    //-----< initialize server >-----------------------------------------
    public bool initServer(int port)
    {
        BinaryClientFormatterSinkProvider cp = new BinaryClientFormatterSinkProvider
();
        BinaryServerFormatterSinkProvider sp = new BinaryServerFormatterSinkProvider
();
        sp.TypeFilterLevel = System.Runtime.Serialization.Formatters.TypeFilterLevel.
Full;
        System.Collections.IDictionary props = new System.Collections.Hashtable();
        props["typeFilterLevel"] = System.Runtime.Serialization.Formatters.
TypeFilterLevel.Full;
        props["port"] = port;
        props["name"] = "server";
        TcpChannel ServerChannel = new TcpChannel(props,cp,sp);
        ChannelServices.RegisterChannel(ServerChannel);
        RemotingConfiguration.RegisterWellKnownServiceType(typeof(FileTransfer.
FileTransfer),
            "FileTransfer",WellKnownObjectMode.SingleCall);
        RemotingConfiguration.RegisterWellKnownServiceType(typeof(FileTransfer.
FileSystemTraversal),
            "Synchronizer",WellKnownObjectMode.SingleCall);
        return true;
    }

    //-----< connect to remote file server/peer >----------------------------
    public bool connect(string RemoteMachine, int RemotePort)
    {

        bool success = initClient("tcp://" + RemoteMachine + ":" + RemotePort + "/
FileTransfer");
        //remote instance
        this.RemoteFileTransfer = (FileTransfer.FileTransfer)Activator.GetObject(
typeof(FileTransfer.FileTransfer),
            "tcp://" + RemoteMachine + ":" + RemotePort + "/FileTransfer");
        this.RemoteFSTraversal = (FileTransfer.FileSystemTraversal)Activator.
GetObject(typeof(FileTransfer.FileSystemTraversal),
                            "tcp://" + RemoteMachine + ":" + RemotePort + "/
Synchronizer");
        return success;
    }

    //-----< send a local file to the remotely connected machine >----------
    public bool sendLocalFile(string SourcePath_FileName,
        string DestinationPath_FileName)
    {
        FileStream fs = new FileStream(SourcePath_FileName,FileMode.Open);
        BinaryReader br = new BinaryReader(fs);
        byte[] buffer = br.ReadBytes((int)fs.Length);
        fs.Close(); br.Close();
        return RemoteFileTransfer.writeFile(buffer,DestinationPath_FileName);
    }

    //-----< get a file from the remotely connected machine >--------------
```

```
        public bool getRemoteFile(string Local_FileName,
            string Remote_FileName)
        {
            byte[] buffer = RemoteFileTransfer.readFile(Remote_FileName);
            FileStream fs = new FileStream(Local_FileName,FileMode.Create);
            BinaryWriter bw = new BinaryWriter(fs);
            bw.Write(buffer);
            bw.Close(); fs.Close();
            return true;
        }
    }
}
```

```
/////////////////////////////////////////////////////////////////////
//  TextSerializer.cs File implementing a "Cell" that provides     //
//                   serialization and deserialization             //
//                   capabilities. Its capability list             //
//                   contains the following Message Types:         //
//                   "SU.TextSerializer.Serialize"                 //
//                   "SU.TextSerializer.Deserialize"               //
//                                                                 //
//  ver 1.0                                                        //
//                                                                 //
//  Language:      C#                                              //
//  Platform:      Dell Dimension Pentium 4, Windows 2000          //
//  Course:        CSE997 Master's Thesis                          //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>               //
//                                                                 //
/////////////////////////////////////////////////////////////////////
using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;


namespace TextSerializer
{
    using Matrix;

    public class TextSerializer : Cell
    {

        #region Wizard Generated Code
        private Guid ID;
        private Messaging messager;
        private BlockingQueue queue;
        private System.Threading.Thread StartThread;
        private System.Collections.Hashtable ResponseList;
        protected System.Collections.Hashtable CapabilityList;
        public event ReceiveDelegate OnReceive;
        private BlockingQueue ResponseQueue;


        //-----------< process messages pulled out of the queue >---------------
        public void extract(object msg)
        {
            ExecObject eo = messager.extract(msg);
            if(eo==null)
                return;
            if(eo.isRequest())
            {
                Object result = process(eo);
                Guid RequestCellGUID;
                String response = messager.buildResponse(msg,result,out RequestCellGUID);
                Matrix.sendResponse(response, RequestCellGUID); //SEND RESPONSE
            }
                //Response Received
            else
            {
                ReceiveInfoEventArgs rargs = new ReceiveInfoEventArgs(new Guid(eo.
    getRequestID()), eo.getReturnValue());
                if(OnReceive!=null)
                    OnReceive(this, rargs);
            }
        }


        //-----< process responses out of the response queue >-----------------------
        public void extractResponse(object msg)
```

```
        {
            ExecObject eo = messager.extract(msg);
            if(eo==null)
                return;
            if(!eo.isRequest())
            {
                ReceiveInfoEventArgs rargs = new ReceiveInfoEventArgs(new Guid(eo.
    getRequestID()), eo.getReturnValue());
                if(OnReceive!=null)
                    OnReceive(this, rargs);
            }
        }

        //-----------< method to register cell in matrix >----------------------
        public bool register()
        {
            Matrix.register((Cell)this);
            StartThread = new System.Threading.Thread(new System.Threading.ThreadStart(
    start));
            StartThread.Start();
            return true;
        }

        //-----------< method to unregister cell in matrix >--------------------
        public bool unregister()
        {
            Matrix.unregister(this);
            StartThread.Abort();
            return true;
        }

        //-----------< accept a request message >-------------------------------
        public void accept(string message)
        {
            accept(message,false);
        }

        //-----< accept a response message >-----------------------------------
        public void accept(string message, bool Response)
        {
            Object msg = message;
            if(Response == false)
                queue.enQ(msg);
            else
                ResponseQueue.enQ(msg);
        }


        //-----------< send message to another registered cell >----------------
        public bool send(string message, Guid ID, int argc, Object[] args)
        {
            return true;
        }

        //-----------< get unique cell ID >------------------------------------
        public Guid getID()
        {
            return ID;
        }

        //-----------< capability interrogation >------------------------------
        public bool queryCapability(string capability)
        {
            if(CapabilityList.ContainsKey(capability))
                return true;
            else
```

```
            return false;
        }

        //-----------< method to send execution messages to other cells >-------
        public Object syncSend(string MessageName, Object[] Params)
        {
            string RequestCellGuid = ID.ToString();
            Object Result = null;
            Messaging msg = new Messaging();
            Guid RequestID;
            string xml = msg.buildRequest(RequestCellGuid,MessageName,
                Result, Params, out RequestID, Guid.Empty);
            Matrix.transmit(xml);
            SyncWait sync = new SyncWait(this, RequestID);
            Result = sync.wait();
            return Result;
        }

        //-----------< method to send execution messages to clone cells >-------
        public Object syncSend(string MessageName, Object[] Params, Clone clone)
        {
            Guid guid = clone.guid();
            string RequestCellGuid = ID.ToString();
            Object Result = null;
            Messaging msg = new Messaging();
            Guid RequestID;
            string xml = msg.buildRequest(RequestCellGuid,MessageName,
                Result, Params, out RequestID,guid);
            Matrix.transmit(xml);
            SyncWait sync = new SyncWait(this, RequestID);
            Result = sync.wait();
            return Result;
        }


        //-----------< method to obtain a Cell clone for stateful operation >---
        public Clone getClone(string MessageName)
        {
            Guid g = Matrix.createClone(MessageName);
            if(g.CompareTo(Guid.Empty) == 0)
                throw new Exception("Could not locate cell to handle: "+
                    MessageName);
            else
                return new Clone(g);
        }

        //-----< check if a Cell ID is valid >-----------------------------------
        bool isValid(Clone clone)
        {
            Guid CellID = clone.guid();
            if(CellID.CompareTo(Guid.Empty)==0)
                return false;
            else
                return true;
        }

        //-----------<add response received flag to ResponseList >--------------
        public void addResponse(Guid RequestID)
        {
            ResponseList.Add(RequestID, null);
        }

        //-----------<remove response received flag to ResponseList >-----------
        public void removeResponse(Guid RequestID)
        {
            if(ResponseList.Contains(RequestID))
```

```
            {
                ResponseList.Remove(RequestID);
            }
        }

        //-----------< retrieve response >-------------------------------------
        public bool checkAndRetrieveResponse(Guid RequestID)
        {

            if(ResponseList.Contains(RequestID))
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        //-----< initialize cell >---------------------------------------------
        private void initializeCell()
        {
            if(ID.CompareTo(Guid.Empty)==0)
            {
                ID = Guid.NewGuid(); //ID//
            }
            messager = new Messaging();
            queue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extract));
            ResponseQueue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extractResponse));
            //Debug.WriteLine(ID);
            CapabilityList = new System.Collections.Hashtable();
            ResponseList = new System.Collections.Hashtable();
        }
        #endregion

        //-----< cell entry point >--------------------------------------------------
        public void start()
        {
            //TODO: add executive processing if required
            //default: do nothing
        }

        //-----------< calls to message-specific processing >------------------
        public Object process(ExecObject eo)
        {
            ReturnObject ro = null;
            Exception exception = null;
            Object ret=null;
            try
            {   //
                //TODO:
                //add calls to processing logic here
                string MessageType = eo.getMessageType();
                switch(MessageType)
                {
                    case "SU.TextSerializer.Serialize":
                        ret = process_TextSerializer_Serialize(eo);
                        break;
                    case "SU.TextSerializer.Deserialize":
                        ret = process_TextSerializer_Deserialize(eo);
                        break;
                }
            }
            catch(Exception ex)
```

```
            {
                exception = ex;
            }
            ro = new ReturnObject(ret,exception);
            return ro;
        }

        //-----------< constructor >-----------------------------------------
        public TextSerializer()
        {
            //
            initializeCell();
            //TODO:
            //Add constructor logic here
            CapabilityList.Add("SU.TextSerializer.Serialize", null);
            CapabilityList.Add("SU.TextSerializer.Deserialize", null);
        }

        //-----------< method to process "SU.TextSerializer.Serialize" messages >---
        public Object process_TextSerializer_Serialize(ExecObject obj)
        {
            Object[] Params = obj.getParams();
            if(Params.Length!=1)
                throw new Exception("Incorrect Input for Add message processing");
            else
            {
                Object o = textSerialize(Params[0]);
                return o;
            }
        }


        //-----------< method to process "SU.TextSerializer.Deserialize" messages >-
        public Object process_TextSerializer_Deserialize(ExecObject obj)
        {
            Object[] Params = obj.getParams();
            if(Params.Length!=1)
                throw new Exception("Incorrect Input for Add message processing");
            else
            {
                Object o = textDeserialize((string)Params[0]);
                return o;
            }
        }


        //-----< method to serialize an object in text (base64) format >--------

        public string textSerialize(Object obj)
        {
            if(obj == null)
                return "";
            MemoryStream ms = new MemoryStream();
            BinaryFormatter b = new BinaryFormatter();
            b.Serialize(ms,obj);
            byte[] buffer = ms.ToArray();
            return Convert.ToBase64String(buffer);
        }


        //----------< method to deserialize a base64 encoded text string >------
        public Object textDeserialize(string ser)
        {
            if(ser.Equals(""))
                return null;
            byte[] buffer = Convert.FromBase64String(ser);
```

```
            MemoryStream ms = new MemoryStream(buffer);
            BinaryFormatter b = new BinaryFormatter();
            return (b.Deserialize(ms));
        }
    }
}
```

```
/////////////////////////////////////////////////////////////////////
//  TextSerializer.cs File implementing a "Cell" that provides     //
//                   serialization and deserialization             //
//                   capabilities. Its capability list             //
//                   contains the following Message Types:         //
//                   "SU.TextSerializer.Serialize"                 //
//                   "SU.TextSerializer.Deserialize"               //
//                                                                 //
//  ver 1.0                                                        //
//                                                                 //
//  Language:      C#                                              //
//  Platform:      Dell Dimension Pentium 4, Windows 2000          //
//  Course:        CSE997 Master's Thesis                          //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>               //
//                                                                 //
/////////////////////////////////////////////////////////////////////
using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;


namespace TextSerializer
{
    using Matrix;

    public class TextSerializer : Cell
    {

        #region Wizard Generated Code
        private Guid ID;
        private Messaging messager;
        private BlockingQueue queue;
        private System.Threading.Thread StartThread;
        private System.Collections.Hashtable ResponseList;
        protected System.Collections.Hashtable CapabilityList;
        public event ReceiveDelegate OnReceive;
        private BlockingQueue ResponseQueue;


        //-----------< process messages pulled out of the queue >---------------
        public void extract(object msg)
        {
            ExecObject eo = messager.extract(msg);
            if(eo==null)
                return;
            if(eo.isRequest())
            {
                Object result = process(eo);
                Guid RequestCellGUID;
                String response = messager.buildResponse(msg,result,out RequestCellGUID);
                Matrix.sendResponse(response, RequestCellGUID); //SEND RESPONSE
            }
                //Response Received
            else
            {
                ReceiveInfoEventArgs rargs = new ReceiveInfoEventArgs(new Guid(eo.
    getRequestID()), eo.getReturnValue());
                if(OnReceive!=null)
                    OnReceive(this, rargs);
            }
        }


        //-----< process responses out of the response queue >-----------------------
        public void extractResponse(object msg)
```

```
        {
            ExecObject eo = messager.extract(msg);
            if(eo==null)
                return;
            if(!eo.isRequest())
            {
                ReceiveInfoEventArgs rargs = new ReceiveInfoEventArgs(new Guid(eo.
    getRequestID()), eo.getReturnValue());
                if(OnReceive!=null)
                    OnReceive(this, rargs);
            }
        }

        //-----------< method to register cell in matrix >----------------------
        public bool register()
        {
            Matrix.register((Cell)this);
            StartThread = new System.Threading.Thread(new System.Threading.ThreadStart(
    start));
            StartThread.Start();
            return true;
        }

        //-----------< method to unregister cell in matrix >--------------------
        public bool unregister()
        {
            Matrix.unregister(this);
            StartThread.Abort();
            return true;
        }

        //-----------< accept a request message >-------------------------------
        public void accept(string message)
        {
            accept(message,false);
        }

        //-----< accept a response message >-----------------------------------
        public void accept(string message, bool Response)
        {
            Object msg = message;
            if(Response == false)
                queue.enQ(msg);
            else
                ResponseQueue.enQ(msg);
        }


        //-----------< send message to another registered cell >----------------
        public bool send(string message, Guid ID, int argc, Object[] args)
        {
            return true;
        }

        //-----------< get unique cell ID >------------------------------------
        public Guid getID()
        {
            return ID;
        }

        //-----------< capability interrogation >------------------------------
        public bool queryCapability(string capability)
        {
            if(CapabilityList.ContainsKey(capability))
                return true;
            else
```

```
            return false;
        }

        //-----------< method to send execution messages to other cells >-------
        public Object syncSend(string MessageName, Object[] Params)
        {
            string RequestCellGuid = ID.ToString();
            Object Result = null;
            Messaging msg = new Messaging();
            Guid RequestID;
            string xml = msg.buildRequest(RequestCellGuid,MessageName,
                Result, Params, out RequestID, Guid.Empty);
            Matrix.transmit(xml);
            SyncWait sync = new SyncWait(this, RequestID);
            Result = sync.wait();
            return Result;
        }

        //-----------< method to send execution messages to clone cells >-------
        public Object syncSend(string MessageName, Object[] Params, Clone clone)
        {
            Guid guid = clone.guid();
            string RequestCellGuid = ID.ToString();
            Object Result = null;
            Messaging msg = new Messaging();
            Guid RequestID;
            string xml = msg.buildRequest(RequestCellGuid,MessageName,
                Result, Params, out RequestID,guid);
            Matrix.transmit(xml);
            SyncWait sync = new SyncWait(this, RequestID);
            Result = sync.wait();
            return Result;
        }


