

March 2012

Environment Cockpit

Qiu Chen

Worcester Polytechnic Institute

Follow this and additional works at: <https://digitalcommons.wpi.edu/mqp-all>

Repository Citation

Chen, Q. (2012). *Environment Cockpit*. Retrieved from <https://digitalcommons.wpi.edu/mqp-all/2021>

This Unrestricted is brought to you for free and open access by the Major Qualifying Projects at Digital WPI. It has been accepted for inclusion in Major Qualifying Projects (All Years) by an authorized administrator of Digital WPI. For more information, please contact digitalwpi@wpi.edu.

ENVIRONMENT COCKPIT

A Major Qualifying Project Report
submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the Degree of
Bachelor of Science by

Qiu CHEN Ruoqing FU
Zhen HE Siqi WANG
3/14/2012

This report represents the work of one or more WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its web site without editorial or peer review.

Abstract

The Environment Cockpit was designed for the BNP Paribas eCommerce technical support team: just as pilots use a cockpit to control an aircraft, our Environment Cockpit is a tool to help visualize and control computing facilities. Specifically, it serves to centralize information from existing tools and to provide a user-friendly and dynamic graphical representation of different elements of the system to help diagnose alerts accurately and effectively. This project defined the scope of the tool, implemented a project prototype with WPF and Graph# library, designed a user interface model and constructed a prototype implementation.

Authorship

The main part of this report was developed during B term 2011 by Qiu Chen and Ruoqing Fu. During C term 2012, Qiu Chen worked with Zhen He and Siqi Wang to write Appendix D, which was done to meet the graduation requirements for their respective double majors.

Table of Contents

INTRODUCTION.....	6
BACKGROUND	11
THE GENERAL CONCEPT OF A SERVER	11
THE GENERAL CONCEPT OF A CLIENT	11
THE CONCEPT OF DEPENDENCY BETWEEN PROCESSES	12
THE WIRE.....	12
DISCOVERY SERVICE	15
WPF	17
MVVM DATA BINDING.....	17
THE IDEA OF AN ‘ABSTRACT’ ITEM.....	19
EXISTING APPLICATIONS AND SERVICES IN THE ECOMMERCE ENVIRONMENT	21
<i>Heartbeat</i>	21
<i>BMC and MQ</i>	21
<i>ITRS</i>	25
<i>SAM</i>	26
GRAPH THEORY IN THE COCKPIT.....	26
AGILE DEVELOPMENT	27
LAYOUT ALGORITHM	28
DEVELOPMENT.....	30
CHALLENGE	30
<i>The scale of Environment Cockpit</i>	30
<i>Difficulties</i>	31
Current Difficulties Supporting the Environment	31
Display Difficulties	32
Technical difficulties	33
Implementation difficulties	34
STRATEGY	35
<i>Initial design ideas</i>	35
<i>Graphing tool comparison</i>	37
OUTCOME.....	39
PROJECT ARCHITECTURE DIAGRAM	39
PROJECT PROTOTYPE	40
PROTOTYPE FUNCTIONALITIES SUMMARY.....	44
<i>The tab view</i>	44
<i>Abstract items and data flow representations</i>	45
<i>Hierarchical relationships representations</i>	47
<i>Logical view and physical view filter</i>	48
<i>Adding and deleting different types of connections</i>	50
<i>Importing and exporting graph files</i>	52
<i>Detailed information and log files display</i>	52
<i>Zoom box</i>	53
<i>Double click vertices</i>	53

USER STORIES	53
<i>User story I – tracing alerts</i>	53
<i>User story II – adding connections</i>	56
<i>User Story III – Deleting Connections</i>	56
<i>User story IV – Exploring real data</i>	57
COCKPIT PROTOTYPE IMPLEMENTATION STRATEGY	58
CODING ANALYSIS.....	58
<i>Two open source libraries</i>	58
<i>Three self-implemented libraries</i>	59
<i>One test package</i>	59
<i>Two builds</i>	60
<i>The Little Cockpit GUI code analysis</i>	60
THE ENVIRONMENT COCKPIT GUI	62
THE ENVIRONMENT COCKPIT PROOF OF CONCEPT	65
METHODOLOGY	70
CONCLUSION.....	73
ACKNOWLEDGEMENTS	75
REFERENCES.....	76
APPENDIX	80
APPENDIX A.1 – COCKPIT PROTOTYPE CODE	80
<i>CockpitService.cs</i>	80
<i>CockpitServerStartupConsole.cs</i>	82
<i>EdgeControl.xaml</i>	83
<i>ClientTest.cs</i>	89
<i>Brushes.xaml</i>	90
<i>DesignerItem.xaml</i>	91
<i>Connector.cs</i>	90
<i>EdgeRouteToPathConverter.cs</i>	91
<i>Singleton.cs</i>	92
APPENDIX A.2 – SERIALIZED COCKPIT ITEM	93
APPENDIX B – ITEMS LIST AND ATTRIBUTES	99
APPENDIX C – TIMELINE	101
APPENDIX D – STRUCTURED FINANCE	103
Figure 1: Sam	7
Figure 2: Proteus monitoring alerts.....	7
Figure 3 Block diagram of the Environment Cockpit (Sunai Patel, Project Description)	10
Figure 4: Wire discovery service flow chart (The Wire).....	15
Figure 5: Discovery Illustration flow chart (The Wire.)	16
Figure 6: MVC diagram	18
Figure 7: Transmitting deal data from EFX to FXO (Storey).....	22
Figure 8: Deal information data flow (Storey)	23
Figure 9: Snapshot of deal data transmitting in BMC I (Storey)	24
Figure 10: Snapshot of deal data transmitting in BMC II (Storey)	25

Figure 11: Sam working diagram (Sam.)	26
Figure 12: EFX flow environment display	31
Figure 13 Logical view and physical view combined	33
Figure 14 the Bubble View	35
Figure 16 Grid View	36
Figure 15 Grid View	36
Figure 17 The Environment Cockpit architecture diagram.....	40
Figure 18 Wire service Window.....	41
Figure 19 Cockpit Prototype snapshot I	42
Figure 20 Tab view demonstration.....	44
Figure 21 The cockpit prototype snapshot II	45
Figure 22The cockpit prototype snapshot III.....	45
Figure 23 Cockpit Prototype snapshot IV	47
Figure 24 Logical view filter	49
Figure 25 Physical view filter—location view	49
Figure 26 Connection Type.....	50
Figure 27 Connectors.....	50
Figure 28 Cockpit Prototype Snapshot V.....	51
Figure 29 Connection pop up box	51
Figure 30 Warning message box	51
Figure 31 Load and Save button	52
Figure 32 Log Information	52
Figure 33 Detailed Information	52
Figure 34 Zoom Box.....	53
Figure 35.....	54
Figure 36 Logical View	55
Figure 37 Overall view	55
Figure 38 Physical View	55
Figure 39 Adding Connection	56
Figure 40 Delete Connection	56
Figure 41 Group View of Monza.....	57
Figure 42 Gfit Processes and Servers.....	57
Figure 43 UX team Gui Design I.....	63
Figure 44 UX team Gui Design II.....	64
Figure 45 UX team Gui Design III.....	65

Introduction

BNP Paribas, headquartered in Paris, is one of the biggest banking groups in the world. The bank can be viewed from both the retail aspect and the investment aspect. The eCommerce Group is a crucial part of investment banking at BNP Paribas because it is chiefly in charge of foreign exchange trading.

In order to support the trading events within the eCommerce group, the eCommerce Application Production Support Group (APS Group) functions to resolve alerts raised by applications, fix non-functioning servers and broken data flow, and maintain all eCommerce machines, applications and processes.