        //-----------< method to obtain a Cell clone for stateful operation >---
        public Clone getClone(string MessageName)
        {
            Guid g = Matrix.createClone(MessageName);
            if(g.CompareTo(Guid.Empty) == 0)
                throw new Exception("Could not locate cell to handle: "+
                    MessageName);
            else
                return new Clone(g);
        }

        //-----< check if a Cell ID is valid >-----------------------------------
        bool isValid(Clone clone)
        {
            Guid CellID = clone.guid();
            if(CellID.CompareTo(Guid.Empty)==0)
                return false;
            else
                return true;
        }

        //-----------<add response received flag to ResponseList >--------------
        public void addResponse(Guid RequestID)
        {
            ResponseList.Add(RequestID, null);
        }

        //-----------<remove response received flag to ResponseList >-----------
        public void removeResponse(Guid RequestID)
        {
            if(ResponseList.Contains(RequestID))
```

```
            {
                ResponseList.Remove(RequestID);
            }
        }

        //-----------< retrieve response >-------------------------------------
        public bool checkAndRetrieveResponse(Guid RequestID)
        {

            if(ResponseList.Contains(RequestID))
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        //-----< initialize cell >---------------------------------------------
        private void initializeCell()
        {
            if(ID.CompareTo(Guid.Empty)==0)
            {
                ID = Guid.NewGuid(); //ID//
            }
            messager = new Messaging();
            queue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extract));
            ResponseQueue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extractResponse));
            //Debug.WriteLine(ID);
            CapabilityList = new System.Collections.Hashtable();
            ResponseList = new System.Collections.Hashtable();
        }
        #endregion

        //-----< cell entry point >--------------------------------------------------
        public void start()
        {
            //TODO: add executive processing if required
            //default: do nothing
        }

        //-----------< calls to message-specific processing >------------------
        public Object process(ExecObject eo)
        {
            ReturnObject ro = null;
            Exception exception = null;
            Object ret=null;
            try
            {   //
                //TODO:
                //add calls to processing logic here
                string MessageType = eo.getMessageType();
                switch(MessageType)
                {
                    case "SU.TextSerializer.Serialize":
                        ret = process_TextSerializer_Serialize(eo);
                        break;
                    case "SU.TextSerializer.Deserialize":
                        ret = process_TextSerializer_Deserialize(eo);
                        break;
                }
            }
            catch(Exception ex)
```

```
            {
                exception = ex;
            }
            ro = new ReturnObject(ret,exception);
            return ro;
        }

        //-----------< constructor >-----------------------------------------
        public TextSerializer()
        {
            //
            initializeCell();
            //TODO:
            //Add constructor logic here
            CapabilityList.Add("SU.TextSerializer.Serialize", null);
            CapabilityList.Add("SU.TextSerializer.Deserialize", null);
        }

        //-----------< method to process "SU.TextSerializer.Serialize" messages >---
        public Object process_TextSerializer_Serialize(ExecObject obj)
        {
            Object[] Params = obj.getParams();
            if(Params.Length!=1)
                throw new Exception("Incorrect Input for Add message processing");
            else
            {
                Object o = textSerialize(Params[0]);
                return o;
            }
        }


        //-----------< method to process "SU.TextSerializer.Deserialize" messages >-
        public Object process_TextSerializer_Deserialize(ExecObject obj)
        {
            Object[] Params = obj.getParams();
            if(Params.Length!=1)
                throw new Exception("Incorrect Input for Add message processing");
            else
            {
                Object o = textDeserialize((string)Params[0]);
                return o;
            }
        }


        //-----< method to serialize an object in text (base64) format >--------

        public string textSerialize(Object obj)
        {
            if(obj == null)
                return "";
            MemoryStream ms = new MemoryStream();
            BinaryFormatter b = new BinaryFormatter();
            b.Serialize(ms,obj);
            byte[] buffer = ms.ToArray();
            return Convert.ToBase64String(buffer);
        }


        //----------< method to deserialize a base64 encoded text string >------
        public Object textDeserialize(string ser)
        {
            if(ser.Equals(""))
                return null;
            byte[] buffer = Convert.FromBase64String(ser);
```

```
            MemoryStream ms = new MemoryStream(buffer);
            BinaryFormatter b = new BinaryFormatter();
            return (b.Deserialize(ms));
        }
    }
}
```

```
/////////////////////////////////////////////////////////////////////
//  NetViewForm.cs Implements a Cell responsible for the       //
//                 graphical user interface of the NetView     //
//                 application.                                //
//                 Has an empty capability list.               //
//                                                             //
//  ver 1.0                                                    //
//                                                             //
//  Language:      C#                                          //
//  Platform:      Dell Dimension Pentium 4, Windows 2000      //
//  Course:        CSE997 Master's Thesis                      //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>           //
//                                                             //
/////////////////////////////////////////////////////////////////////

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.IO;
using System.Diagnostics;
using System.Runtime.Remoting;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using Matrix;

namespace NetView
{
    public class NetViewForm : System.Windows.Forms.Form, Cell
    {
        #region User Interface Code
        private delegate void DisplayDelegate(string path);
        private System.ComponentModel.Container components = null;
        private string StorePath = @"..\..\SlideStore\";
        private ArrayList Slides = new ArrayList();
        private Clone ftclone;
        private CircularPointer cp;
        private System.Windows.Forms.PictureBox SlideWindow;
        private System.Windows.Forms.TextBox RemotePortTextBox;
        private System.Windows.Forms.TextBox RemoteMachineTextBox;
        private Display RemoteDisplay;
        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Button AddSlidesButton;
        private System.Windows.Forms.OpenFileDialog AddSlidesFileDialog;
        private System.Windows.Forms.Button FirstButton;
        private System.Windows.Forms.Button BackButton;
        private System.Windows.Forms.Button ForwardButton;
        private System.Windows.Forms.Button LastButton;
        private System.Windows.Forms.Button ConnectButton;
        private System.Windows.Forms.Button ExitButton;
        private System.Windows.Forms.GroupBox groupBox2;
        private System.Windows.Forms.TextBox ServerPortTextBox;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Button ListenButton;
        private System.Windows.Forms.PictureBox pictureBox1;
        private int ServerPort=0;
        private bool Connected = false;
        #endregion

        #region Wizard Generated Code
        private Guid ID;
        private Messaging messager;
```

```
        private BlockingQueue queue;
        private System.Threading.Thread StartThread;
        private System.Collections.Hashtable ResponseList;
        protected System.Collections.Hashtable CapabilityList;
        public event ReceiveDelegate OnReceive;
        private BlockingQueue ResponseQueue;

        //-----------< process messages pulled out of the queue >---------------
        public void extract(object msg)
        {
            ExecObject eo = messager.extract(msg);
            if(eo==null)
                return;
            if(eo.isRequest())
            {
                Object result = process(eo);
                Guid RequestCellGUID;
                String response = messager.buildResponse(msg,result,out RequestCellGUID);
                Matrix.Matrix.sendResponse(response, RequestCellGUID); //SEND RESPONSE
            }
                //Response Received
            else
            {
                ReceiveInfoEventArgs rargs = new ReceiveInfoEventArgs(new Guid(eo.
    getRequestID()), eo.getReturnValue());
                if(OnReceive!=null)
                    OnReceive(this, rargs);
            }
        }


        //-----< process responses out of the response queue >-------------------------
        public void extractResponse(object msg)
        {
            ExecObject eo = messager.extract(msg);
            if(eo==null)
                return;
            if(!eo.isRequest())
            {
                ReceiveInfoEventArgs rargs = new ReceiveInfoEventArgs(new Guid(eo.
    getRequestID()), eo.getReturnValue());
                if(OnReceive!=null)
                    OnReceive(this, rargs);
            }
        }

        //-----------< method to register cell in matrix >----------------------
        public bool register()
        {
            Matrix.Matrix.register((Cell)this);
            StartThread = new System.Threading.Thread(new System.Threading.ThreadStart(
    start));
            StartThread.Start();
            return true;
        }

        //-----------< method to unregister cell in matrix >--------------------
        public bool unregister()
        {
            Matrix.Matrix.unregister(this);
            StartThread.Abort();
            return true;
        }

        //-----------< accept a request message >-------------------------------------
        public void accept(string message)
```

```
        {
            accept(message,false);
        }

        //-----< accept a response message >-----------------------------------------
        public void accept(string message, bool Response)
        {
            Object msg = message;
            if(Response == false)
                queue.enQ(msg);
            else
                ResponseQueue.enQ(msg);
        }


        //-----------< send message to another registered cell >----------------
        public bool send(string message, Guid ID, int argc, Object[] args)
        {
            return true;
        }

        //-----------< get unique cell ID >-------------------------------------
        public Guid getID()
        {
            return ID;
        }

        //-----------< capability interrogation >-------------------------------------
        public bool queryCapability(string capability)
        {
            if(CapabilityList.ContainsKey(capability))
                return true;
            else
                return false;
        }

        //-----------< method to send execution messages to other cells >-------
        public Object syncSend(string MessageName, Object[] Params)
        {
            string RequestCellGuid = ID.ToString();
            Object Result = null;
            Messaging msg = new Messaging();
            Guid RequestID;
            string xml = msg.buildRequest(RequestCellGuid,MessageName,
                Result, Params, out RequestID, Guid.Empty);
            Matrix.Matrix.transmit(xml);
            SyncWait sync = new SyncWait(this, RequestID);
            Result = sync.wait();
            return Result;
        }

        //-----------< method to send execution messages to clone cells >-------
        public Object syncSend(string MessageName, Object[] Params, Clone clone)
        {
            Guid guid = clone.guid();
            string RequestCellGuid = ID.ToString();
            Object Result = null;
            Messaging msg = new Messaging();
            Guid RequestID;
            string xml = msg.buildRequest(RequestCellGuid,MessageName,
                Result, Params, out RequestID,guid);
            Matrix.Matrix.transmit(xml);
            SyncWait sync = new SyncWait(this, RequestID);
            Result = sync.wait();
            return Result;
        }
```

```
//-----------< method to obtain a Cell clone for stateful operation >---
public Clone getClone(string MessageName)
{
    Guid g = Matrix.Matrix.createClone(MessageName);
    if(g.CompareTo(Guid.Empty) == 0)
        throw new Exception("Could not locate cell to handle: "+
            MessageName);
    else
        return new Clone(g);
}

//-----< check if a Cell ID is valid >-----------------------------------
bool isValid(Clone clone)
{
    Guid CellID = clone.guid();
    if(CellID.CompareTo(Guid.Empty)==0)
        return false;
    else
        return true;
}

//-----------<add response received flag to ResponseList >--------------
public void addResponse(Guid RequestID)
{
    ResponseList.Add(RequestID, null);
}

//-----------<remove response received flag to ResponseList >-----------
public void removeResponse(Guid RequestID)
{
    if(ResponseList.Contains(RequestID))
    {
        ResponseList.Remove(RequestID);
    }
}

//-----------< retrieve response >---------------------------------------
public bool checkAndRetrieveResponse(Guid RequestID)
{

    if(ResponseList.Contains(RequestID))
    {
        return true;
    }
    else
    {
        return false;
    }
}

//-----< initialize cell >-----------------------------------------------
private void initializeCell()
{
    if(ID.CompareTo(Guid.Empty)==0)
    {
        ID = Guid.NewGuid(); //ID//
    }
    messager = new Messaging();
    queue = new BlockingQueue(
        new BlockingQueue.DequeueDelegate(extract));
    ResponseQueue = new BlockingQueue(
        new BlockingQueue.DequeueDelegate(extractResponse));
    CapabilityList = new System.Collections.Hashtable();
    ResponseList = new System.Collections.Hashtable();
```

```
        }
        #endregion

        //-----< cell entry point >------------------------------------------
        public void start()
        {
            //TODO: add executive processing if required
            //default: do nothing
            try
            {
                Application.Run(this);
            }//any exceptions not caught so far
            catch(Exception ex)
            {
               MessageBox.Show("The following exception occured: " + ex.Message);
            }
        }

        //-----------< calls to message-specific processing >-----------------
        public Object process(ExecObject eo)
        {
            ReturnObject ro = null;
            Exception exception = null;
            Object ret=null;
            try
            {
                //TODO:
                //add calls to processing logic here
            }
            catch(Exception ex)
            {
                exception = ex;
            }
            ro = new ReturnObject(ret,exception);
            return ro;
        }

        //-----< constructor >-----------------------------------------------
        public NetViewForm(int _ServerPort)
        {
            InitializeComponent();
            initializeCell();
            ServerPort = _ServerPort;
            Display d = new Display(this);
        }

        //-----< constructor >-----------------------------------------------
        public NetViewForm()
        {
            InitializeComponent();
            initializeCell();
            Display d = new Display(this);
        }

        //-----< listen for connections >------------------------------------
        public void init(int ListeningPort)
        {
            Object[] Args = new Object[]{ListeningPort};
            ftclone = getClone("SU.FileTransfer.RunServer");
            if(isValid(ftclone))
            {
                bool success = (bool)syncSend("SU.FileTransfer.RunServer",Args,ftclone);
                if(! success)
                    throw new Exception("Could not setup server on port: " +
    ListeningPort);
            }
```

```
        }

        //-----< clean-up >---------------------------------------------------
        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if (components != null)
                {
                    components.Dispose();
                }
            }
            base.Dispose( disposing );
        }

        #region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            System.Resources.ResourceManager resources = new System.Resources.
    ResourceManager(typeof(NetViewForm));
            this.SlideWindow = new System.Windows.Forms.PictureBox();
            this.ConnectButton = new System.Windows.Forms.Button();
            this.RemotePortTextBox = new System.Windows.Forms.TextBox();
            this.RemoteMachineTextBox = new System.Windows.Forms.TextBox();
            this.groupBox1 = new System.Windows.Forms.GroupBox();
            this.label1 = new System.Windows.Forms.Label();
            this.label2 = new System.Windows.Forms.Label();
            this.AddSlidesButton = new System.Windows.Forms.Button();
            this.AddSlidesFileDialog = new System.Windows.Forms.OpenFileDialog();
            this.FirstButton = new System.Windows.Forms.Button();
            this.BackButton = new System.Windows.Forms.Button();
            this.ForwardButton = new System.Windows.Forms.Button();
            this.LastButton = new System.Windows.Forms.Button();
            this.ExitButton = new System.Windows.Forms.Button();
            this.groupBox2 = new System.Windows.Forms.GroupBox();
            this.ListenButton = new System.Windows.Forms.Button();
            this.ServerPortTextBox = new System.Windows.Forms.TextBox();
            this.label3 = new System.Windows.Forms.Label();
            this.pictureBox1 = new System.Windows.Forms.PictureBox();
            this.groupBox1.SuspendLayout();
            this.groupBox2.SuspendLayout();
            this.SuspendLayout();
            //
            // SlideWindow
            //
            this.SlideWindow.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;
            this.SlideWindow.Location = new System.Drawing.Point(8, 72);
            this.SlideWindow.Name = "SlideWindow";
            this.SlideWindow.Size = new System.Drawing.Size(632, 480);
            this.SlideWindow.SizeMode = System.Windows.Forms.PictureBoxSizeMode.
    StretchImage;
            this.SlideWindow.TabIndex = 0;
            this.SlideWindow.TabStop = false;
            //
            // ConnectButton
            //
            this.ConnectButton.Font = new System.Drawing.Font("Tahoma", 9F, System.
    Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.ConnectButton.Location = new System.Drawing.Point(2, 40);
            this.ConnectButton.Name = "ConnectButton";
            this.ConnectButton.Size = new System.Drawing.Size(316, 24);
            this.ConnectButton.TabIndex = 2;
```