Currently, the eCommerce APS group utilizes many small applications to monitor the department environment. These small applications include Sam (Figure 1), a relatively new application used for starting and stopping processes, and Proteus(Figure 2), one of the most widely used applications by the APS group supporters to monitor alerts. Each of these applications has its own purpose, yet displays a small portion of the eCommerce environment, so it is difficult for the APS group supporters to understand the status of the general health condition of the environment. Therefore, it would be very convenient to pull all the environment data and information together and display the entire system in one application. In this way, the APS group supporters can easily view the system and quickly respond to the environment alerts.

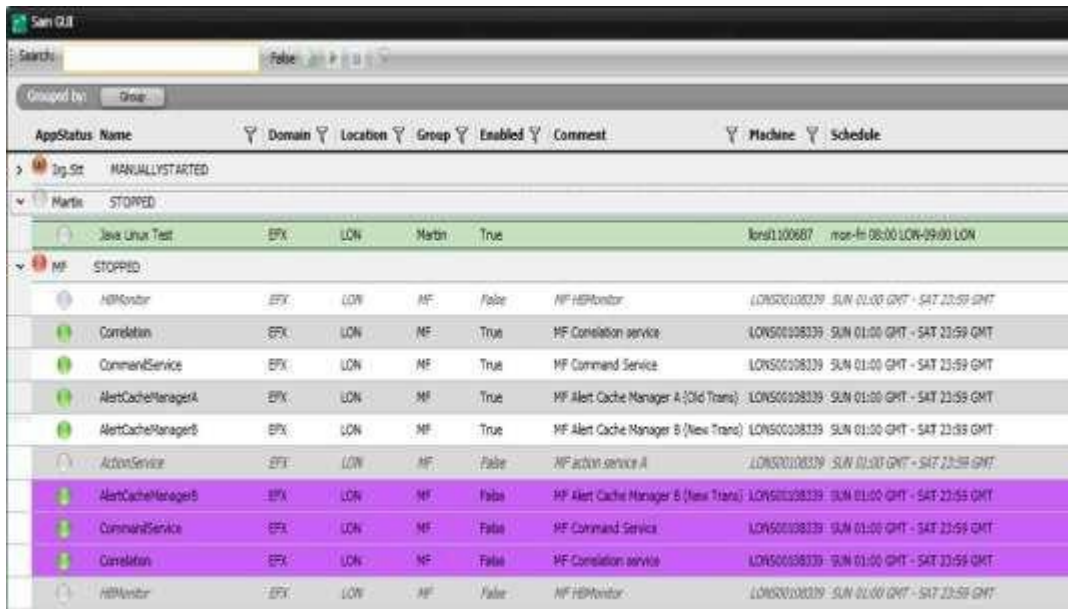


Figure 1: Sam

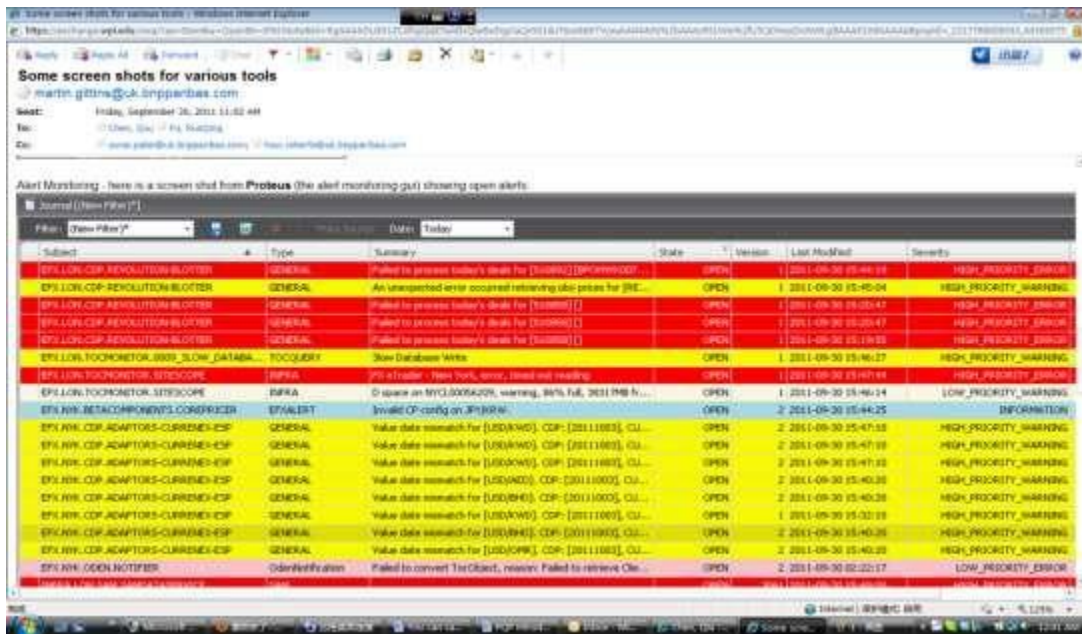


Figure 2: Proteus monitoring alerts

The need for a central application to monitor many small applications brought about the idea of – environment cockpit proposed by Wells Powell, the head of

eCommerce Technology Group. Our project consisted of carrying out the first stage of Environment Cockpit development with the eCommerce architecture team.

The first stage of the cockpit development includes collecting user requirements, developing the cockpit prototype and designing a user interface. After meetings and discussions with the eCommerce APS supporters and the architecture group, we concluded that the user interface should be a dynamic and graphic representation of the environment. Functionality requirements were adjusted daily to meet the rising business need.

Throughout a timeframe of seven weeks, we actively communicated with different stakeholders and produced the definition, functionalities summary and development strategies of Environment Cockpit. By definition, the Environment Cockpit is a support monitoring tool that can visually present system information and manage the entire environment for the eCommerce group. It borrows the idea from the plane cockpit, where pilots sit to control the whole aircraft. Similarly, for APS supporters, the environment cockpit graphically displays a large number of environment elements in use, both production and non-production, to allow easier management and diagnostics. With this solution, the APS supporters will be able to view the overall environment and efficiently control and manage the system.

The primary goal of the cockpit is to show infrastructure setup and ultimately give users control to manage the environment. The first phase, which is to show infrastructure setup, includes showing which processes are running, where the processes are located, communication and dependencies between processes, alerts being raised, log information,

server statistics and links to other internal and external environments. By centralizing all of this information from existing tools, system usage can be monitored more conveniently and effectively to achieve the maximum utilization. Moreover, developers can quickly diagnose issues by viewing alerts from the Environment Cockpit and identify the source of the alert by using just one application rather than looking into many different ones. On the second phase, the Environment Cockpit may potentially allow control of the environment, such as having the ability to start and stop processes or move processes between servers. In order to move a process from one server to another, the APS supporters have to switch between applications. The Environment Cockpit will potentially allow users to drag and drop processes between servers, which will reduce the number of steps to be taken. This kind of control will significantly improve the efficiency of supporters. If the Environment Cockpit is proven to be successful in the future, other teams in the bank could apply its design idea, user interface and all the related technologies to their system maintenance and usage control management.

Considering the complex nature of the project itself, implementing such an Environment Cockpit application will take years of effort. Given the project timeframe of seven weeks, we decided to implement a prototype to explore and research different possibilities of the GUI design and implementation strategies. We then worked with the User Experience (UX team) to develop a mature user interface design. Based on the prototype and GUI design, we proposed a proof of concept of how the Environment Cockpit can be implemented in the future.