```
            this.ConnectButton.Text = "Connect";
            this.ConnectButton.Click += new System.EventHandler(this.ConnectButton_Click)
    ;
            //
            // RemotePortTextBox
            //
            this.RemotePortTextBox.Location = new System.Drawing.Point(232, 16);
            this.RemotePortTextBox.Name = "RemotePortTextBox";
            this.RemotePortTextBox.Size = new System.Drawing.Size(80, 22);
            this.RemotePortTextBox.TabIndex = 3;
            this.RemotePortTextBox.Text = "";
            //
            // RemoteMachineTextBox
            //
            this.RemoteMachineTextBox.Location = new System.Drawing.Point(88, 16);
            this.RemoteMachineTextBox.Name = "RemoteMachineTextBox";
            this.RemoteMachineTextBox.Size = new System.Drawing.Size(112, 22);
            this.RemoteMachineTextBox.TabIndex = 3;
            this.RemoteMachineTextBox.Text = "";
            //
            // groupBox1
            //
            this.groupBox1.Controls.Add(this.label1);
            this.groupBox1.Controls.Add(this.RemoteMachineTextBox);
            this.groupBox1.Controls.Add(this.RemotePortTextBox);
            this.groupBox1.Controls.Add(this.label2);
            this.groupBox1.Controls.Add(this.ConnectButton);
            this.groupBox1.Font = new System.Drawing.Font("Tahoma", 9F, System.Drawing.
    FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.groupBox1.Location = new System.Drawing.Point(8, 592);
            this.groupBox1.Name = "groupBox1";
            this.groupBox1.Size = new System.Drawing.Size(320, 64);
            this.groupBox1.TabIndex = 4;
            this.groupBox1.TabStop = false;
            this.groupBox1.Text = "Connection Settings";
            //
            // label1
            //
            this.label1.Font = new System.Drawing.Font("Tahoma", 9F, System.Drawing.
    FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.label1.Location = new System.Drawing.Point(8, 16);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(80, 18);
            this.label1.TabIndex = 4;
            this.label1.Text = "Remote Host";
            //
            // label2
            //
            this.label2.Font = new System.Drawing.Font("Tahoma", 9F, System.Drawing.
    FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.label2.Location = new System.Drawing.Point(200, 16);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(32, 18);
            this.label2.TabIndex = 4;
            this.label2.Text = "Port";
            //
            // AddSlidesButton
            //
            this.AddSlidesButton.Font = new System.Drawing.Font("Tahoma", 9F, System.
    Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.AddSlidesButton.Location = new System.Drawing.Point(2, 560);
            this.AddSlidesButton.Name = "AddSlidesButton";
            this.AddSlidesButton.Size = new System.Drawing.Size(240, 32);
            this.AddSlidesButton.TabIndex = 5;
            this.AddSlidesButton.Text = "Add Slides...";
            this.AddSlidesButton.Click += new System.EventHandler(this.
```

```
AddSlidesButton_Click);
        //
        // AddSlidesFileDialog
        //
        this.AddSlidesFileDialog.Filter = "All Image Files|*.jpg; *.jpeg; *.bmp; *.
gif|*.jpg|*.jpg|*.bmp|*.bmp|*.gif|*.gif|*" +
            ".jpeg|*.jpeg";
        this.AddSlidesFileDialog.InitialDirectory = "..\\..\\SlideStore";
        this.AddSlidesFileDialog.Multiselect = true;
        this.AddSlidesFileDialog.RestoreDirectory = true;
        this.AddSlidesFileDialog.Title = "Select Slides to Add";
        //
        // FirstButton
        //
        this.FirstButton.BackColor = System.Drawing.Color.White;
        this.FirstButton.Image = ((System.Drawing.Image)(resources.GetObject("
FirstButton.Image")));
        this.FirstButton.Location = new System.Drawing.Point(248, 560);
        this.FirstButton.Name = "FirstButton";
        this.FirstButton.Size = new System.Drawing.Size(29, 29);
        this.FirstButton.TabIndex = 6;
        this.FirstButton.Click += new System.EventHandler(this.FirstButton_Click);
        //
        // BackButton
        //
        this.BackButton.BackColor = System.Drawing.Color.White;
        this.BackButton.Image = ((System.Drawing.Image)(resources.GetObject("
BackButton.Image")));
        this.BackButton.Location = new System.Drawing.Point(280, 560);
        this.BackButton.Name = "BackButton";
        this.BackButton.Size = new System.Drawing.Size(29, 29);
        this.BackButton.TabIndex = 6;
        this.BackButton.Click += new System.EventHandler(this.BackButton_Click);
        //
        // ForwardButton
        //
        this.ForwardButton.BackColor = System.Drawing.Color.White;
        this.ForwardButton.Image = ((System.Drawing.Image)(resources.GetObject("
ForwardButton.Image")));
        this.ForwardButton.Location = new System.Drawing.Point(312, 560);
        this.ForwardButton.Name = "ForwardButton";
        this.ForwardButton.Size = new System.Drawing.Size(29, 29);
        this.ForwardButton.TabIndex = 6;
        this.ForwardButton.Click += new System.EventHandler(this.ForwardButton_Click)
;
        //
        // LastButton
        //
        this.LastButton.BackColor = System.Drawing.Color.White;
        this.LastButton.Image = ((System.Drawing.Image)(resources.GetObject("
LastButton.Image")));
        this.LastButton.Location = new System.Drawing.Point(344, 560);
        this.LastButton.Name = "LastButton";
        this.LastButton.Size = new System.Drawing.Size(29, 29);
        this.LastButton.TabIndex = 6;
        this.LastButton.Click += new System.EventHandler(this.LastButton_Click);
        //
        // ExitButton
        //
        this.ExitButton.Font = new System.Drawing.Font("Tahoma", 9F, System.Drawing.
FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
        this.ExitButton.Location = new System.Drawing.Point(378, 560);
        this.ExitButton.Name = "ExitButton";
        this.ExitButton.Size = new System.Drawing.Size(264, 32);
        this.ExitButton.TabIndex = 5;
        this.ExitButton.Text = "Exit";
```

```
            this.ExitButton.Click += new System.EventHandler(this.button1_Click_1);
            //
            // groupBox2
            //
            this.groupBox2.Controls.Add(this.ListenButton);
            this.groupBox2.Controls.Add(this.ServerPortTextBox);
            this.groupBox2.Controls.Add(this.label3);
            this.groupBox2.Font = new System.Drawing.Font("Tahoma", 9F, System.Drawing.
    FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.groupBox2.Location = new System.Drawing.Point(336, 592);
            this.groupBox2.Name = "groupBox2";
            this.groupBox2.Size = new System.Drawing.Size(312, 64);
            this.groupBox2.TabIndex = 7;
            this.groupBox2.TabStop = false;
            this.groupBox2.Text = "Server Settings";
            //
            // ListenButton
            //
            this.ListenButton.Font = new System.Drawing.Font("Tahoma", 9F, System.Drawing
    .FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.ListenButton.Location = new System.Drawing.Point(2, 40);
            this.ListenButton.Name = "ListenButton";
            this.ListenButton.Size = new System.Drawing.Size(307, 24);
            this.ListenButton.TabIndex = 7;
            this.ListenButton.Text = "Listen...";
            this.ListenButton.Click += new System.EventHandler(this.ListenButton_Click);
            //
            // ServerPortTextBox
            //
            this.ServerPortTextBox.Location = new System.Drawing.Point(224, 16);
            this.ServerPortTextBox.Name = "ServerPortTextBox";
            this.ServerPortTextBox.Size = new System.Drawing.Size(80, 22);
            this.ServerPortTextBox.TabIndex = 5;
            this.ServerPortTextBox.Text = "";
            //
            // label3
            //
            this.label3.Font = new System.Drawing.Font("Tahoma", 9F, System.Drawing.
    FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.label3.Location = new System.Drawing.Point(192, 16);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(32, 18);
            this.label3.TabIndex = 6;
            this.label3.Text = "Port";
            //
            // pictureBox1
            //
            this.pictureBox1.BackColor = System.Drawing.Color.White;
            this.pictureBox1.Image = ((System.Drawing.Image)(resources.GetObject("
    pictureBox1.Image")));
            this.pictureBox1.Location = new System.Drawing.Point(8, 0);
            this.pictureBox1.Name = "pictureBox1";
            this.pictureBox1.Size = new System.Drawing.Size(632, 72);
            this.pictureBox1.TabIndex = 8;
            this.pictureBox1.TabStop = false;
            //
            // NetViewForm
            //
            this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
            this.ClientSize = new System.Drawing.Size(648, 663);
            this.Controls.Add(this.pictureBox1);
            this.Controls.Add(this.groupBox2);
            this.Controls.Add(this.AddSlidesButton);
            this.Controls.Add(this.groupBox1);
            this.Controls.Add(this.SlideWindow);
            this.Controls.Add(this.FirstButton);
```

```
            this.Controls.Add(this.BackButton);
            this.Controls.Add(this.ForwardButton);
            this.Controls.Add(this.LastButton);
            this.Controls.Add(this.ExitButton);
            this.Name = "NetViewForm";
            this.Text = "NetVIEW";
            this.Load += new System.EventHandler(this.NetViewForm_Load);
            this.groupBox1.ResumeLayout(false);
            this.groupBox2.ResumeLayout(false);
            this.ResumeLayout(false);

        }
        #endregion

        //-----< main >-------------------------------------------------------
        [STAThread]
        static void Main(string[] Args)
        {

            Application.Run(new NetViewForm(Int32.Parse(Args[0])));
        }

        //-----< form load handler >------------------------------------------
        private void NetViewForm_Load(object sender, System.EventArgs e)
        {
            ConnectButton.Enabled = false;
            AddSlidesButton.Enabled = false;
            registerNetViewServer();
        }

        //-----< connect click handler >--------------------------------------
        private void ConnectButton_Click(object sender, System.EventArgs e)
        {
            ConnectButton.Enabled = false;
            string RemoteMachine = RemoteMachineTextBox.Text;
            int RemotePort = Int32.Parse(RemotePortTextBox.Text);
            Object[] Args = new Object[]{RemoteMachine, RemotePort};
            bool success = (bool)syncSend("SU.FileTransfer.Connect",Args,ftclone);

            this.RemoteDisplay = (Display)Activator.GetObject(typeof(Display),
                "tcp://" + RemoteMachineTextBox.Text + ":" + RemotePortTextBox.Text + "/ ↙
    NetViewDisplay");
            Connected = true;
        }


        //-----< register NetView service on already established server >-------
        public bool registerNetViewServer()
        {
            RemotingConfiguration.RegisterWellKnownServiceType(typeof(NetView.Display),
                "NetViewDisplay",WellKnownObjectMode.SingleCall);
            return true;
        }

        //-----< display a slide >--------------------------------------------
        public void displaySlide(string path)
        {
            if(this.InvokeRequired)
            {
                DisplayDelegate d = new DisplayDelegate(displaySlide);
                this.Invoke(new DisplayDelegate(displaySlide),new object[] {path});
            }
            else
            {
                StreamReader sr = new StreamReader(path);
                string ser = sr.ReadToEnd(); sr.Close();
```

```
                Object[] Args = new Object[]{ser};
                Image img = (Image)syncSend("SU.TextSerializer.Deserialize",Args);
                this.SlideWindow.Image = img;
                this.SlideWindow.Show();
            }
        }

        //-----< add a slide >-------------------------------------------------
        private void AddSlidesButton_Click(object sender, System.EventArgs e)
        {
            DialogResult dr = AddSlidesFileDialog.ShowDialog();
            if(dr != DialogResult.OK)
            {
                return;

            }
            string[] files = AddSlidesFileDialog.FileNames;
            foreach(string s in files)
            {
                Slides.Add(s);
            }
            cp = new CircularPointer(files.Length,0);

        }

        //-----< forward click handler >---------------------------------------
        private void ForwardButton_Click(object sender, System.EventArgs e)
        {
            try
            {
                string filename = (string)Slides[cp.moveNext()];
                Image img = Image.FromFile(filename);
                Object[] Args = new Object[]{img};
                string ser = (string)syncSend("SU.TextSerializer.Serialize",Args);
                StreamWriter sw = new StreamWriter(StorePath + "temp.tmp");
                sw.Write(ser); sw.Close();
                SlideWindow.Image = img;
                SlideWindow.Show();
                if(Connected)
                {
                    Args = new Object[]{StorePath + "temp.tmp",StorePath + "temp.tmp"};
                    bool success = (bool)syncSend("SU.FileTransfer.UploadLocalFile",Args,↙
    ftclone);
                    RemoteDisplay.display(StorePath + "temp.tmp");
                }
            }
            catch(Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        //-----< back click handler >------------------------------------------
        private void BackButton_Click(object sender, System.EventArgs e)
        {
            try
            {
                string filename = (string)Slides[cp.movePrevious()];
                Image img = Image.FromFile(filename);
                Object[] Args = new Object[]{img};
                string ser = (string)syncSend("SU.TextSerializer.Serialize",Args);
                StreamWriter sw = new StreamWriter(StorePath + "temp.tmp");
                sw.Write(ser); sw.Close();
                SlideWindow.Image = img;
                SlideWindow.Show();
                if(Connected)
```

```
                {
                    Args = new Object[]{StorePath + "temp.tmp",StorePath + "temp.tmp"};
                    bool success = (bool)syncSend("SU.FileTransfer.UploadLocalFile",Args,↙
ftclone);
                    RemoteDisplay.display(StorePath + "temp.tmp");
                }
            }
            catch(Exception ex)
            {
                MessageBox.Show(ex.Message);
            }

        }

        //-----< last click handler >-----------------------------------------
        private void LastButton_Click(object sender, System.EventArgs e)
        {
            try
            {
                string filename = (string)Slides[cp.moveLast()];
                Image img = Image.FromFile(filename);
                Object[] Args = new Object[]{img};
                string ser = (string)syncSend("SU.TextSerializer.Serialize",Args);
                //string ser = textSerialize(img);
                StreamWriter sw = new StreamWriter(StorePath + "temp.tmp");
                sw.Write(ser); sw.Close();
                SlideWindow.Image = img;
                SlideWindow.Show();
                if(Connected)
                {
                    Args = new Object[]{StorePath + "temp.tmp",StorePath + "temp.tmp"};
                    bool success = (bool)syncSend("SU.FileTransfer.UploadLocalFile",Args,↙
ftclone);
                    RemoteDisplay.display(StorePath + "temp.tmp");
                }
            }
            catch(Exception ex)
            {
                MessageBox.Show(ex.Message);
            }

        }

        //-----< first click handler >----------------------------------------
        private void FirstButton_Click(object sender, System.EventArgs e)
        {
            try
            {
                string filename = (string)Slides[cp.moveFirst()];
                Image img = Image.FromFile(filename);
                Object[] Args = new Object[]{img};
                string ser = (string)syncSend("SU.TextSerializer.Serialize",Args);
                StreamWriter sw = new StreamWriter(StorePath + "temp.tmp");
                sw.Write(ser); sw.Close();
                SlideWindow.Image = img;
                SlideWindow.Show();
                if(Connected)
                {
                    Args = new Object[]{StorePath + "temp.tmp",StorePath + "temp.tmp"};
                    bool success = (bool)syncSend("SU.FileTransfer.UploadLocalFile",Args,↙
ftclone);
                    RemoteDisplay.display(StorePath + "temp.tmp");
                }
            }
            catch(Exception ex)
            {
```

```
                MessageBox.Show(ex.Message);
            }
        }

        //-----< exit >------------------------------------------------------
        private void button1_Click_1(object sender, System.EventArgs e)
        {
            Application.Exit();
        }

        //-----< listen click handler >--------------------------------------
        private void ListenButton_Click(object sender, System.EventArgs e)
        {
            ServerPort = Int32.Parse(ServerPortTextBox.Text);
            init(ServerPort);
            ConnectButton.Enabled = true;
            AddSlidesButton.Enabled = true;
            ListenButton.Enabled=false;
        }

    }

    ////////////////////////////////////////////
    //// helper class
    ////////////////////////////////////////////
    public class CircularPointer
    {
        int StartIndex;
        int CurrentIndex;
        int TotalCount;
        //ArrayList Indices = new ArrayList();

        //-----< constructor >-----------------------------------------------
        public CircularPointer(int IndexCount, int start)
        {
            StartIndex = start;
            TotalCount = IndexCount;
            CurrentIndex = StartIndex;
        }