The diagram shown below demonstrates the simplified Environment Cockpit dataflow. From right to left, server information flows to the existing control agents and

monitoring tools. Server information and traffic information (alerts, heartbeats) flows to the Environment Cockpit represented by the control box on the left

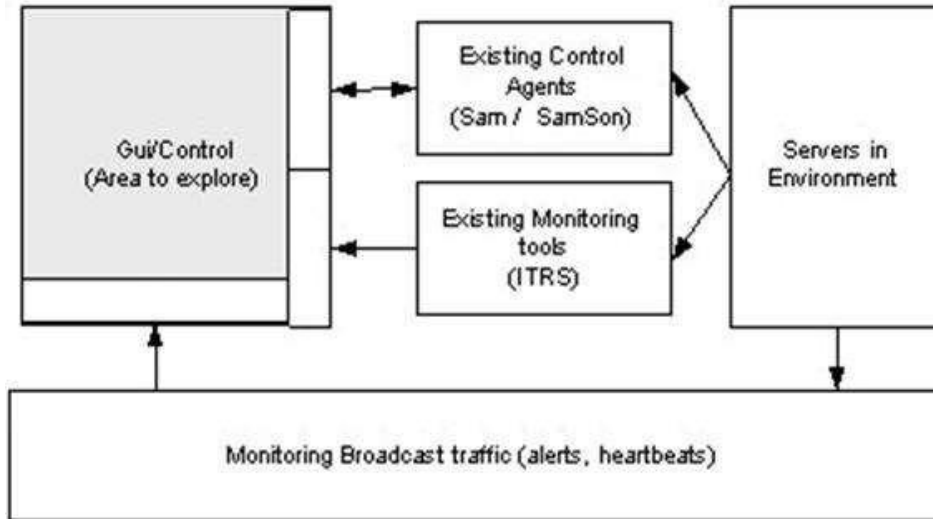


Figure 3 Block diagram of the Environment Cockpit (Sunai Patel, Project Description)

Background

The general concept of a server

In the Environment Cockpit project, servers have two different meanings. The two meanings are distinct and cannot be confused with each other. First, servers can refer to the machines that are used in the bank for processing applications such as Bloomberg, SAM and TOC. The Environment Cockpit displays the status of these servers. Second, servers can also represent the server component in the environment cockpit client-server architecture diagram (Figure 16). In the Environment Cockpit, they are the crucial machines that are running to collect the environment data from plug-ins and dealing with the requests of the clients.

The general concept of a client

A client is an application or a system that accesses a service made available by a server. A client could be connected to a server via network remotely or by inter-process communication techniques on the same machine. One way of connection applies to devices that are not capable of running their own programs but communicate with computers by way of network. The other way of connection is founded on the client-server model that client and server can run on the same machine and connect through Unix domain sockets, shared memory, named pipes or other inter communication methods (Client). The Environment Cockpit was developed by the first means. Various

sources of information about the production environment are gathered up into a couple of central servers, and then distributed to the Cockpit GUI.

The concept of dependency between processes

When one process cannot occur until another process is completed, there is a dependency between the two processes. Theoretically, process dependency come in two forms—resource dependency and data dependency. Resources dependency means –several segments cannot be executed in parallel if they aren't sufficient processing resources and data dependency means –data modified by one segment must not be modified by another parallel segment. (Borysowich)

When one process produces or modifies some tangible resource or data that is used by another process, the affected process cannot proceed until the prior process completes modification. For example, the CDP service contains streaming services that include pricing data service while 360T is an application that depends on the pricing data service for further display and computation.

The wire

The wire is a set of libraries that are developed internally by BNP Paribas about half a year ago. It was defined as –a set of components designed to enable client-server and server-server communication on BNP Paribas Wiki page. Both server and client

machines can send message between each other by utilizing the wire library, which can be accessed in C#, Java and C++ language. There are three key components in the wire.

- Data Object - the definition of the message sent between servers and clients.
- Client - the component to make requests for the message.
- Server - the component to service requests for the message

Data Object:

All the information and data in the message have to be encrypted in a protocol buffer in order to be transmitted through the wire. The protocol buffer was developed by Google and defined as –a language-neutral, platform-neutral, extensible way of serializing structured data for use in communications protocols, data storage, and morell (Developer Guide). The protocol buffer was implemented as a standard for free transmission between multiple platforms and applications in the bank. Developers need to define the structure of the message being sent in a file ending with .proto. The file extension follows not only the protocol buffer syntax rules on the Google Code, but also a set of stricter rules of naming and syntax required by the bank. All the .proto files have to be maintained in the definition folder of the DCTV repository in SVN with a rigid folder structure. Each folder inside the definition folder maps to a .NET binary file in the output folder. After developers saves .proto files in SVN, the protocol buffer definition will be automatically converted to an equivalent source code in C#, Java and C++. This compilation can be realized either locally or remotely in SVN, which usually takes about 20 to 30 minutes. C# and java binary code can then be generated from the output file. By referencing the generated binary libraries in the wire, developers can directly use setter

and getter methods that are automatically generated through the compilation to manipulate all the fields defined within the protocol buffer.

Client:

A wire client instantiates either a request or a subscription and sends it to a wire service. The client references the protocol buffer definition structure in the request message and will wait for responses from the service after. The difference between a request and a subscription is that a request will only receive a single synchronous response from a server immediately while a subscription set up can receive multiple asynchronous responses from a server whose results will be streamed as they are created. Before setting up a wire client, the wire environment needs to be configured correctly.

Server:

Each wire service endpoint is defined by its environment and location. A client has two ways to connect to a wire service endpoint. One way is simply to connect to the target service by using its hostname and its port number; the other way is to use the wire discovery service by setting up a unique service identifier and then having the client request the service identifier. Then, the wire discovery service will provide service location information according to the service identifier to the client. The second way is a better mechanism for its simple server migration.

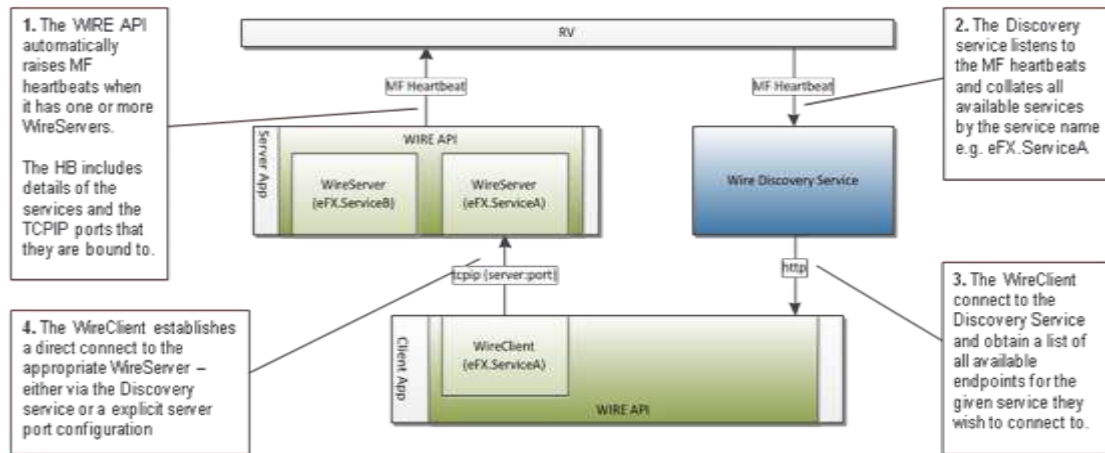


Figure 4: Wire discovery service flow chart (The Wire)

Discovery service

Before getting into the discussion of discovery service, it would be necessary to introduce a few concepts.

Application: –a logical name for a deployable component that runs as one or more processes on one or more hosts. || (eCommerce)

Process: –a separate unit of execution that can be started and stopped by OS commands. || (eCommerce)

Service: –It consists of a named collection of Protocol Buffer message types, that are interpreted as inputs, and is associated with a specified Endpoint. Services with the same name are assumed to implement the same functionality. For simple cases it's perfectly acceptable for an Application name and Service name to be the same. || (eCommerce)

Endpoint: –a Hostname and port number in the TCP implementation, Other wire implementation are possible and their Endpoints may differ. (eCommerce)

MF Heartbeat:” an MF message on RV that includes a subject, the final part of the subject being the **Application.** (The Wire.)

Discovery service identifies applications if they generate the appropriate MF heartbeats.

As indicated by the graph shown below, a service provider transmits the information through heartbeat. Then the discovery service makes the endpoint information available to let users view the current service end points. Afterwards, the discovery service publishes the information onto service data RV for the other instances in the other regions to discover (The Wire.).

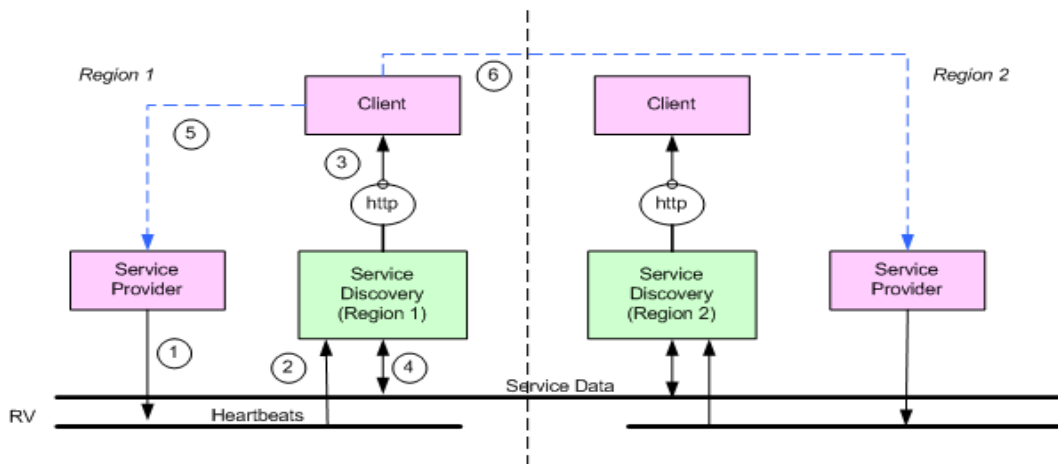


Figure 5: Discovery Illustration flow chart (The Wire.)

WPF

WPF, short for Windows presentation Foundation, is a computer-software graphical subsystem to make the user interface. It was developed by Microsoft and released as a part of .NET Framework. XAML scripts are used in WPF for defining and linking various UI elements. In WPF, users can define the look of an element directly or with the internal templates and styles indirectly. The styles can be composed by a bunch of property settings on different types of templates provided in WPF, such as the control template and the data template that we use a lot for our project (Windows Presentation Foundation.).

MVVM data binding

Model-View-ViewModel (MVVM) design pattern is a widely spread design pattern within the software world recently. It was originally developed by John Gossman in 2005, based on the idea of a very classic design pattern called MVC (Model View Controller). Model-View-Controller pattern contains the View (what you see on the screen), the Model (the data displays on the screen) and the Controller (the component that hooks the view and the model together). The figure below shows the relationship between these three components graphically. (Bucanek)

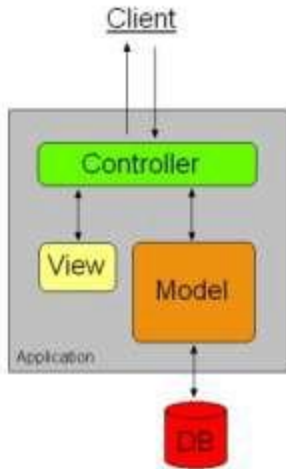


Figure 6: MVC diagram

This pattern above enables the isolation of the application logic from the user interface. Developers who are specialized in the interface design and the backend implementation are able to develop in a rather independent and simultaneous way. The loose coupling of developing user interface and backend will also create a smooth designer and developer workflow and then allow efficient coding process.

The difference between MVVM pattern and the classic MVC pattern is that MVVM pattern replaces the Controller with the ViewModel component. The ViewModel's advantage over Controller is that it not only serves as a data binding between the view and the model, but also is an abstraction of the view. It's a specialized aspect of the Controller that exposes public properties and abstractions. However the discussion of their differences is still an ongoing area as the MVVM pattern is getting more standardized.

There is a fundamental connection between MVVM and WPF. To simplify the creation of the user interface, MVVM model is introduced as a standardized way to leverage somecore features of WPF, which as we mentioned before, contains XAML files for the view and .Net files for the Model and the Viewmodel. WPF is well suited to MVVM pattern. Most importantly, by binding the View to the ViewModel, the loose coupling provided by WPF entirely removes the need for writing codes in the ViewModel that directly updates a View. Other useful features include the data templates which can apply Views to the ViewModel objects shown in the user interface and resource system that can automatically locate and apply the templates. The ViewModel classes are easy to unit test too. The testability of the ViewModel can assist in properly designing the user interface because developers can write unit tests for the ViewModel without actually creating any UI object.

The idea of an 'abstract' Item

In the environment, there are different types of components that the APS team monitors, such as servers, processes, bubbles and so on. In order to represent these different types of environment elements, it is very useful to develop the idea of an abstract item for the Environment Cockpit first. In the cockpit, an item is a unit of data that we have to manage, such as a process, a computer, a data center, a sub-net, a component, a group, a domain, a suite or a database (Roberts). All the items can be assembled into item hierarchies. For example, the processes running on a server that is in a data center or the processes that are streaming in an application within a group. The

types of hierarchy will be system generated from the API plug in applications, which local users have no right to change them. The relationships between items can also be defined in other ways, such as process connections and data flows between processes. All types of connections between items are rooted in process connection. For instance, when two servers are connected to each other, it is actually the processes running on each server that transmit information between each other, rather than the servers themselves. The concept of process connection is very important because it reflects the dependency between each item.

Each item has its unique id, which is used for the environment identification or hierarchy inferring, a type and a collection of item attributes associated with it (Roberts). For example, servers have attributes called CPU usage or disk size. Not all item attributes are fixed. Some of them can be updated over time from the company's real system data (Roberts). With the concept of an '_abstract' item, Environment Cockpit can transmit information between server and clients. The wire just needs to send a collection of items without knowing what type of information it is transmitting. A list of abstract items and their attributes was summarized and attached in Appendix B - a list of abstract items and attributes.

Existing applications and services in the eCommerce environment

The information that the Environment Cockpit GUI displays is entirely gathered from the API plug in applications, including Autopilot, BlackbirdMonitor, SamGUI, StarGazer, TOCAAdmin, Cab Admin, ITRS, RMDS, Syslog, RV, Proteus and so on (Patel). RV is a messaging framework and Proteus is the most widely used alerts monitoring tool. Below list a few other important applications that can potentially provide information for the Environment Cockpit.

Heartbeat

A heartbeat is a broadcast to application monitoring tools about the life and death of services. There are many kinds of heartbeats, like MF heartbeat and wire heartbeat. MF heartbeat is used to help the discovery service identify the applications. Although a Wire instance may support several services, only one heartbeat needs to be sent. Heartbeat usually includes the application name and the specific services that are included in the body of the message. MF heartbeat is broadcast on RV whereas a wire heartbeat is a point to point TCP/IP message between the client and the server (Heartbeat).