        //-----< advance pointer to next item >------------------------------
        public int moveNext()
        {
            if( (CurrentIndex+1) < TotalCount )
            {
                CurrentIndex++;
            }
            else
            {
                CurrentIndex=StartIndex;
            }
            return CurrentIndex;
        }

        //-----< rewind pointer to previous item >---------------------------
        public int movePrevious()
        {
            if( (CurrentIndex-1) > StartIndex)
            {
                CurrentIndex--;
            }
            else
            {
                CurrentIndex = StartIndex + (TotalCount-1);
            }
            return CurrentIndex;
```

```
        }

        //-----< advance pointer to last item >---------------------------------
        public int moveLast()
        {
            CurrentIndex = StartIndex + (TotalCount-1);
            return CurrentIndex;
        }

        //-----< advance pointer to last item >---------------------------------
        public int moveFirst()
        {
            CurrentIndex = StartIndex;
            return CurrentIndex;
        }

    }


    ////////////////////////////////////////////////////
    //// remotable class to handle slide display
    ////////////////////////////////////////////////////
    public class Display : MarshalByRefObject
    {
        private static NetViewForm Ui;

        //-----< constructor > >-------------------------------------------------
        public Display(NetViewForm _Ui)
        {
            Ui = _Ui;
        }

        //-----< constructor >---------------------------------------------------
        public Display()
        {
            if (Ui == null)
                throw new Exception("Member Ui not initialized");
        }

        //-----< initiate slide display >---------------------------------------
        public void display(string path)
        {
            Ui.displaySlide(path);
        }
    }

}
```

File Synchronizer Matrix Application

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;

namespace FileSynchronizer
{
    public class ChangeConfirmation : System.Windows.Forms.Form
    {
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.ColumnHeader Files_Header_1;
        private System.Windows.Forms.ColumnHeader Files_Header2;
        private System.Windows.Forms.ColumnHeader Files_Header3;
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.Button YesButton;
        private System.Windows.Forms.Button NoButton;
        private System.Windows.Forms.ListView CopyListView;
        private System.Windows.Forms.ListView UpdateListView;
        private System.Windows.Forms.ListView OverwriteListView;
        private System.ComponentModel.Container components = null;

        //-----< constructor >-------------------------------------------------
        public ChangeConfirmation(CompareStatus[] NodesToUpdate)
        {
            //
            // Required for Windows Form Designer support
            //
            InitializeComponent();
            //initialize form
            initGUI(NodesToUpdate);
        }

        //-----< clean-up >----------------------------------------------------
        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if(components != null)
                {
                    components.Dispose();
                }
            }
            base.Dispose( disposing );
        }

        #region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.label1 = new System.Windows.Forms.Label();
            this.label2 = new System.Windows.Forms.Label();
            this.label3 = new System.Windows.Forms.Label();
            this.CopyListView = new System.Windows.Forms.ListView();
            this.Files_Header_1 = new System.Windows.Forms.ColumnHeader();
            this.UpdateListView = new System.Windows.Forms.ListView();
            this.Files_Header2 = new System.Windows.Forms.ColumnHeader();
            this.OverwriteListView = new System.Windows.Forms.ListView();
            this.Files_Header3 = new System.Windows.Forms.ColumnHeader();
            this.label4 = new System.Windows.Forms.Label();
            this.YesButton = new System.Windows.Forms.Button();
```

```
            this.NoButton = new System.Windows.Forms.Button();
            this.SuspendLayout();
            //
            // label1
            //
            this.label1.Font = new System.Drawing.Font("Tahoma", 11.25F, System.Drawing.
    FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.label1.Location = new System.Drawing.Point(0, 0);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(376, 40);
            this.label1.TabIndex = 0;
            this.label1.Text = "The following files will be created in the destination
    directory:";
            //
            // label2
            //
            this.label2.Font = new System.Drawing.Font("Tahoma", 11.25F, System.Drawing.
    FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.label2.Location = new System.Drawing.Point(0, 160);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(376, 40);
            this.label2.TabIndex = 0;
            this.label2.Text = "The following files will be updated in the destination
    directory:";
            //
            // label3
            //
            this.label3.Font = new System.Drawing.Font("Tahoma", 11.25F, System.Drawing.
    FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.label3.Location = new System.Drawing.Point(0, 320);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(368, 48);
            this.label3.TabIndex = 0;
            this.label3.Text = "The following files will be overwritten (with newer
    versions) in the destination " +
                "directory:";
            //
            // CopyListView
            //
            this.CopyListView.Columns.AddRange(new System.Windows.Forms.ColumnHeader[] {
this.Files_Header_1});
            this.CopyListView.GridLines = true;
            this.CopyListView.Location = new System.Drawing.Point(0, 48);
            this.CopyListView.Name = "CopyListView";
            this.CopyListView.Size = new System.Drawing.Size(376, 104);
            this.CopyListView.Sorting = System.Windows.Forms.SortOrder.Ascending;
            this.CopyListView.TabIndex = 1;
            this.CopyListView.View = System.Windows.Forms.View.Details;
            //
            // Files_Header_1
            //
            this.Files_Header_1.Text = "Files";
            this.Files_Header_1.Width = 370;
            //
            // UpdateListView
            //
            this.UpdateListView.Columns.AddRange(new System.Windows.Forms.ColumnHeader[]
    {
this.Files_Header2});
            this.UpdateListView.GridLines = true;
            this.UpdateListView.Location = new System.Drawing.Point(0, 208);
            this.UpdateListView.Name = "UpdateListView";
            this.UpdateListView.Size = new System.Drawing.Size(376, 104);
            this.UpdateListView.Sorting = System.Windows.Forms.SortOrder.Ascending;
            this.UpdateListView.TabIndex = 1;
            this.UpdateListView.View = System.Windows.Forms.View.Details;
```

```
            //
            // Files_Header2
            //
            this.Files_Header2.Text = "Files";
            this.Files_Header2.Width = 371;
            //
            // OverwriteListView
            //
            this.OverwriteListView.Columns.AddRange(new System.Windows.Forms.ColumnHeader
    [] {
this.Files_Header3});
            this.OverwriteListView.GridLines = true;
            this.OverwriteListView.Location = new System.Drawing.Point(0, 368);
            this.OverwriteListView.Name = "OverwriteListView";
            this.OverwriteListView.Size = new System.Drawing.Size(376, 104);
            this.OverwriteListView.Sorting = System.Windows.Forms.SortOrder.Ascending;
            this.OverwriteListView.TabIndex = 1;
            this.OverwriteListView.View = System.Windows.Forms.View.Details;
            //
            // Files_Header3
            //
            this.Files_Header3.Text = "Files";
            this.Files_Header3.Width = 369;
            //
            // label4
            //
            this.label4.Font = new System.Drawing.Font("Tahoma", 11.25F, System.Drawing.
    FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.label4.Location = new System.Drawing.Point(0, 480);
            this.label4.Name = "label4";
            this.label4.Size = new System.Drawing.Size(376, 24);
            this.label4.TabIndex = 2;
            this.label4.Text = "Proceed?";
            //
            // YesButton
            //
            this.YesButton.DialogResult = System.Windows.Forms.DialogResult.OK;
            this.YesButton.Font = new System.Drawing.Font("Tahoma", 11.25F, System.
    Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.YesButton.Location = new System.Drawing.Point(0, 504);
            this.YesButton.Name = "YesButton";
            this.YesButton.Size = new System.Drawing.Size(192, 24);
            this.YesButton.TabIndex = 3;
            this.YesButton.Text = "Yes";
            this.YesButton.Click += new System.EventHandler(this.YesButton_Click);
            //
            // NoButton
            //
            this.NoButton.DialogResult = System.Windows.Forms.DialogResult.Cancel;
            this.NoButton.Font = new System.Drawing.Font("Tahoma", 11.25F, System.Drawing
    .FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.NoButton.Location = new System.Drawing.Point(192, 504);
            this.NoButton.Name = "NoButton";
            this.NoButton.Size = new System.Drawing.Size(184, 24);
            this.NoButton.TabIndex = 3;
            this.NoButton.Text = "No";
            this.NoButton.Click += new System.EventHandler(this.NoButton_Click);
            //
            // ChangeConfirmation
            //
            this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
            this.ClientSize = new System.Drawing.Size(376, 526);
            this.Controls.Add(this.YesButton);
            this.Controls.Add(this.label4);
            this.Controls.Add(this.CopyListView);
            this.Controls.Add(this.label1);
```

```
            this.Controls.Add(this.label2);
            this.Controls.Add(this.label3);
            this.Controls.Add(this.UpdateListView);
            this.Controls.Add(this.OverwriteListView);
            this.Controls.Add(this.NoButton);
            this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;
            this.Name = "ChangeConfirmation";
            this.Text = "Change Confirmation";
            this.ResumeLayout(false);

        }
        #endregion

        //-----< UI control handler >----------------------------------------
        private void YesButton_Click(object sender, System.EventArgs e)
        {
            this.Close();
        }

        //-----< UI control handler >----------------------------------------
        private void NoButton_Click(object sender, System.EventArgs e)
        {
            this.Close();
        }

        //-----< initialize form >-------------------------------------------
        private void initGUI(CompareStatus[] NodesToUpdate)
        {
            foreach(CompareStatus cs in NodesToUpdate)
            {
                Node n = cs.node();
                switch(cs.status())
                {
                    case CompareStatus.Status.Copy:
                        CopyListView.Items.Add(n._name+ "("+n.timeStamp()+")");
                        break;
                    case CompareStatus.Status.Update:
                        UpdateListView.Items.Add(n._name+ "("+n.timeStamp()+")");
                        break;
                    case CompareStatus.Status.Overwrite:
                        OverwriteListView.Items.Add(n._name+ "("+n.timeStamp()+")");
                        break;
                }
            }
        }
    }
}
```

```
/////////////////////////////////////////////////////////////////
//  UI.cs          Implements a Cell resposnsible for the      //
//                 graphical user interface of the File        //
//                 Synchronizer application.                   //
//                 Has an empty capability list.               //
//                                                             //
//  ver 1.0                                                    //
//                                                             //
//  Language:      C#                                          //
//  Platform:      Dell Dimension Pentium 4, Windows 2000      //
//  Course:        CSE997 Master's Thesis                      //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>           //
//                                                             //
/////////////////////////////////////////////////////////////////
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.IO;
using System.Resources;
using Matrix;

namespace FileSynchronizer
{

    ////////////////////////////////////////////////////////////////////////
    ///// class implementing
    ///// the graphical user interface
    ///// for this program
    ////////////////////////////////////////////////////////////////////////
    public class UI : System.Windows.Forms.Form, Cell
    {
        private System.ComponentModel.Container components = null;
        private int ListeningPort = 0;
        private string RemoteMachine = "";
        private System.Windows.Forms.RichTextBox ConsoleTextBox;
        private int RemotePort = 0;
        private Clone clone;

        private TreeViewExplorer LocalExplorer;
        private TreeViewExplorer RemoteExplorer;
        private System.Windows.Forms.Button ConnectTextBox;
        private System.Windows.Forms.Button StartButton;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.TextBox RemoteHostTextBox;
        private System.Windows.Forms.Label label6;
        private System.Windows.Forms.Label label7;
        private System.Windows.Forms.Button DestinationToSourceButton;
        private System.Windows.Forms.Button SourceToDestUpdateButton;
        private System.Windows.Forms.ListView SourceFilesList;
        private System.Windows.Forms.ColumnHeader FileNameHeader;
        private System.Windows.Forms.TextBox DestinationTextBox;
        private System.Windows.Forms.TextBox SourceTextBox;
        private System.Windows.Forms.TreeView RemoteTreeView;
        private System.Windows.Forms.TreeView LocalTreeView;
        private System.Windows.Forms.ListView DestinationFilesList;
        private System.Windows.Forms.ColumnHeader FileNameHeader2;
        private System.Windows.Forms.Panel panel2;
        private System.Windows.Forms.CheckBox LocalSelectAllCheckBox;
        private System.Windows.Forms.CheckBox RemoteSelectAllCheckBox;
        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.Button LocalRefreshButton;
```

```
        private System.Windows.Forms.Button RemoteRefreshButton;
        private ImageList images;

        #region Wizard Generated Code
        private Guid ID;
        private Messaging messager;
        private BlockingQueue queue;
        private BlockingQueue ResponseQueue;
        private System.Threading.Thread StartThread;
        private System.Collections.Hashtable ResponseList;
        private System.Windows.Forms.TextBox ServerPortTextBox;
        private System.Windows.Forms.TextBox RemotePortTextBox;
        private System.Windows.Forms.PictureBox pictureBox1;
        protected System.Collections.Hashtable CapabilityList;
        public event ReceiveDelegate OnReceive;

        //-----------< process messages pulled out of the queue >---------------
        public void extract(object msg)
        {
            ExecObject eo = messager.extract(msg);
            if(eo==null)
                return;
            if(eo.isRequest())
            {
                Object result = process(eo);
                Guid RequestCellGUID;
                String response = messager.buildResponse(msg,result,out RequestCellGUID);
                Matrix.Matrix.send(response, RequestCellGUID);
            }
                //Response Received
            else
            {
                ReceiveInfoEventArgs rargs = new ReceiveInfoEventArgs(new Guid(eo.
    getRequestID()), eo.getReturnValue());
                if(this.OnReceive!=null)
                    OnReceive(this, rargs);
            }
        }

        //-----------< method to register cell in matrix >----------------------
        public bool register()
        {
            Matrix.Matrix.register((Cell)this);
            StartThread = new System.Threading.Thread(new System.Threading.ThreadStart(
    start));
            StartThread.Start();
            return true;
        }

        //-----------< method to unregister cell in matrix >--------------------
        public bool unregister()
        {
            Matrix.Matrix.unregister(this);
            StartThread.Abort();
            return true;
        }

        //-----------< accept a request message >-------------------------------------
        public void accept(string message)
        {
            accept(message,false);
        }

        //-----< accept a response message >-----------------------------------------
        public void accept(string message, bool Response)
        {
```

```
            Object msg = message;
            if(Response == false)
                queue.enQ(msg);
            else
                ResponseQueue.enQ(msg);
        }


        //-----------< send message to another registered cell >----------------
        public bool send(string message, Guid ID, int argc, Object[] args)
        {
            return true;
        }

        //-----------< get unique cell ID >-------------------------------------
        public Guid getID()
        {
            return ID;
        }

        //-----------< capability interrogation >-------------------------------------
        public bool queryCapability(string capability)
        {
            if(CapabilityList.ContainsKey(capability))
                return true;
            else
                return false;
        }

        //-----------< method to send execution messages to other cells >-------
        public Object syncSend(string MessageName, Object[] Params)
        {
            string RequestCellGuid = ID.ToString();
            Object Result = null;
            Messaging msg = new Messaging();
            Guid RequestID;
            string xml = msg.buildRequest(RequestCellGuid,MessageName,
                Result, Params, out RequestID, Guid.Empty);
            Matrix.Matrix.transmit(xml);
            SyncWait sync = new SyncWait(this, RequestID);
            Result = sync.wait();
            return Result;
        }
        //-----------< method to send execution messages to clone cells >-------
        public Object syncSend(string MessageName, Object[] Params, Clone clone)
        {
            Guid guid = clone.guid();
            string RequestCellGuid = ID.ToString();
            Object Result = null;
            Messaging msg = new Messaging();
            Guid RequestID;
            string xml = msg.buildRequest(RequestCellGuid,MessageName,
                Result, Params, out RequestID,guid);
            Matrix.Matrix.transmit(xml);
            SyncWait sync = new SyncWait(this, RequestID);
            Result = sync.wait();
            return Result;
        }

        //-----------< method to obtain a Cell clone for stateful operation >---
        public Clone getClone(string MessageName)
        {
            Guid g = Matrix.Matrix.createClone(MessageName);
            if(g.CompareTo(Guid.Empty) == 0)
                throw new Exception("Could not locate cell to handle: "+
                    MessageName);
```

```
            else
                return new Clone(g);
        }

        //-----< check if a Cell ID is valid >---------------------------------
        bool isValid(Clone clone)
        {
            Guid CellID = clone.guid();
            if(CellID.CompareTo(Guid.Empty)==0)
                return false;
            else
                return true;
        }

        //-----------<add response received flag to ResponseList >--------------
        public void addResponse(Guid RequestID)
        {
            ResponseList.Add(RequestID, null);
        }

        //-----------<remove response received flag to ResponseList >-----------
        public void removeResponse(Guid RequestID)
        {
            if(ResponseList.Contains(RequestID))
            {
                ResponseList.Remove(RequestID);
            }
        }

        //-----------< retrieve response >-------------------------------------
        public bool checkAndRetrieveResponse(Guid RequestID)
        {

            if(ResponseList.Contains(RequestID))
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        //-----< initialize cell >---------------------------------------------
        private void initializeCell()
        {
            if(ID.CompareTo(Guid.Empty)==0)
            {
                ID = Guid.NewGuid(); //ID//
            }
            messager = new Messaging();
            queue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extract));
            ResponseQueue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extract));
            CapabilityList = new System.Collections.Hashtable();
            ResponseList = new System.Collections.Hashtable();
        }
        #endregion

        //-----< cell entry point >--------------------------------------------
        public void start()
        {
            //TODO: add executive processing if required
            //default: do nothing
            try
```

```
            {
                Application.Run(this);
            }//any exceptions not caught so far
            catch(Exception ex)
            {
                MessageBox.Show("The following exception occured: " + ex.Message);
            }
        }

        //-----------< calls to message-specific processing >-------------------
        public Object process(ExecObject eo)
        {
            ReturnObject ro = null;
            Exception exception = null;
            Object ret=null;
            try
            {
                //TODO:
                //add calls to processing logic here
            }
            catch(Exception ex)
            {
                exception = ex;
            }
            ro = new ReturnObject(ret,exception);
            return ro;
        }

        //-----< constructor >-------------------------------------------------
        public UI()
        {
            // Required for Windows Form Designer support
            InitializeComponent();

            //
            initializeCell();
            //TODO:
            //Add constructor logic here

            images = new ImageList();
            ResourceManager Res = new ResourceManager("UI.Icons", GetType().Assembly);
            System.Drawing.Bitmap  folder = (Bitmap)Res.GetObject("Folder");
            System.Drawing.Bitmap  file = (Bitmap)Res.GetObject("File");
            System.Drawing.Bitmap  drive = (Bitmap)Res.GetObject("Drive");
            images.Images.Add(folder);
            images.Images.Add(file);
            images.Images.Add(drive);

            LocalExplorer = new TreeViewExplorer(LocalTreeView, SourceFilesList);
            RemoteExplorer = new TreeViewExplorer(RemoteTreeView, DestinationFilesList);

            LocalTreeView.ImageList=images;
            RemoteTreeView.ImageList=images;
        }

        //-----< clean-up >----------------------------------------------------
        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if (components != null)
                {
                    components.Dispose();
                }
            }
            base.Dispose( disposing );
```

```
        }

        #region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            System.Resources.ResourceManager resources = new System.Resources.
    ResourceManager(typeof(UI));
            this.ConsoleTextBox = new System.Windows.Forms.RichTextBox();
            this.label3 = new System.Windows.Forms.Label();
            this.label2 = new System.Windows.Forms.Label();
            this.ServerPortTextBox = new System.Windows.Forms.TextBox();
            this.label1 = new System.Windows.Forms.Label();
            this.RemoteHostTextBox = new System.Windows.Forms.TextBox();
            this.RemotePortTextBox = new System.Windows.Forms.TextBox();
            this.ConnectTextBox = new System.Windows.Forms.Button();
            this.StartButton = new System.Windows.Forms.Button();
            this.label6 = new System.Windows.Forms.Label();
            this.label7 = new System.Windows.Forms.Label();
            this.DestinationToSourceButton = new System.Windows.Forms.Button();
            this.SourceToDestUpdateButton = new System.Windows.Forms.Button();
            this.SourceFilesList = new System.Windows.Forms.ListView();
            this.FileNameHeader = new System.Windows.Forms.ColumnHeader();