BMC and MQ

BMC is a MQ monitoring tool, known as Queuepassa in the bank. MQ is a queue component to transmit data between different environments and systems. All the data put in MQ will for sure reach its destination sooner or later. BMC is mainly used by infrastructure team to check the MQ data transferring status and speed in the environment. For example, if an alert is raised by the high latency of the data transferring

process that used MQ component, such as from EFX to FXT, supporters can then explore the specific MQ queue to check the work flow processing status, speed, data flow size and other important information on BMC GUI. There are two types of MQ transmitting. One is inter-application data transferring. Below is a graph to illustrate this type of transfer. The graph shows an example of transmitting deal data from EFX to FXO through MQ directly without duplicating the data (Storey).

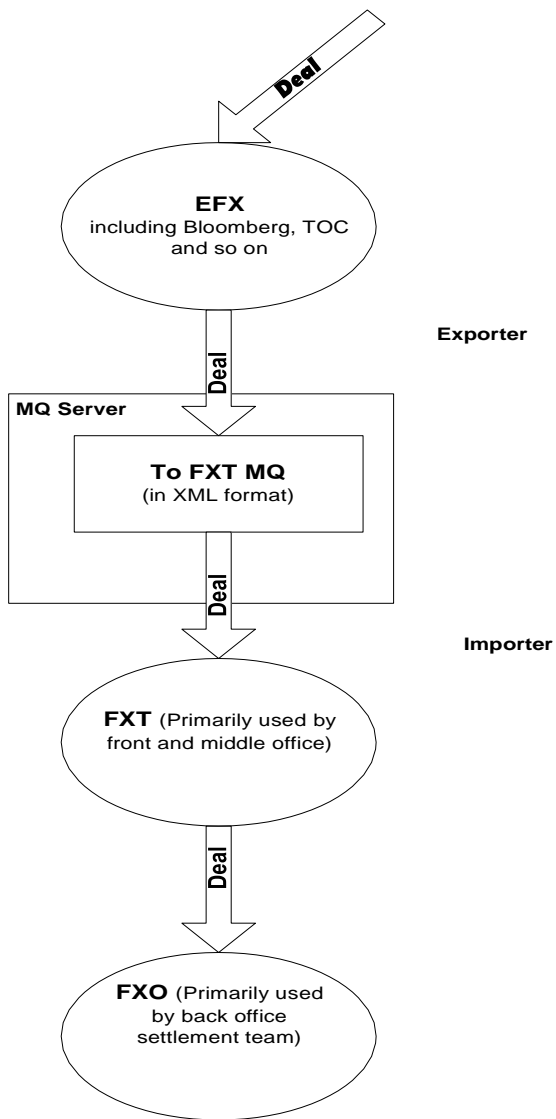


Figure 7: Transmitting deal data from EFX to FXO (Storey)

The second type of MQ data transmitting is called intra-application data transferring. This involves duplicating data and distributing the information to different places. Below is a graph to illustrate the data flow. As you can see, the deal data gets duplicated and is transferred by MQ from FXT London to FXT New York, FXT Singapore and FXT Tokyo (Storey).

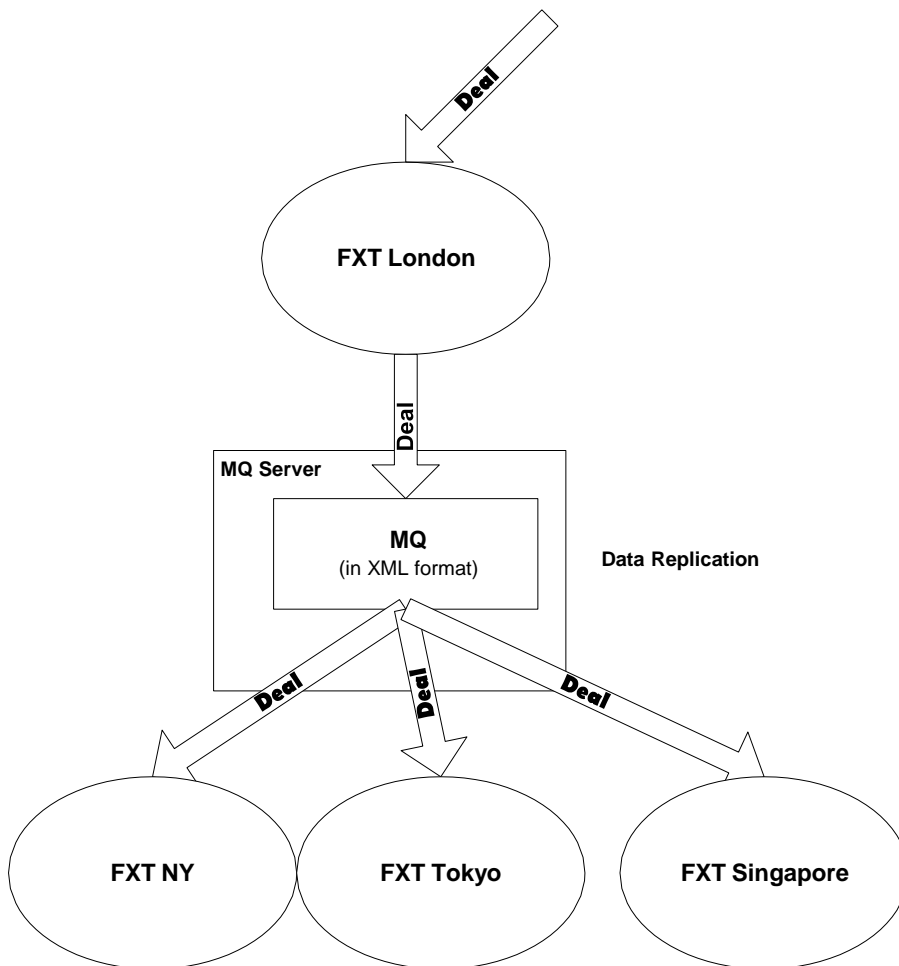


Figure 8: Deal information data flow (Storey)

BMC has another functionality of monitoring transaction process. People can see the specific timing of when a data flow get transmitted from one source to the other

source. However this functionality has not been used very often by supporters yet (Storey).

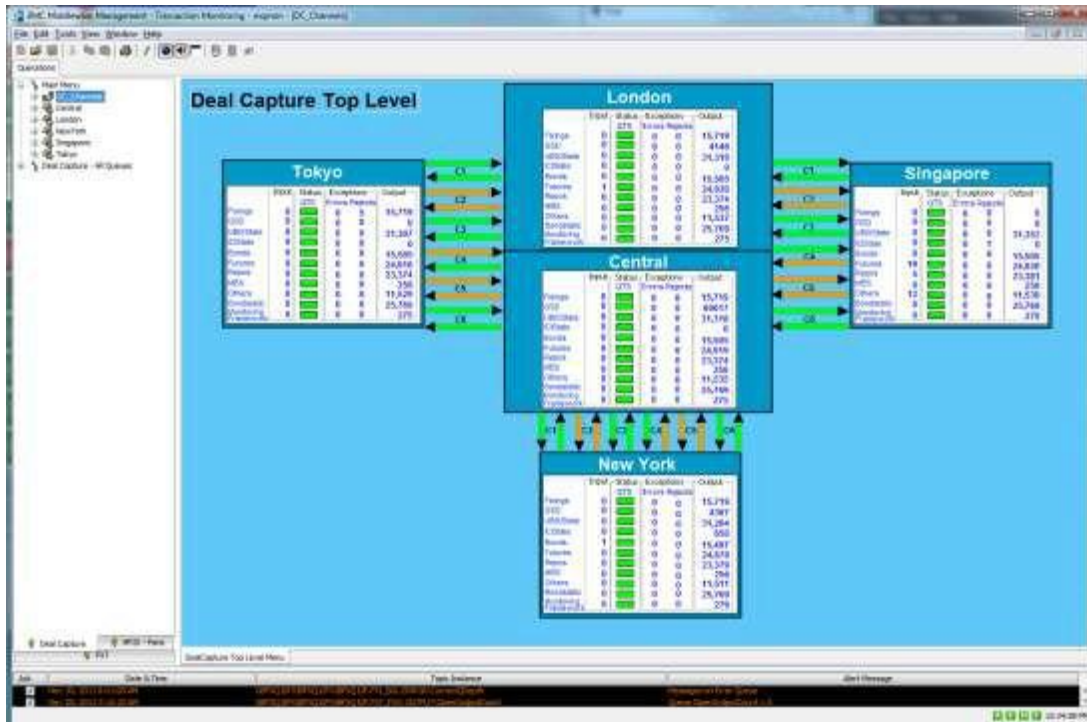


Figure 9: Snapshot of deal data transmitting in BMC I (Storey)

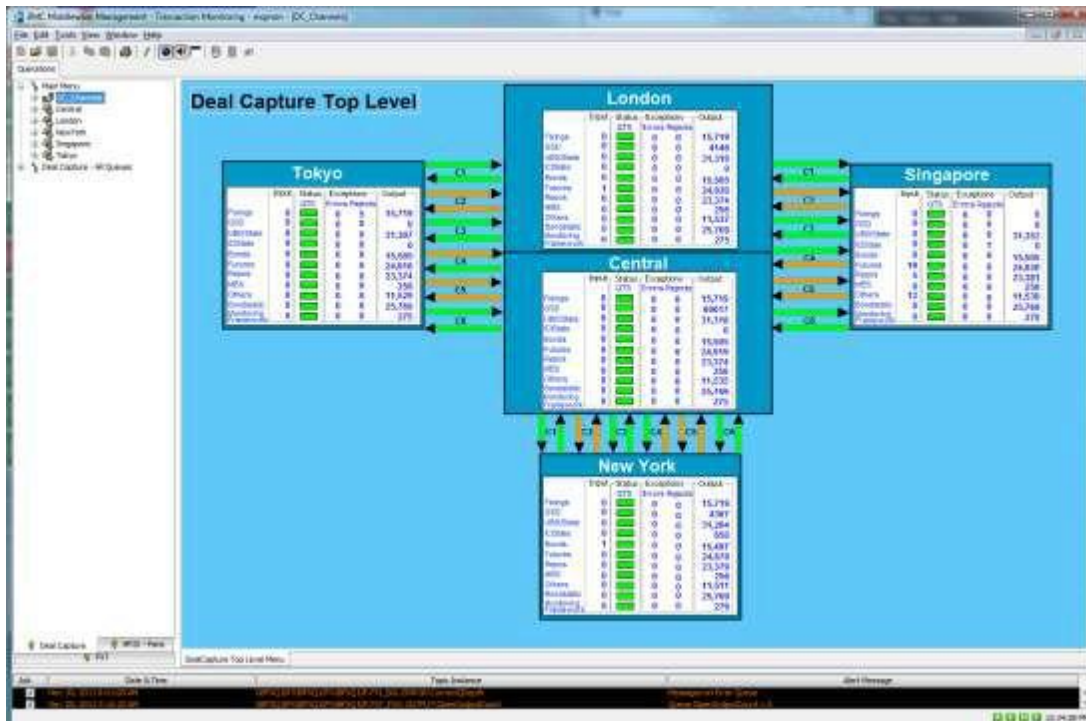


Figure 10: Snapshot of deal data transmitting in BMC II (Storey)

ITRS

ITRS, a third party environment monitoring tool, is a very powerful application and was already used by several teams within the bank. ITRS provides both hardware and applications information. Furthermore, it allows users to customize a dashboard overview of graphically displaying the environment elements that the users are concerned about. The data on the dashboard overview will also update in the real time. This functionality seems to be very similar to the Environment Cockpit requirements. However after the team interacted with Sebastien Dubuisson, an expert on ITRS from Market Data Team, we discovered that there is a very long learning curve of ITRS and in addition, the budget is also another huge concern for the eCommerce APS group to adopt ITRS in a short period of term (Dubuisson).

SAM

SAM stands for Service Agent Manager to manage and monitor server-side eCommerce processes. Sam provides a database of process descriptors which describe the processes that run on different machines, a controlling process that manages the processes schedule, a GUI for viewing and manipulating process descriptors, and an agent process - SamSon that runs on each server to manage the individual process lifecycle (Sam.).

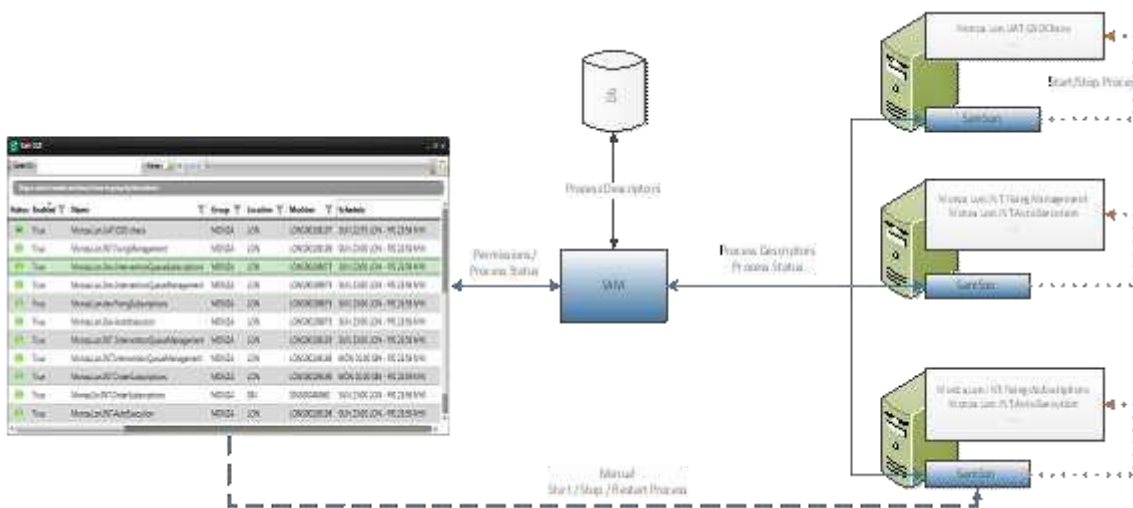


Figure 11: Sam working diagram (Sam.)

Graph theory in the cockpit

Dots connected by lines comprise a graph. A -dot|| is called a vertex. A -line|| is called an edge. The connecting edges between vertices indicate the relationship between all the items. The way that dots and lines are presented makes up the layout of a graph (West). In the Cockpit GUI solution, we defined a few basic classes - PocVertex,

PocEdge and PocGraph. In the PocVertex class, each item is assigned with an unique ID. These unique IDs are identified in PocEdge for the connection use. When users move the mouse over an item, four sticking rectangles that belong to the connector class will show up. Each rectangle has a vertex inherited from PocVertex associated with it. When we say connecting two items, it is actually the vertices encompassed in the rectangles that are connected to each other.