            this.DestinationTextBox = new System.Windows.Forms.TextBox();
            this.SourceTextBox = new System.Windows.Forms.TextBox();
            this.RemoteTreeView = new System.Windows.Forms.TreeView();
            this.LocalTreeView = new System.Windows.Forms.TreeView();
            this.DestinationFilesList = new System.Windows.Forms.ListView();
            this.FileNameHeader2 = new System.Windows.Forms.ColumnHeader();
            this.panel2 = new System.Windows.Forms.Panel();
            this.LocalSelectAllCheckBox = new System.Windows.Forms.CheckBox();
            this.RemoteSelectAllCheckBox = new System.Windows.Forms.CheckBox();
            this.groupBox1 = new System.Windows.Forms.GroupBox();
            this.LocalRefreshButton = new System.Windows.Forms.Button();
            this.RemoteRefreshButton = new System.Windows.Forms.Button();
            this.pictureBox1 = new System.Windows.Forms.PictureBox();
            this.panel2.SuspendLayout();
            this.groupBox1.SuspendLayout();
            this.SuspendLayout();
            //
            // ConsoleTextBox
            //
            this.ConsoleTextBox.Location = new System.Drawing.Point(83, 488);
            this.ConsoleTextBox.Name = "ConsoleTextBox";
            this.ConsoleTextBox.Size = new System.Drawing.Size(549, 48);
            this.ConsoleTextBox.TabIndex = 3;
            this.ConsoleTextBox.Text = "";
            //
            // label3
            //
            this.label3.Location = new System.Drawing.Point(8, 40);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(64, 24);
            this.label3.TabIndex = 11;
            this.label3.Text = "Remote Port";
            //
            // label2
            //
            this.label2.Enabled = false;
            this.label2.Location = new System.Drawing.Point(200, 40);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(59, 24);
            this.label2.TabIndex = 12;
```

```
            this.label2.Text = "Server (local) Port";
            //
            // ServerPortTextBox
            //
            this.ServerPortTextBox.Enabled = false;
            this.ServerPortTextBox.Location = new System.Drawing.Point(200, 16);
            this.ServerPortTextBox.Name = "ServerPortTextBox";
            this.ServerPortTextBox.Size = new System.Drawing.Size(56, 20);
            this.ServerPortTextBox.TabIndex = 7;
            this.ServerPortTextBox.Text = "9000";
            //
            // label1
            //
            this.label1.Location = new System.Drawing.Point(80, 40);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(80, 24);
            this.label1.TabIndex = 10;
            this.label1.Text = "Remote Host";
            //
            // RemoteHostTextBox
            //
            this.RemoteHostTextBox.Location = new System.Drawing.Point(80, 16);
            this.RemoteHostTextBox.Name = "RemoteHostTextBox";
            this.RemoteHostTextBox.Size = new System.Drawing.Size(112, 20);
            this.RemoteHostTextBox.TabIndex = 8;
            this.RemoteHostTextBox.Text = "localhost";
            //
            // RemotePortTextBox
            //
            this.RemotePortTextBox.Location = new System.Drawing.Point(8, 16);
            this.RemotePortTextBox.Name = "RemotePortTextBox";
            this.RemotePortTextBox.Size = new System.Drawing.Size(57, 20);
            this.RemotePortTextBox.TabIndex = 9;
            this.RemotePortTextBox.Text = "9000";
            //
            // ConnectTextBox
            //
            this.ConnectTextBox.Location = new System.Drawing.Point(8, 69);
            this.ConnectTextBox.Name = "ConnectTextBox";
            this.ConnectTextBox.Size = new System.Drawing.Size(248, 24);
            this.ConnectTextBox.TabIndex = 6;
            this.ConnectTextBox.Text = "Connect";
            this.ConnectTextBox.Click += new System.EventHandler(this.
    ConnectTextBox_Click);
            //
            // StartButton
            //
            this.StartButton.Enabled = false;
            this.StartButton.Location = new System.Drawing.Point(0, 648);
            this.StartButton.Name = "StartButton";
            this.StartButton.Size = new System.Drawing.Size(136, 24);
            this.StartButton.TabIndex = 5;
            this.StartButton.Text = "Start";
            this.StartButton.Click += new System.EventHandler(this.StartButton_Click);
            //
            // label6
            //
            this.label6.Location = new System.Drawing.Point(2, 536);
            this.label6.Name = "label6";
            this.label6.Size = new System.Drawing.Size(80, 24);
            this.label6.TabIndex = 10;
            this.label6.Text = "Local to Remote";
            //
            // label7
            //
            this.label7.Location = new System.Drawing.Point(632, 536);
```

```
            this.label7.Name = "label7";
            this.label7.Size = new System.Drawing.Size(80, 24);
            this.label7.TabIndex = 11;
            this.label7.Text = "Remote to Local";
            //
            // DestinationToSourceButton
            //
            this.DestinationToSourceButton.FlatStyle = System.Windows.Forms.FlatStyle.
    Popup;
            this.DestinationToSourceButton.Image = ((System.Drawing.Image)(resources.
    GetObject("DestinationToSourceButton.Image")));
            this.DestinationToSourceButton.Location = new System.Drawing.Point(632, 488);
            this.DestinationToSourceButton.Name = "DestinationToSourceButton";
            this.DestinationToSourceButton.Size = new System.Drawing.Size(80, 48);
            this.DestinationToSourceButton.TabIndex = 20;
            this.DestinationToSourceButton.Text = "Update";
            this.DestinationToSourceButton.Click += new System.EventHandler(this.
    DestinationToSourceButton_Click);
            //
            // SourceToDestUpdateButton
            //
            this.SourceToDestUpdateButton.FlatStyle = System.Windows.Forms.FlatStyle.
    Popup;
            this.SourceToDestUpdateButton.Image = ((System.Drawing.Image)(resources.
    GetObject("SourceToDestUpdateButton.Image")));
            this.SourceToDestUpdateButton.Location = new System.Drawing.Point(2, 488);
            this.SourceToDestUpdateButton.Name = "SourceToDestUpdateButton";
            this.SourceToDestUpdateButton.Size = new System.Drawing.Size(80, 48);
            this.SourceToDestUpdateButton.TabIndex = 19;
            this.SourceToDestUpdateButton.Text = "Update";
            this.SourceToDestUpdateButton.Click += new System.EventHandler(this.
    SourceToDestUpdateButton_Click);
            //
            // SourceFilesList
            //
            this.SourceFilesList.CheckBoxes = true;
            this.SourceFilesList.Columns.AddRange(new System.Windows.Forms.ColumnHeader[]
     {
this.FileNameHeader});
            this.SourceFilesList.FullRowSelect = true;
            this.SourceFilesList.GridLines = true;
            this.SourceFilesList.Location = new System.Drawing.Point(0, 160);
            this.SourceFilesList.Name = "SourceFilesList";
            this.SourceFilesList.Size = new System.Drawing.Size(352, 248);
            this.SourceFilesList.Sorting = System.Windows.Forms.SortOrder.Ascending;
            this.SourceFilesList.TabIndex = 18;
            this.SourceFilesList.View = System.Windows.Forms.View.Details;
            //
            // FileNameHeader
            //
            this.FileNameHeader.Text = "Files";
            this.FileNameHeader.Width = 349;
            //
            // DestinationTextBox
            //
            this.DestinationTextBox.Font = new System.Drawing.Font("Arial Narrow", 9F,
    System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0
    )));
            this.DestinationTextBox.Location = new System.Drawing.Point(360, 136);
            this.DestinationTextBox.Name = "DestinationTextBox";
            this.DestinationTextBox.ReadOnly = true;
            this.DestinationTextBox.Size = new System.Drawing.Size(352, 21);
            this.DestinationTextBox.TabIndex = 17;
            this.DestinationTextBox.Text = "";
            //
            // SourceTextBox
```

```
            //
            this.SourceTextBox.Font = new System.Drawing.Font("Arial Narrow", 9F, System.
    Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((System.Byte)(0)));
            this.SourceTextBox.Location = new System.Drawing.Point(0, 136);
            this.SourceTextBox.Name = "SourceTextBox";
            this.SourceTextBox.ReadOnly = true;
            this.SourceTextBox.Size = new System.Drawing.Size(352, 21);
            this.SourceTextBox.TabIndex = 15;
            this.SourceTextBox.Text = "";
            //
            // RemoteTreeView
            //
            this.RemoteTreeView.HotTracking = true;
            this.RemoteTreeView.ImageIndex = -1;
            this.RemoteTreeView.Location = new System.Drawing.Point(360, 0);
            this.RemoteTreeView.Name = "RemoteTreeView";
            this.RemoteTreeView.SelectedImageIndex = -1;
            this.RemoteTreeView.Size = new System.Drawing.Size(352, 136);
            this.RemoteTreeView.Sorted = true;
            this.RemoteTreeView.TabIndex = 14;
            this.RemoteTreeView.AfterSelect += new System.Windows.Forms.
    TreeViewEventHandler(this.RemoteTreeView_AfterSelect);
            //
            // LocalTreeView
            //
            this.LocalTreeView.FullRowSelect = true;
            this.LocalTreeView.HotTracking = true;
            this.LocalTreeView.ImageIndex = -1;
            this.LocalTreeView.Location = new System.Drawing.Point(0, 0);
            this.LocalTreeView.Name = "LocalTreeView";
            this.LocalTreeView.SelectedImageIndex = -1;
            this.LocalTreeView.Size = new System.Drawing.Size(352, 136);
            this.LocalTreeView.Sorted = true;
            this.LocalTreeView.TabIndex = 13;
            this.LocalTreeView.AfterSelect += new System.Windows.Forms.
    TreeViewEventHandler(this.LocalTreeView_AfterSelect);
            //
            // DestinationFilesList
            //
            this.DestinationFilesList.CheckBoxes = true;
            this.DestinationFilesList.Columns.AddRange(new System.Windows.Forms.
    ColumnHeader[] {
this.FileNameHeader2});
            this.DestinationFilesList.FullRowSelect = true;
            this.DestinationFilesList.GridLines = true;
            this.DestinationFilesList.Location = new System.Drawing.Point(360, 160);
            this.DestinationFilesList.Name = "DestinationFilesList";
            this.DestinationFilesList.Size = new System.Drawing.Size(352, 248);
            this.DestinationFilesList.Sorting = System.Windows.Forms.SortOrder.Ascending;
            this.DestinationFilesList.TabIndex = 18;
            this.DestinationFilesList.View = System.Windows.Forms.View.Details;
            //
            // FileNameHeader2
            //
            this.FileNameHeader2.Text = "Files";
            this.FileNameHeader2.Width = 349;
            //
            // panel2
            //
            this.panel2.Controls.Add(this.SourceFilesList);
            this.panel2.Controls.Add(this.DestinationTextBox);
            this.panel2.Controls.Add(this.SourceTextBox);
            this.panel2.Controls.Add(this.RemoteTreeView);
            this.panel2.Controls.Add(this.LocalTreeView);
            this.panel2.Controls.Add(this.DestinationFilesList);
            this.panel2.Location = new System.Drawing.Point(0, 56);
```

```
            this.panel2.Name = "panel2";
            this.panel2.Size = new System.Drawing.Size(714, 408);
            this.panel2.TabIndex = 9;
            //
            // LocalSelectAllCheckBox
            //
            this.LocalSelectAllCheckBox.Location = new System.Drawing.Point(2, 464);
            this.LocalSelectAllCheckBox.Name = "LocalSelectAllCheckBox";
            this.LocalSelectAllCheckBox.Size = new System.Drawing.Size(216, 16);
            this.LocalSelectAllCheckBox.TabIndex = 21;
            this.LocalSelectAllCheckBox.Text = "Select all";
            this.LocalSelectAllCheckBox.CheckedChanged += new System.EventHandler(this.
    LocalSelectAllCheckBox_CheckedChanged);
            //
            // RemoteSelectAllCheckBox
            //
            this.RemoteSelectAllCheckBox.Location = new System.Drawing.Point(361, 464);
            this.RemoteSelectAllCheckBox.Name = "RemoteSelectAllCheckBox";
            this.RemoteSelectAllCheckBox.Size = new System.Drawing.Size(216, 16);
            this.RemoteSelectAllCheckBox.TabIndex = 22;
            this.RemoteSelectAllCheckBox.Text = "Select all";
            this.RemoteSelectAllCheckBox.CheckedChanged += new System.EventHandler(this.
    RemoteSelectAllCheckBox_CheckedChanged);
            //
            // groupBox1
            //
            this.groupBox1.Controls.Add(this.label2);
            this.groupBox1.Controls.Add(this.ServerPortTextBox);
            this.groupBox1.Controls.Add(this.RemoteHostTextBox);
            this.groupBox1.Controls.Add(this.RemotePortTextBox);
            this.groupBox1.Controls.Add(this.label3);
            this.groupBox1.Controls.Add(this.label1);
            this.groupBox1.Controls.Add(this.ConnectTextBox);
            this.groupBox1.Location = new System.Drawing.Point(448, 568);
            this.groupBox1.Name = "groupBox1";
            this.groupBox1.Size = new System.Drawing.Size(264, 96);
            this.groupBox1.TabIndex = 23;
            this.groupBox1.TabStop = false;
            this.groupBox1.Text = "Connection Settings";
            //
            // LocalRefreshButton
            //
            this.LocalRefreshButton.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
            this.LocalRefreshButton.Location = new System.Drawing.Point(224, 464);
            this.LocalRefreshButton.Name = "LocalRefreshButton";
            this.LocalRefreshButton.Size = new System.Drawing.Size(128, 24);
            this.LocalRefreshButton.TabIndex = 24;
            this.LocalRefreshButton.Text = "Refresh";
            this.LocalRefreshButton.Click += new System.EventHandler(this.
    LocalRefreshButton_Click);
            //
            // RemoteRefreshButton
            //
            this.RemoteRefreshButton.FlatStyle = System.Windows.Forms.FlatStyle.Popup;
            this.RemoteRefreshButton.Location = new System.Drawing.Point(584, 464);
            this.RemoteRefreshButton.Name = "RemoteRefreshButton";
            this.RemoteRefreshButton.Size = new System.Drawing.Size(128, 24);
            this.RemoteRefreshButton.TabIndex = 25;
            this.RemoteRefreshButton.Text = "Refresh";
            this.RemoteRefreshButton.Click += new System.EventHandler(this.
    RemoteRefreshButton_Click);
            //
            // pictureBox1
            //
            this.pictureBox1.BackColor = System.Drawing.Color.White;
            this.pictureBox1.Image = ((System.Drawing.Image)(resources.GetObject("
```

```
    pictureBox1.Image")));
            this.pictureBox1.Location = new System.Drawing.Point(0, 0);
            this.pictureBox1.Name = "pictureBox1";
            this.pictureBox1.Size = new System.Drawing.Size(704, 56);
            this.pictureBox1.TabIndex = 26;
            this.pictureBox1.TabStop = false;
            //
            // UI
            //
            this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
            this.ClientSize = new System.Drawing.Size(714, 672);
            this.Controls.Add(this.pictureBox1);
            this.Controls.Add(this.RemoteRefreshButton);
            this.Controls.Add(this.LocalRefreshButton);
            this.Controls.Add(this.groupBox1);
            this.Controls.Add(this.RemoteSelectAllCheckBox);
            this.Controls.Add(this.LocalSelectAllCheckBox);
            this.Controls.Add(this.label7);
            this.Controls.Add(this.label6);
            this.Controls.Add(this.panel2);
            this.Controls.Add(this.ConsoleTextBox);
            this.Controls.Add(this.SourceToDestUpdateButton);
            this.Controls.Add(this.DestinationToSourceButton);
            this.Controls.Add(this.StartButton);
            this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;
            this.MaximizeBox = false;
            this.Name = "UI";
            this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;
            this.Text = "File Synchronizer v1.1";
            this.Load += new System.EventHandler(this.UI_Load);
            this.panel2.ResumeLayout(false);
            this.groupBox1.ResumeLayout(false);
            this.ResumeLayout(false);

        }
        #endregion

        //-----< program entry-point >-----------------------------------------
        [STAThread]
        static void Main()
        {
            try
            {
                Application.Run(new UI());
            }//any exceptions not caught so far
            catch(Exception ex)
            {
                MessageBox.Show("The following exception occured: " + ex.Message);
            }

        }

        //-----< click handler for connect textbox >---------------------------
        private void ConnectTextBox_Click(object sender, System.EventArgs e)
        {
            ConnectTextBox.Enabled = false;

            try
            {
                RemoteMachine = RemoteHostTextBox.Text;
                RemotePort = Int32.Parse(RemotePortTextBox.Text);
                Object[] Args = new Object[]{RemoteMachine, RemotePort};
                bool success = (bool)syncSend("SU.FileTransfer.Connect",Args,clone);
                if(!success)
                {
                    MessageBox.Show("Could not locate remote machine");
```

```
                }
                initGUI();
            }
            catch(Exception ex)
            {
                MessageBox.Show(ex.Message + "\n"+ConsoleTextBox.Text);
            }
        }

        //-----< init server >-------------------------------------------------
        private void init()
        {
            ListeningPort = Int32.Parse(ServerPortTextBox.Text);
            Object[] Args = new Object[]{ListeningPort};
            clone = getClone("SU.FileTransfer.RunServer");
            if(isValid(clone))
            {
                bool success = (bool)syncSend("SU.FileTransfer.RunServer",Args,clone);
                if(! success)
                    throw new Exception("Could not setup server on port: " +
    ListeningPort);
            }
        }

        //-----< initialize user interface >-----------------------------------
        private void initGUI()
        {
            Node[] LocalNodes = (Node[])syncSend("SU.FileTransfer.GetLocalDirsFiles",null
    );
            LocalExplorer.addChildren(LocalNodes);
            if(isValid(clone))
            {
                Node[] RemoteNodes = (Node[])syncSend("SU.FileTransfer.GetRemoteDirsFiles
    ",null,clone);
                RemoteExplorer.addChildren(RemoteNodes);
            }
        }

        //-----< handler for form load event >---------------------------------
        private void UI_Load(object sender, System.EventArgs e)
        {
            try
            {
                init();
            }
            catch(Exception ex)
            {
                ConsoleTextBox.Text = ex.Message + "\n"+ConsoleTextBox.Text;
            }
        }

        //-----< click handler for start button >------------------------------
        private void StartButton_Click(object sender, System.EventArgs e)
        {
            //StartButton.Enabled= false;
            //init();
        }

        //-----< select handler for local tree view >--------------------------
        private void LocalTreeView_AfterSelect(object sender, System.Windows.Forms.
    TreeViewEventArgs e)
        {
            TreeNode tn = e.Node;
            Node n = (Node)tn.Tag;
            SourceTextBox.Text = n._path;
            if(!n.isDir())
```

```
            return;
        try
        {
            Object[] Args = new Object[]{n._path };
            Node[] nodes = (Node[])syncSend("SU.FileTransfer.GetLocalDirsFiles",Args)
;
            LocalExplorer.addChildren(tn,nodes);
        }
        catch(Exception ex)
        {
            ConsoleTextBox.Text = ex.Message + "\n"+ConsoleTextBox.Text;
        }
    }

    //-----< select handler for remote tree view >-------------------------
    private void RemoteTreeView_AfterSelect(object sender, System.Windows.Forms.
TreeViewEventArgs e)
    {
        TreeNode tn = e.Node;
        Node n = (Node)tn.Tag;
        DestinationTextBox.Text = n._path;
        if(!n.isDir())
            return;
        try
        {
            Object[] Args = new Object[]{n._path };
            if(isValid(clone))
            {
                Node[] nodes = (Node[])syncSend("SU.FileTransfer.GetRemoteDirsFiles",
Args,clone);
                RemoteExplorer.addChildren(tn,nodes);
            }
        }
        catch(Exception ex)
        {
            ConsoleTextBox.Text = ex.Message + "\n"+ConsoleTextBox.Text;
        }
    }

    //-----< source to destination update button handler >-----------------
    private void SourceToDestUpdateButton_Click(object sender, System.EventArgs e)
    {
        if(SourceTextBox.Text.Trim()=="" || DestinationTextBox.Text.Trim()=="")
        {
            MessageBox.Show("Please select both source and destination","File
Synchronizer");
            return;
        }
        updateFiles(true); //local --> remote
        RemoteRefreshButton_Click(null,null); //refresh the other one depending on
which button, let UI handle
    }

    private void DestinationToSourceButton_Click(object sender, System.EventArgs e)
    {
        if(SourceTextBox.Text.Trim()=="" || DestinationTextBox.Text.Trim()=="")
        {
            MessageBox.Show("Please select both source and destination","File
Synchronizer");
            return;
        }
        updateFiles(false); //remote --> local
        LocalRefreshButton_Click(null,null);
    }

    //-----< update files in selected direction >--------------------------------
```

```
        private void updateFiles(bool DirectionLocalToRemote)
        {
            ListView Source;
            ListView Destination;
            if(DirectionLocalToRemote) //local-->remote
            {
                Source = SourceFilesList;
                Destination = DestinationFilesList;
            }
            else //remote-->local
            {
                Source = DestinationFilesList;
                Destination = SourceFilesList;
            }
            ArrayList SourceNodes = new ArrayList();
            foreach(ListViewItem lvi in Source.Items)//destfileslist for remote to local
            {
                //only checked ones
                if(lvi.Checked == true)
                    SourceNodes.Add(lvi.Tag);
            }
            ArrayList DestNodes = new ArrayList();
            foreach(ListViewItem lvi in Destination.Items)//sourcefileslist for remote to↙
 local
            {
                //all nodes
                DestNodes.Add(lvi.Tag);
            }
            Object[] Args = new Object[]{(Node[])SourceNodes.ToArray(typeof(Node)),