Agile development

Agile Development is an umbrella term for several iterative and incremental software development methodologies such as Extreme Programming, Scrum and Lean Development. Though different methods have their own approaches, they all share a common rule of incorporating iteration and continuous feedbacks to develop a software system. They all involve continuous planning, continuous testing, continuous integration, and other forms of continuous evolution of both the project and the software. As important, they all focus on empowering people to collaborate and make decisions together quickly and effectively. Agile methods break the whole projects into discrete tasks and these tasks involve a software development cycle, including planning, requirements analysis, design, coding, unit testing, and acceptance testing when a working product is demonstrated to stakeholders (Martin). However, in the case of our project, we worked separately in different locations. Though we didn't really follow the agile method rules that emphasize more face-to-face communications in the same office because of the physical constraint, we had daily discussions with sponsors to make sure

that we were following the requirements of the users. Other than the daily conference meetings with our sponsors, we also had our own discussions to assign daily independent tasks and facilitate team spirit. The advantage of agile methods is to minimize the overall risk and make the project more adaptable to customer requests quickly.

Layout algorithm

There are two types of layout algorithm commonly used. They are force-directed algorithm and parallel algorithm. Force-directed algorithm is often used to draw general graphs, with the goal of finding the minimum energy to represent the most aesthetically pleasing drawings. This algorithm employs partitioning of the spatial domain, clever initial positioning of vertices and multi-level approaches (Fruchterman and Reingold). Parallel layout algorithm involves three approaches involving multi-level force-directed graph layout algorithm, parallel simulated annealing algorithm, and graph and visualization on display walls (Brent and Kung). The Environment Cockpit prototype made use of force-directed algorithm that the graph# library brought in. However, with this type of algorithm, the prototype encountered some technical difficulties that users have to rearrange the layout to get a clear view. This resulted from the compound view mechanism built on the prototype. Simply put, graph# restricted in computing local and global attractive and repulsive forces between all the compound layers. Thus far, the vertex positions could not be updated as accurately at once to reach an equilibrium state that is to have all the attractive and repulsive forces between nodes balanced. Consequently, nodes that are supposed to be attracted to each other may act repulsively

on the graph, resulting in an unpleasant visual effect. This can be where the future Environment Cockpit improves on.

Development

Challenge

The scale of Environment Cockpit

In order to understand the purpose of the Environment Cockpit, it is necessary to depict the environment numerically. EFX Flow Environment is used here as an example for simple illustration. In this environment, there are more than 80 running machines across 4 different regions, around 225 unique process applications, 450 processes and numerous technologies. These technologies include two operating systems—Windows and Linux, four programming languages — C#, C++, Java and Python and about 20 different middleware such as RMDS, MQ, RV, LQ2 and FQP. Additionally, as the picture indicates, the complex data flow between each service collection makes it very difficult for users to understand the environment situation at a glance.

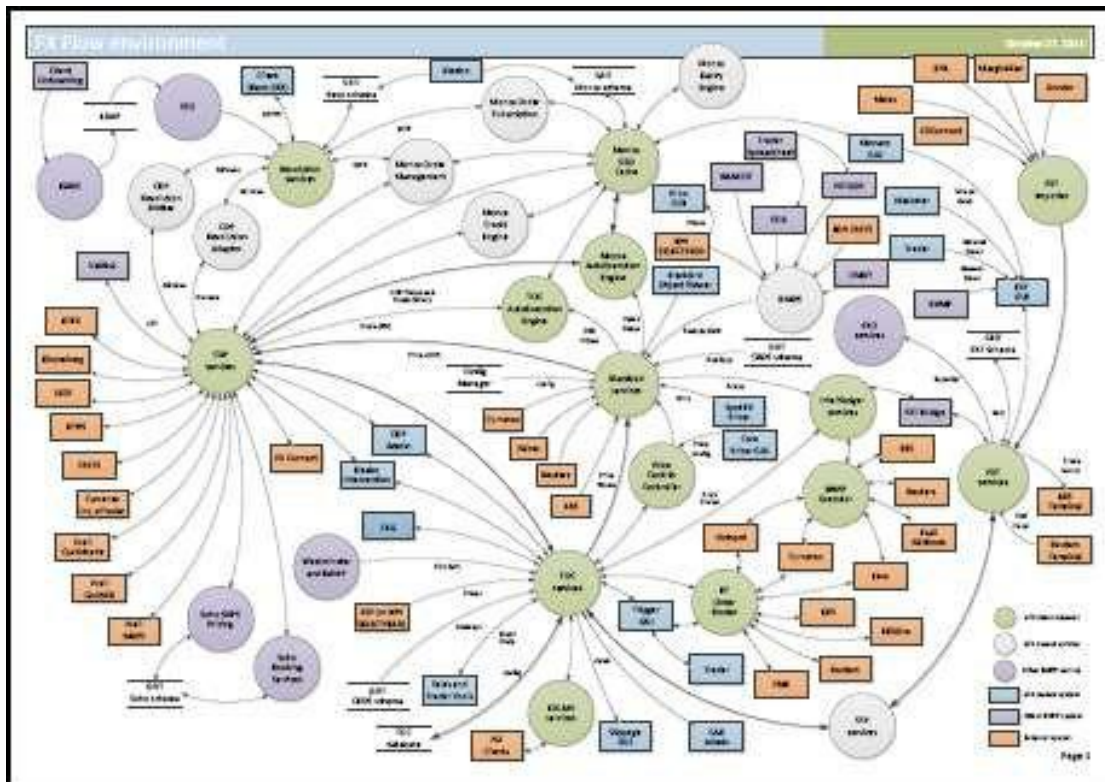


Figure 12: EFX flow environment display

Difficulties

Current Difficulties Supporting the Environment

First, the eCommerce APS supporters are currently using multiple applications to manage the system, such as SAM, TOC and Proteus. These existing monitoring applications all provide different perspectives to the environment and, therefore, take up much of the screen. When all the applications cannot fit into a supporter’s screen, he or she needs to switch between applications, which decreases his or her efficiency.

Second, there is no easy way to view the overall health of applications and services. Although the APS supporters have various monitoring tools to view different

perspectives of the environment, there is no straightforward way for them to view the overall system.

Third, there is no easy way to view information about where the servers are located, overall servers' health and where firewalls are located. Also, the relationship between processes is not displayed by any existing environment monitoring tool. Although experienced supporters are familiar with the data flow and relationships within the system, Environment Cockpit will increase their efficiency and provide a representation of data flow. For new employees, it will provide a central point from which they can learn.

Display Difficulties

The environment can be displayed in two different perspectives. The first view is logical view, which displays the logical hierarchy relationship within the system, for example, processes located in different service groups and service groups located in different domains. The second view is the physical view, which demonstrates the physical hierarchy of relationships. This includes processes located in different servers, servers located in different zones, and zones located in different locations. In order to show the most of the environment hierarchical relationships, the Environment Cockpit combines both physical view and logical view together and displays both perspectives in one graph. The graph in figure 13 is an example of combining physical and logical view.

The physical view on this graph displays servers in different locations and processes working on different servers. The logical view displays processes belonging to different applications and the data flow between processes. This is only a very small portion of the environment. There are actually more than 500 processes within the eFX flow environment in the bank. There is information for all processes within physical and logical view, therefore it is a difficult feat to display all this information clearly.