                                         (Node[])DestNodes.ToArray(typeof(Node))};
            CompareStatus[] NodesToUpdate = (CompareStatus[])syncSend("SU.Synchronizer.  ↙
Policy.Compare",Args);
            ChangeConfirmation CcForm = new ChangeConfirmation(NodesToUpdate);
            DialogResult dr = CcForm.ShowDialog();
            if(dr == DialogResult.Cancel)
                return;
            transferFiles(NodesToUpdate, DirectionLocalToRemote); //opposite direction of↙
 transfer
        }

        //-----< select all local files >---------------------------------------
        private void LocalSelectAllCheckBox_CheckedChanged(object sender, System.       ↙
EventArgs e)
        {
            bool Checked = true;
            if(LocalSelectAllCheckBox.Checked == true)
            {
                Checked = true;
            }
            else
            {
                Checked = false;
            }
            foreach(ListViewItem lvi in SourceFilesList.Items)
            {
                lvi.Checked = Checked;
            }
        }

        //-----< select all local files >---------------------------------------
        private void RemoteSelectAllCheckBox_CheckedChanged(object sender, System.       ↙
EventArgs e)
        {
            bool Checked = true;
            if(RemoteSelectAllCheckBox.Checked == true)
            {
```

```
                Checked = true;
            }
            else
            {
                Checked = false;
            }
            foreach(ListViewItem lvi in DestinationFilesList.Items)
            {
                lvi.Checked = Checked;
            }
        }

        //-----< transfer files specified by the NodesToUpdate collection >-----
        private void transferFiles(CompareStatus[] NodesToUpdate, bool DirectionLocalToRemote)
        {
            foreach(CompareStatus cs in NodesToUpdate)
            {
                Node n = null;
                bool success=false;
                try
                {
                    n = cs.node();
                    if(DirectionLocalToRemote) //local-->remote
                    {
                        ConsoleTextBox.Text = "Transferring file "+ n._name + "\n"+ ConsoleTextBox.Text;
                        Object[] Args = new Object[]{n._path,(DestinationTextBox.Text+@"\"+n._name)};
                        success = (bool)syncSend("SU.FileTransfer.UploadLocalFile",Args, clone);
                    }
                    else //remote-->local
                    {
                        ConsoleTextBox.Text = "Retrieving file "+ n._name + "\n"+ ConsoleTextBox.Text;
                        Object[] Args = new Object[]{(SourceTextBox.Text+ @"\" + n._name),n._path};
                        success = (bool)syncSend("SU.FileTransfer.DownloadRemoteFile", Args,clone);
                    }
                }
                catch(Exception ex)
                {
                    MessageBox.Show("Error transferring file" + n._path + "to: "
                        + DestinationTextBox.Text+@"\"+n._name + " ("+ex.Message+" )");
                }
            }
            ConsoleTextBox.Text = "Done. \n"+ConsoleTextBox.Text;
        }

        //-----< handler for local refresh button click >-----------------------
        private void LocalRefreshButton_Click(object sender, System.EventArgs e)
        {
            TreeNode tn = LocalTreeView.SelectedNode;
            if( tn == null)
                return;
            else
            {
                Node n = (Node)tn.Tag;
                SourceTextBox.Text = n._path;
                if(!n.isDir())
                    return;
                try
                {
                    Object[] Args = new Object[]{n._path };
```

```
                    Node[] nodes = (Node[])syncSend("SU.FileTransfer.GetLocalDirsFiles", ↙
    Args);
                    LocalExplorer.addChildren(tn,nodes);
                }
                catch(Exception ex)
                {
                    ConsoleTextBox.Text = ex.Message + "\n"+ConsoleTextBox.Text;
                }
            }
        }

        //-----< handler for remote refresh button click >----------------------
        private void RemoteRefreshButton_Click(object sender, System.EventArgs e)
        {
                        TreeNode tn = RemoteTreeView.SelectedNode;
            if( tn == null)
                return;
            else
            {
                Node n = (Node)tn.Tag;
                DestinationTextBox.Text = n._path;
                if(!n.isDir())
                    return;
                try
                {
                    Object[] Args = new Object[]{n._path };
                    if(isValid(clone))
                    {
                        Node[] nodes = (Node[])syncSend("SU.FileTransfer.                ↙
    GetRemoteDirsFiles",Args,clone);
                        RemoteExplorer.addChildren(tn,nodes);
                    }
                }
                catch(Exception ex)
                {
                    ConsoleTextBox.Text = ex.Message + "\n"+ConsoleTextBox.Text;
                }
            }

        }

    }
}
```

```
////////////////////////////////////////////////////////////////
//  TreeViewExplorer.cs Helper class, responsible for         //
//                      managing the dispay and manipulation  //
//                      of nodes in tree-view and list-view   //
//                      controls.                             //
//                                                            //
//  ver 1.0                                                   //
//                                                            //
//  Language:      C#                                         //
//  Platform:      Dell Dimension Pentium 4, Windows 2000     //
//  Course:        CSE997 Master's Thesis                     //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>          //
//                                                            //
////////////////////////////////////////////////////////////////
using System;
using System.Windows.Forms;

namespace FileSynchronizer
{
    ////////////////////////////////////////////////
    /// helper class to manage treeview UI controls
    ////////////////////////////////////////////////
    public class TreeViewExplorer
    {
        TreeView tv;
        ListView lv;

        //-----< constructor >-------------------------------------------------
        public TreeViewExplorer(TreeView _tv, ListView _lv)
        {
            tv = _tv;
            lv = _lv;
        }

        //-----< add a child (directory and files) to the controls >------------
        public void addChildren(TreeNode Parent, Node[] nodes)
        {
            lv.Items.Clear();
            Parent.Nodes.Clear();
            int index=0;
            if(nodes == null)
                return;
            foreach(Node n in nodes)
            {

                if(n.isDir())
                {
                    index = 0;
                    TreeNode tn = new TreeNode(n._name,index,index);
                    tn.Tag = n;
                    Parent.Nodes.Add(tn);
                }
                else
                {
                    ListViewItem lvi = new ListViewItem(n._name);
                    lvi.SubItems.Add(n._path);
                    lvi.Tag = n;
                    lv.Items.Add(lvi);
                }
            }
            Parent.Expand();
        }

        //-----< add initial nodes >-------------------------------------------
        public void addChildren(Node[] nodes)
        {
```

```
            int index=2; //root level additions
            if(nodes == null)
                return;
            foreach(Node n in nodes)
            {
                TreeNode tn = new TreeNode(n._name,index,index);
                tn.Tag = n;
                tv.Nodes.Add(tn);
            }
        }

    }
}
```

```
using System;
using Matrix;

namespace TestCell
{
    class TestCell : Cell
    {
        Wizard Generated Code
        //------------< entry point of execution for each cell >-----------------
        public void start()
        {
        }

        //------------< calls to message-specific processing >-------------------
        public Object process(ExecObject eo)
        {
            return null;
        }

        //------------< constructor >---------------------------------------------
        public TestCell()
        {
            initializeCell();
        }
    }
}
```

```
/////////////////////////////////////////////////////////////////////
//  Synchronizer.cs  Implements a Cell that specifies among        //
//                   other things, the policy used for             //
//                   synchronizing sets of files.                  //
//                   Possesses the capability to handle the        //
//                   following Message Types:                      //
//                   "SU.Synchronizer.Policy.Compare"              //
//                                                                 //
//  ver 1.0                                                        //
//                                                                 //
//  Language:      C#                                              //
//  Platform:      Dell Dimension Pentium 4, Windows 2000          //
//  Course:        CSE997 Master's Thesis                          //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>               //
//                                                                 //
/////////////////////////////////////////////////////////////////////
using System;
using System.Collections;
using Matrix;

namespace Synchronizer
{
    public class Synchronizer : Cell
    {
        #region Wizard Generated Code
        private Guid ID;
        private Messaging messager;
        private BlockingQueue queue;
        private System.Threading.Thread StartThread;
        private System.Collections.Hashtable ResponseList;
        protected System.Collections.Hashtable CapabilityList;
        public event ReceiveDelegate OnReceive;
        private BlockingQueue ResponseQueue;

        //-----------< process messages pulled out of the queue >---------------
        public void extract(object msg)
        {
            ExecObject eo = messager.extract(msg);
            if(eo==null)
                return;
            if(eo.isRequest())
            {
                Object result = process(eo);
                Guid RequestCellGUID;
                String response = messager.buildResponse(msg,result,out RequestCellGUID);
                Matrix.Matrix.sendResponse(response, RequestCellGUID); //SEND RESPONSE
            }
                //Response Received
            else
            {
                ReceiveInfoEventArgs rargs = new ReceiveInfoEventArgs(new Guid(eo.
    getRequestID()), eo.getReturnValue());
                if(OnReceive!=null)
                    OnReceive(this, rargs);
            }
        }


        //-----< process responses out of the response queue >-----------------------
        public void extractResponse(object msg)
        {
            ExecObject eo = messager.extract(msg);
            if(eo==null)
                return;
            if(!eo.isRequest())
            {
```

```
            ReceiveInfoEventArgs rargs = new ReceiveInfoEventArgs(new Guid(eo.
getRequestID()), eo.getReturnValue());
            if(OnReceive!=null)
                OnReceive(this, rargs);
        }
    }

    //-----------< method to register cell in matrix >----------------------
    public bool register()
    {
        Matrix.Matrix.register((Cell)this);
        StartThread = new System.Threading.Thread(new System.Threading.ThreadStart(
start));
        StartThread.Start();
        return true;
    }

    //-----------< method to unregister cell in matrix >--------------------
    public bool unregister()
    {
        Matrix.Matrix.unregister(this);
        StartThread.Abort();
        return true;
    }

    //-----------< accept a request message >-------------------------------------
    public void accept(string message)
    {
        accept(message,false);
    }

    //-----< accept a response message >-----------------------------------------
    public void accept(string message, bool Response)
    {
        Object msg = message;
        if(Response == false)
            queue.enQ(msg);
        else
            ResponseQueue.enQ(msg);
    }


    //-----------< send message to another registered cell >----------------
    public bool send(string message, Guid ID, int argc, Object[] args)
    {
        return true;
    }

    //-----------< get unique cell ID >-------------------------------------
    public Guid getID()
    {
        return ID;
    }

    //-----------< capability interrogation >-------------------------------------
    public bool queryCapability(string capability)
    {
        if(CapabilityList.ContainsKey(capability))
            return true;
        else
            return false;
    }

    //-----------< method to send execution messages to other cells >-------
    public Object syncSend(string MessageName, Object[] Params)
    {
```

```
        string RequestCellGuid = ID.ToString();
        Object Result = null;
        Messaging msg = new Messaging();
        Guid RequestID;
        string xml = msg.buildRequest(RequestCellGuid,MessageName,
            Result, Params, out RequestID, Guid.Empty);
        Matrix.Matrix.transmit(xml);
        SyncWait sync = new SyncWait(this, RequestID);
        Result = sync.wait();
        return Result;
    }

    //-----------< method to send execution messages to clone cells >-------
    public Object syncSend(string MessageName, Object[] Params, Clone clone)
    {
        Guid guid = clone.guid();
        string RequestCellGuid = ID.ToString();
        Object Result = null;
        Messaging msg = new Messaging();
        Guid RequestID;
        string xml = msg.buildRequest(RequestCellGuid,MessageName,
            Result, Params, out RequestID,guid);
        Matrix.Matrix.transmit(xml);
        SyncWait sync = new SyncWait(this, RequestID);
        Result = sync.wait();
        return Result;
    }


    //-----------< method to obtain a Cell clone for stateful operation >---
    public Clone getClone(string MessageName)
    {
        Guid g = Matrix.Matrix.createClone(MessageName);
        if(g.CompareTo(Guid.Empty) == 0)
            throw new Exception("Could not locate cell to handle: "+
                MessageName);
        else
            return new Clone(g);
    }

    //-----< check if a Cell ID is valid >-----------------------------------
    bool isValid(Clone clone)
    {
        Guid CellID = clone.guid();
        if(CellID.CompareTo(Guid.Empty)==0)
            return false;
        else
            return true;
    }

    //-----------<add response received flag to ResponseList >--------------
    public void addResponse(Guid RequestID)
    {
        ResponseList.Add(RequestID, null);
    }

    //-----------<remove response received flag to ResponseList >-----------
    public void removeResponse(Guid RequestID)
    {
        if(ResponseList.Contains(RequestID))
        {
            ResponseList.Remove(RequestID);
        }
    }

    //-----------< retrieve response >-----------------------------------
```

```
        public bool checkAndRetrieveResponse(Guid RequestID)
        {

            if(ResponseList.Contains(RequestID))
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        //-----< initialize cell >-------------------------------------------
        private void initializeCell()
        {
            if(ID.CompareTo(Guid.Empty)==0)
            {
                ID = Guid.NewGuid(); //ID//
            }
            messager = new Messaging();
            queue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extract));
            ResponseQueue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extractResponse));
            CapabilityList = new System.Collections.Hashtable();
            ResponseList = new System.Collections.Hashtable();
        }
        #endregion

        //-----< cell entry point >------------------------------------------
        public void start()
        {
            //TODO: add executive processing if required
            //default: do nothing
        }

        //-----------< calls to message-specific processing >-----------------
        public Object process(ExecObject eo)
        {
            ReturnObject ro = null;
            Exception exception = null;
            Object ret=null;
            try
            {   //
                //TODO:
                //add calls to processing logic here
                string MessageType = eo.getMessageType();
                switch(MessageType)
                {
                    case "SU.Synchronizer.Policy.Compare":
                        ret = process_Synchronizer_Policy_Compare(eo);
                        break;

                }
            }
            catch(Exception ex)
            {
                exception = ex;
            }
            ro = new ReturnObject(ret,exception);
            return ro;
        }

        //-----< method to process SU.Synchronizer.Policy.Compare messages >----------
        private Object process_Synchronizer_Policy_Compare(ExecObject obj)
```

```
        {
            Object[] Params = obj.getParams();
            if(Params.Length!=2)
            {
                throw new Exception("Incorrect Input for Compare message processing");
            }
            Node[] Source = (Node[])Params[0];
            Node[] Destination = (Node[])Params[1];
            CompareStatus[] cs = (CompareStatus[])compare(Source,Destination);
            return cs;
        }

        //-----------< constructor >-------------------------------------------
        public Synchronizer()
        {
            //
            initializeCell();
            //TODO:
            //Add constructor logic here
            CapabilityList.Add("SU.Synchronizer.Policy.Compare", null);
        }

        //-----< compares source & destination, returns nodes needing update >--
        public CompareStatus[] compare(Node[] Source, Node[] Destination)
        {
            ArrayList NodesToUpdate = new ArrayList();
            Hashtable DestinationNodes = new Hashtable(); //for fast lookup
            foreach(Node n in Destination)
                DestinationNodes.Add(n._name,n);
            foreach(Node n in Source)
            {
                if(!(DestinationNodes.ContainsKey(n._name)))
                {
                    //file not present in destination, so copy
                    NodesToUpdate.Add(new CompareStatus(n,CompareStatus.Status.Copy));
                }
                else
                {
                    //if node present in destination, check timestamp
                    int diff = DateTime.Compare(n._LastWriteTimeUtc, ((Node)
    DestinationNodes[n._name])._LastWriteTimeUtc);
                    if(diff >= 0)
                    {
                        //destination node is not newer, so copy/update
                        NodesToUpdate.Add(new CompareStatus(n, CompareStatus.Status.
    Update));
                    }
                    else
                    {
                        NodesToUpdate.Add(new CompareStatus(n,CompareStatus.Status.
    Overwrite));
                    }
                }
            }
            return (CompareStatus[])NodesToUpdate.ToArray(typeof(CompareStatus));
        }
    }
}
```

```
/////////////////////////////////////////////////////////////////
//  Node.cs          Defines helper types to be used in this   //
//                   application.                              //
//  ver 1.0                                                    //
//                                                             //
//  Language:      C#                                         //
//  Platform:      Dell Dimension Pentium 4, Windows 2000     //
//  Course:        CSE997 Master's Thesis                     //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>          //
//                                                             //
/////////////////////////////////////////////////////////////////
using System;
using System.Runtime.Remoting;
//////////////////////////
//// a helper class - Node
//////////////////////////
[Serializable]
public class Node
{
    bool _isDir;
    public string _path;
    public string _name;
    public DateTime _LastWriteTimeUtc;


    //----< does this node refer to a directory >---------------------------
    public bool isDir()
    {
        return _isDir;
    }

    //-----< string representation of time stamp of node >------------------
    public string timeStamp()
    {
        return _LastWriteTimeUtc.ToString();
    }
    //-----< Node class constructor >---------------------------------------
    public Node(string path, bool isDir, DateTime LastWriteTimeUtc, string name)
    {
        _isDir = isDir;
        _path = path;
        _LastWriteTimeUtc = LastWriteTimeUtc;
        _name = name;
    }
}

////////////////////////////////////
//// a helper class - CompareStatus
////////////////////////////////////
[Serializable]
public class CompareStatus
{
    //enumeration of possible compare status values
    public enum Status
    {
        Copy=0,
        Update=1,
        Overwrite=2
    };
    private Status _status;
    private Node n;

    //-----< constructor >--------------------------------------------------
    public CompareStatus(Node _n, Status s)
    {
        n = _n;
```

```
        _status = s;
    }

    //-----< return status >---------------------------------------------
    public Status status()
    {
        return _status;
    }

    //-----< return node >-----------------------------------------------
    public Node node()
    {
        return n;
    }
}
```

```
//////////////////////////////////////////////////////////////////
// FileTransferPeer.cs File implementing a "Cell".              //
//                     Possesses the capability to handle the   //
//                     following Message Types:                 //
//                      "SU.FileTransfer.GetLocalDirsFiles"     //
//                      "SU.FileTransfer.RunServer"             //
//                      "SU.FileTransfer.Connect"               //
//                      "SU.FileTransfer.GetRemoteDirsFiles"    //
//                      "SU.FileTransfer.UploadLocalFile"       //
//                      "SU.FileTransfer.DownloadRemoteFile"    //
//                                                              //
//  ver 1.0                                                     //
//                                                              //
//  Language:      C#                                          //
//  Platform:      Dell Dimension Pentium 4, Windows 2000      //
//  Course:        CSE997 Master's Thesis                      //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>           //
//                                                              //
//////////////////////////////////////////////////////////////////
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
using System.IO;
using Matrix;


namespace FileTransferPeer
{
    ////////////////////////////////////
    ///// file transfer/synchronizer peer
    ////////////////////////////////////
    class FileTransferPeer : Cell
    {
        private FileTransfer.FileTransfer RemoteFileTransfer;
        private FileTransfer.FileSystemTraversal LocalFSTraversal;
        private FileTransfer.FileTransfer LocalFileTransfer;
        private FileTransfer.FileSystemTraversal RemoteFSTraversal;
        #region Wizard Generated Code
        private Guid ID;
        private Messaging messager;
        private BlockingQueue queue;
        private BlockingQueue ResponseQueue;
        private System.Threading.Thread StartThread;
        private System.Collections.Hashtable ResponseList;
        protected System.Collections.Hashtable CapabilityList;
        public event ReceiveDelegate OnReceive;

        //-----------< process messages pulled out of the queue >---------------
        public void extract(object msg)
        {
            ExecObject eo = messager.extract(msg);
            if(eo==null)
                return;
            if(eo.isRequest())
            {
                Object result = process(eo);
                Guid RequestCellGUID;
                String response = messager.buildResponse(msg,result,out RequestCellGUID);
                Matrix.Matrix.send(response, RequestCellGUID);
            }
                //Response Received
            else
            {
                ReceiveInfoEventArgs rargs = new ReceiveInfoEventArgs(new Guid(eo.
    getRequestID()), eo.getReturnValue());
```

```
                if(OnReceive!=null)
                    OnReceive(this, rargs);
            }
        }

        //-----------< method to register cell in matrix >----------------------
        public bool register()
        {
            Matrix.Matrix.register((Cell)this);
            StartThread = new System.Threading.Thread(new System.Threading.ThreadStart(
    start));
            StartThread.Start();
            return true;
        }

        //-----------< method to unregister cell in matrix >--------------------
        public bool unregister()
        {
            Matrix.Matrix.unregister(this);
            StartThread.Abort();
            return true;
        }

        //-----------< accept a request message >-------------------------------------
        public void accept(string message)
        {
            accept(message,false);
        }

        //-----< accept a response message >-----------------------------------------
        public void accept(string message, bool Response)
        {
            Object msg = message;
            if(Response == false)
                queue.enQ(msg);
            else
                ResponseQueue.enQ(msg);
        }


        //-----------< send message to another registered cell >----------------
        public bool send(string message, Guid ID, int argc, Object[] args)
        {
            return true;
        }

        //-----------< get unique cell ID >-------------------------------------
        public Guid getID()
        {
            return ID;
        }

        //-----------< capability interrogation >-------------------------------------
        public bool queryCapability(string capability)
        {
            if(CapabilityList.ContainsKey(capability))
                return true;
            else
                return false;
        }

        //-----------< method to send execution messages to other cells >-------
        public Object syncSend(string MessageName, Object[] Params)
        {
            string RequestCellGuid = ID.ToString();
            Object Result = null;
```

```
        Messaging msg = new Messaging();
        Guid RequestID;
        string xml = msg.buildRequest(RequestCellGuid,MessageName,
            Result, Params, out RequestID, Guid.Empty);
        Matrix.Matrix.transmit(xml);
        SyncWait sync = new SyncWait(this, RequestID);
        Result = sync.wait();
        return Result;
    }

    //-----------< method to send execution messages to clone cells >-------
    public Object syncSend(string MessageName, Object[] Params, Clone clone)
    {
        Guid guid = clone.guid();
        string RequestCellGuid = ID.ToString();
        Object Result = null;
        Messaging msg = new Messaging();
        Guid RequestID;
        string xml = msg.buildRequest(RequestCellGuid,MessageName,
            Result, Params, out RequestID,guid);
        Matrix.Matrix.transmit(xml);
        SyncWait sync = new SyncWait(this, RequestID);
        Result = sync.wait();
        return Result;
    }

    //-----------< method to obtain a Cell clone for stateful operation >---
    public Clone getClone(string MessageName)
    {
        Guid g = Matrix.Matrix.createClone(MessageName);
        if(g.CompareTo(Guid.Empty) == 0)
            throw new Exception("Could not locate cell to handle: "+
                MessageName);
        else
            return new Clone(g);
    }

    //-----< check if a Cell ID is valid >-----------------------------------
    bool isValid(Clone clone)
    {
        Guid CellID = clone.guid();
        if(CellID.CompareTo(Guid.Empty)==0)
            return false;
        else
            return true;
    }

    //-----------<add response received flag to ResponseList >--------------
    public void addResponse(Guid RequestID)
    {
        ResponseList.Add(RequestID, null);
    }

    //-----------<remove response received flag to ResponseList >-----------
    public void removeResponse(Guid RequestID)
    {
        if(ResponseList.Contains(RequestID))
        {
            ResponseList.Remove(RequestID);
        }
    }

    //-----------< retrieve response >---------------------------------------
    public bool checkAndRetrieveResponse(Guid RequestID)
    {
```

```
            if(ResponseList.Contains(RequestID))
            {
                return true;
            }
            else
            {
                return false;
            }
        }

        //-----< initialize cell >-------------------------------------------------
        private void initializeCell()
        {
            if(ID.CompareTo(Guid.Empty)==0)
            {
                ID = Guid.NewGuid(); //ID//
            }
            messager = new Messaging();
            queue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extract));
            ResponseQueue = new BlockingQueue(
                new BlockingQueue.DequeueDelegate(extract));
            CapabilityList = new System.Collections.Hashtable();
            ResponseList = new System.Collections.Hashtable();
        }
        #endregion

        //-----< cell entry point >------------------------------------------------
        public void start()
        {
            //TODO: add executive processing if required
            //default: do nothing
        }
        //-----------< calls to message-specific processing >--------------------
        public Object process(ExecObject eo)
        {
            ReturnObject ro = null;
            Exception exception = null;
            Object ret=null;
            try
            {   //
                //TODO:
                //add calls to processing logic here
                string MessageType = eo.getMessageType();
                switch(MessageType)
                {
                    case "SU.FileTransfer.GetLocalDirsFiles":
                        ret = process_FileTransfer_GetLocalDirsFiles(eo);
                        break;
                    case "SU.FileTransfer.GetRemoteDirsFiles":
                        ret = process_FileTransfer_GetRemoteDirsFiles(eo);
                        break;
                    case "SU.FileTransfer.RunServer":
                        ret = process_FileTransfer_RunServer(eo);
                        break;
                    case "SU.FileTransfer.Connect":
                        ret = process_FileTransfer_Connect(eo);
                        break;
                    case "SU.FileTransfer.UploadLocalFile":
                        ret = process_FileTransfer_UploadLocalFile(eo);
                        break;
                    case "SU.FileTransfer.DownloadRemoteFile":
                        ret = process_FileTransfer_DownloadRemoteFile(eo);
                        break;
                }
            }
```

```
            catch(Exception ex)
            {
                exception = ex;
            }
            ro = new ReturnObject(ret,exception);
            return ro;
        }


        //-----------< method to process GetLocalFiles messages >-----------------------
        public Object process_FileTransfer_GetLocalDirsFiles(ExecObject obj)
        {
            Object[] Params = obj.getParams();
            if(Params.Length>2)
            {
                throw new Exception("Incorrect Input for GetLocalDirsFiles message processing");
            }
            else if(Params.Length == 0)
            {
                Object Nodes = LocalFSTraversal.getDirsFiles();
                return Nodes;
            }
            else
            {
                Object Nodes = LocalFSTraversal.getDirsFiles((string)Params[0]);
                return Nodes;
            }
        }

        //-----------< method to process GetRemoteFiles messages >-----------------------
        public Object process_FileTransfer_GetRemoteDirsFiles(ExecObject obj)
        {
            Object[] Params = obj.getParams();
            if(Params.Length>2)
            {
                throw new Exception("Incorrect Input for GetRemoteDirsFiles message processing");
            }
            else if(Params.Length == 0)
            {
                Object Nodes = RemoteFSTraversal.getDirsFiles();
                return Nodes;
            }
            else
            {
                Object Nodes = RemoteFSTraversal.getDirsFiles((string)Params[0]);
                return Nodes;
            }
        }

        //-----------< method to process Connect messages >-----------------------
        public Object process_FileTransfer_Connect(ExecObject obj)
        {
            Object[] Params = obj.getParams();
            if(Params.Length!=2)
            {
                throw new Exception("Incorrect Input for Connect message processing");
            }
            bool success = connect((string)Params[0],(int)Params[1]);
            return success;
        }

        //-----------< method to process RunServer messages >-----------------------
        public Object process_FileTransfer_RunServer(ExecObject obj)
        {
```

```
            bool success = false;
            Object[] Params = obj.getParams();
            if(Params.Length!=1)
            {
                throw new Exception("Incorrect Input for RunServer message processing");
            }
            else
            {
                int ListeningPort = (int)Params[0];
                success = initServer(ListeningPort);
            }
            return success;
        }

        //-----------< method to process UploadLocalFile messages >----------------------
        public Object process_FileTransfer_UploadLocalFile(ExecObject obj)
        {
            Object[] Params = obj.getParams();
            if(Params.Length!=2)
            {
                throw new Exception("Incorrect Input for UploadLocalFile message processing");
            }
            bool success = sendLocalFile((string)Params[0],(string)Params[1]);
            return success;
        }

        //-----------< method to process DownloadRemoteFile messages >-------------------
        public Object process_FileTransfer_DownloadRemoteFile(ExecObject obj)
        {
            Object[] Params = obj.getParams();
            if(Params.Length!=2)
            {
                throw new Exception("Incorrect Input for DownloadRemoteFile message processing");
            }
            bool success = getRemoteFile((string)Params[0],(string)Params[1]);
            return success;
        }

        //-----------< constructor >-----------------------------------------
        public FileTransferPeer()
        {
            //
            initializeCell();
            //TODO:
            //Add constructor logic here
            LocalFSTraversal = new FileTransfer.FileSystemTraversal();
            LocalFileTransfer = new FileTransfer.FileTransfer();
            CapabilityList.Add("SU.FileTransfer.GetLocalDirsFiles",null);
            CapabilityList.Add("SU.FileTransfer.RunServer",null);
            CapabilityList.Add("SU.FileTransfer.Connect",null);
            CapabilityList.Add("SU.FileTransfer.GetRemoteDirsFiles",null);
            CapabilityList.Add("SU.FileTransfer.UploadLocalFile",null);
            CapabilityList.Add("SU.FileTransfer.DownloadRemoteFile",null);
        }


        //-----< init client >-------------------------------------------------
        public bool initClient(string uri)
        {
            BinaryClientFormatterSinkProvider cp = new BinaryClientFormatterSinkProvider();
            BinaryServerFormatterSinkProvider sp = new BinaryServerFormatterSinkProvider
```

```
();
        sp.TypeFilterLevel = System.Runtime.Serialization.Formatters.TypeFilterLevel.
Full;
        System.Collections.IDictionary props = new System.Collections.Hashtable();
        props["typeFilterLevel"] = System.Runtime.Serialization.Formatters.
TypeFilterLevel.Full;
        props["name"] = "client";
        TcpChannel ClientChannel = new TcpChannel(props,cp,sp);
        ChannelServices.RegisterChannel(ClientChannel);
        return true;
    }

    //-----< initialize server >-----------------------------------------
    public bool initServer(int port)
    {
        BinaryClientFormatterSinkProvider cp = new BinaryClientFormatterSinkProvider
();
        BinaryServerFormatterSinkProvider sp = new BinaryServerFormatterSinkProvider
();
        sp.TypeFilterLevel = System.Runtime.Serialization.Formatters.TypeFilterLevel.
Full;
        System.Collections.IDictionary props = new System.Collections.Hashtable();
        props["typeFilterLevel"] = System.Runtime.Serialization.Formatters.
TypeFilterLevel.Full;
        props["port"] = port;
        props["name"] = "server";
        TcpChannel ServerChannel = new TcpChannel(props,cp,sp);
        ChannelServices.RegisterChannel(ServerChannel);
        RemotingConfiguration.RegisterWellKnownServiceType(typeof(FileTransfer.
FileTransfer),
            "FileTransfer",WellKnownObjectMode.SingleCall);
        RemotingConfiguration.RegisterWellKnownServiceType(typeof(FileTransfer.
FileSystemTraversal),
            "Synchronizer",WellKnownObjectMode.SingleCall);
        return true;
    }

    //-----< connect to remote file server/peer >----------------------------
    public bool connect(string RemoteMachine, int RemotePort)
    {

        bool success = initClient("tcp://" + RemoteMachine + ":" + RemotePort + "/
FileTransfer");
        //remote instance
        this.RemoteFileTransfer = (FileTransfer.FileTransfer)Activator.GetObject(
typeof(FileTransfer.FileTransfer),
            "tcp://" + RemoteMachine + ":" + RemotePort + "/FileTransfer");
        this.RemoteFSTraversal = (FileTransfer.FileSystemTraversal)Activator.
GetObject(typeof(FileTransfer.FileSystemTraversal),
                            "tcp://" + RemoteMachine + ":" + RemotePort + "/
Synchronizer");
        return success;
    }

    //-----< send a local file to the remotely connected machine >----------
    public bool sendLocalFile(string SourcePath_FileName,
        string DestinationPath_FileName)
    {
        FileStream fs = new FileStream(SourcePath_FileName,FileMode.Open);
        BinaryReader br = new BinaryReader(fs);
        byte[] buffer = br.ReadBytes((int)fs.Length);
        fs.Close(); br.Close();
        return RemoteFileTransfer.writeFile(buffer,DestinationPath_FileName);
    }

    //-----< get a file from the remotely connected machine >--------------
```

```
        public bool getRemoteFile(string Local_FileName,
            string Remote_FileName)
        {
            byte[] buffer = RemoteFileTransfer.readFile(Remote_FileName);
            FileStream fs = new FileStream(Local_FileName,FileMode.Create);
            BinaryWriter bw = new BinaryWriter(fs);
            bw.Write(buffer);
            bw.Close(); fs.Close();
            return true;
        }
    }
}
```

```
////////////////////////////////////////////////////////////////
//  FileTransfer.cs  Implements functionality to transfer     //
//                   files to and fro between remote and local //
//                   machines                                  //
//  ver 1.0                                                    //
//                                                             //
//  Language:      C#                                         //
//  Platform:      Dell Dimension Pentium 4, Windows 2000     //
//  Course:        CSE997 Master's Thesis                     //
//  Author:        Riddhiman Ghosh <rdghosh@syr.edu>          //
//                                                             //
////////////////////////////////////////////////////////////////
using System;
using System.Runtime.Remoting;
using System.IO;
using System.Collections;

namespace FileTransfer
{
    ////////////////////////////////////////////
    /// class to upload download files
    ////////////////////////////////////////////
    public class FileTransfer : MarshalByRefObject
    {
        //-----< write a byte stream to a file >--------------------------------
        public bool writeFile(byte[] FileData, string DestinationPath_FileName)
        {
            FileStream fs = new FileStream(DestinationPath_FileName, FileMode.Create);
            BinaryWriter bw = new BinaryWriter(fs);
            bw.Write(FileData);
            bw.Close(); fs.Close();
            //Console.WriteLine("Written file: " + DestinationPath_FileName);
            return true;
        }

        //-----< read from a file into a byte stream >--------------------------
        public byte[] readFile(string FileName)
        {
            FileStream fs = new FileStream(FileName, FileMode.Open);
            BinaryReader br = new BinaryReader(fs);
            byte[] buffer = br.ReadBytes((int)fs.Length);
            fs.Close(); br.Close();
            return buffer;
        }
    }

    ////////////////////////////////////////////////////
    /// class for file system traversal
    ////////////////////////////////////////////////////
    public class FileSystemTraversal : MarshalByRefObject
    {
        string SourceDirectory=null;
        ArrayList Contents = null;

        public FileSystemTraversal()
        {
            SourceDirectory = "";
            Contents = new ArrayList();
        }

        //-----< set the search directory >-------------------------------------
        public void setSearchDirectory(string SearchDirectory)
        {
            SourceDirectory = SearchDirectory;
        }
```

```
        public FileSystemTraversal(string _SourceDirectory)
        {
            SourceDirectory = _SourceDirectory;
            Contents = new ArrayList();
        }

        //-----< get list of logical drives on the system >---------------------------
        public Node[] getDirsFiles()
        {
            ArrayList Contents = new ArrayList();
            foreach(string s in Directory.GetLogicalDrives())
            {
                FileSystemInfo fi = new DirectoryInfo(s);
                Node n = new Node(fi.FullName, true, fi.LastWriteTimeUtc, fi.Name);
                Contents.Add(n);
                //Console.WriteLine(fi.FullName);
            }
            return (Node[])Contents.ToArray(typeof(Node));
        }

        //-----< get contents of a directory upto one level down >--------------------
        public Node[] getDirsFiles(string DirectoryPath)
        {
            ArrayList DirContents = new ArrayList();
            DirectoryInfo di = new DirectoryInfo(DirectoryPath);
            if(di.Exists == false)
            {
                Console.WriteLine("Cannot find: " + DirectoryPath);
                return null;
            }
            bool isDir=false;
            foreach( FileSystemInfo finfo in di.GetFileSystemInfos())
            {
                if( (finfo.Attributes & FileAttributes.Directory) == FileAttributes.
    Directory)
                    isDir = true;
                else
                    isDir = false;
                Node n = new Node(finfo.FullName,isDir, finfo.LastWriteTimeUtc, finfo.
    Name);
                DirContents.Add(n);
                //Console.WriteLine(finfo.FullName);
            }
            return (Node[]) DirContents.ToArray(typeof(Node));
        }

        //-----< scan path for files and directories >--------------------------
        public Node[] scan(string SourceDirectory)
        {
            System.IO.FileSystemInfo fi= new DirectoryInfo(SourceDirectory);
            if(fi.Exists == false)
            {
                throw new Exception("Could not resolve given directory path");
            }
            Node n = new Node(fi.FullName, true, fi.LastWriteTimeUtc, fi.Name);
            traverse(n);
            return (Node[])Contents.ToArray(typeof(Node));
        }

        //-----< helper function to traverse directory structure >--------------
        private void traverse(Node n)
        {
            if(n.isDir())
            {
                Contents.Add(n);
                //Console.WriteLine(n._path);
```

```
                DirectoryInfo di = new DirectoryInfo(n._path);
                foreach( FileSystemInfo finfo in di.GetFileSystemInfos())
                {
                    bool isDir = false;
                    if((finfo.Attributes & FileAttributes.Directory) == FileAttributes.
    Directory)
                    {
                        isDir = true;
                    }
                    traverse(new Node(finfo.FullName,isDir, finfo.LastWriteTimeUtc, finfo
    .Name));
                }
            }
            else
            {
                Contents.Add(new Node(n._path, false, n._LastWriteTimeUtc, n._name));
                //Contents.Add(fi);
                //Console.WriteLine(n._path);
            }
        }

        //-----< print out Contents >-----------------------------------------------
        public void printContents()
        {
            foreach(FileSystemInfo fi in Contents)
            {
                if((fi.Attributes & FileAttributes.Directory) == FileAttributes.Directory
    )
                {
                    Console.WriteLine("D: " + fi.FullName);
                    Console.WriteLine(fi.LastWriteTime.ToString());
                }
                else
                {
                    Console.WriteLine(fi.FullName);
                    Console.WriteLine(fi.LastWriteTime.ToString());
                }
            }
        }
    }
}
```