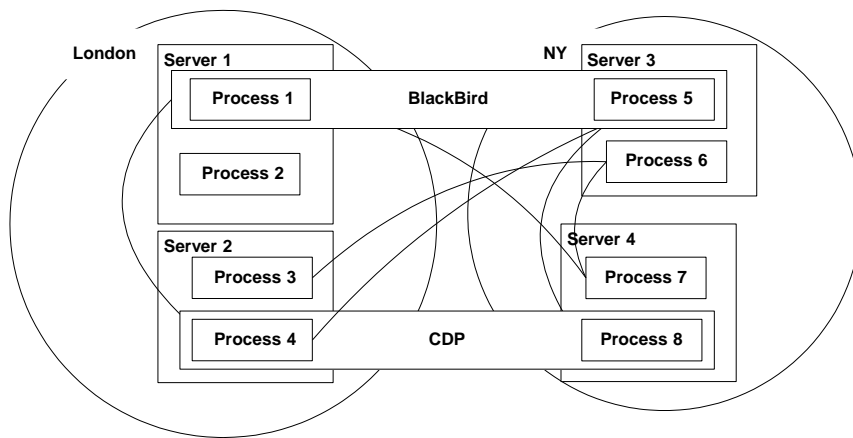


Figure 13 Logical view and physical view combined

Technical difficulties

First, it is not possible to retrieve all the logical relationships between processes from the system, especially for database connections. For example, the program, electro, is using an SQL query to get information from the database and there is currently no way for the system to detect when the program is actually accessing the database. This leads to the impossibility for the Environment Cockpit to create and remove the connections on

its GUI accordingly. Another example is that currently there is a list of wire service end points available within the bank, but there is no mechanism to get information about which clients or applications are actually connecting to which wire services.

Second, there are currently no plug-ins developed to get information from the Wire, Sam, TOC and other applications, so developers don't know what information they will bring to Environment Cockpit, what the format of the data is and how difficult it is to integrate all the data.

Implementation difficulties

There are numerous monitoring applications out there to assist the APS supporters with system management. All of these monitoring applications depict the system in different ways, but there is currently no single application supporters can use exclusively to gather all environment information. The Environment cockpit was designed to compensate for this deficiency. Its most distinct feature is to centralize and integrate all environment data collected from different existing tools such as, RMDS, RV, MQ, SAM, SAMSON, and TOC. The integration of data from different resources makes the Environment Cockpit more challenging to implement than the other monitoring tools.

Since the Environment Cockpit deals with a large amount of environment data, the maintenance is difficult. To resolve maintenance issues, the Environment Cockpit will utilize automation where appropriate. For example, when any team in the bank

deploys a new process, it will automatically be informed and will display the update on the GUI.

Strategy

Initial design ideas

The bubble view was one of the original ideas for the Environment Cockpit proposed by Wells Powell. It was designed to display different elements in the environment. As the

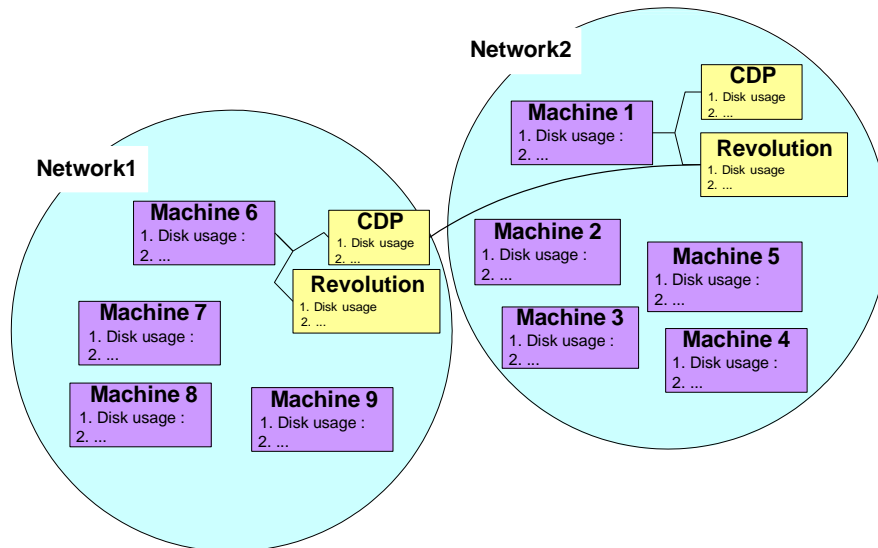


Figure 14 the Bubble View

Figure 14 indicates, each bubble represents a network and within each network. Users can view a list of machines with service, data flow, disk usage and other information related to each machines.

The grid view was another original design idea of the Environment Cockpit proposed by Huw Roberts. Similar to the bubble view, the grid view was also designed to show all the elements in the environment but in a different way. As shown below in Figure 15, the outside rectangles represent environment locations such as London and Tokyo. The second largest rectangles divide the environment in different locations by service groups such as CDP and Wibble. Within each service group, it contains many booking processes. The colors red, yellow and green indicate how busy the processes are. The color red, for instance, means that the process is handling a relatively large number of bookings per second and indicates that the process is overloaded. The advantage of this view is that it can fit the largest amount of information with the smallest space used and shows the health condition of all the processes.

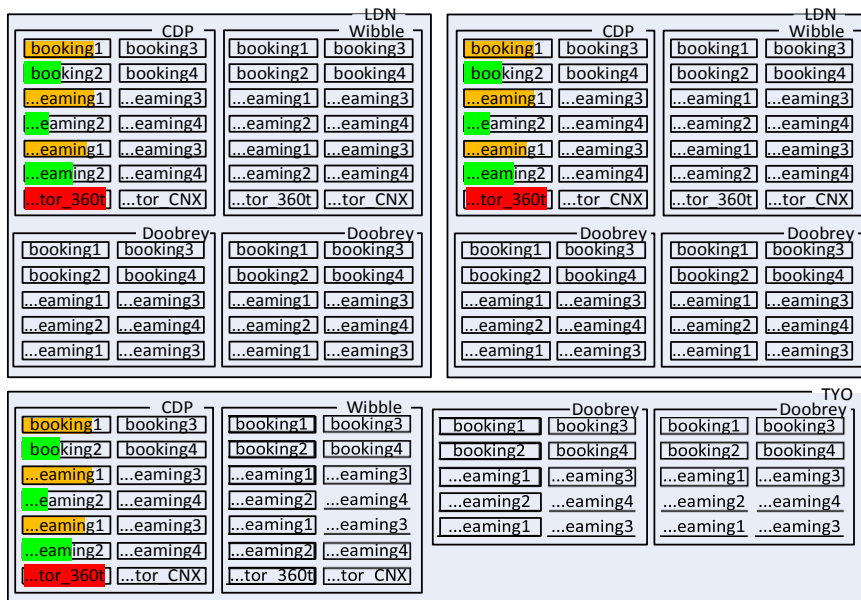


Figure 15 Grid View

Graphing tool comparison

Below are the comparison results between different graphing tools after detailed investigation into each of them

Graphing Tools	Advantages	Disadvantages
Nshape	<ul style="list-style-type: none"> • able to drag items on the graph • able to create template • able to customize the size of the view • able to generate XML format of image files that suit the project needs 	<ul style="list-style-type: none"> • only 10 basic shapes • very complicated to create shapes
Visio	<ul style="list-style-type: none"> • able to select templates • look very professional • easy to add graph 	<ul style="list-style-type: none"> • not possible to develop a WPF or .Net based application
Ywork	<ul style="list-style-type: none"> • plenty of documentation and tutorial • super nice automatic layout • able to present overview and detailed view at the same time 	<ul style="list-style-type: none"> • too expensive to buy the license

	<ul style="list-style-type: none"> • Convenient users control such as expanding or collapsing nodes and hovering over effects 	
QuickGraph	<ul style="list-style-type: none"> • able to add shapes dynamically • simple to code with 	<ul style="list-style-type: none"> • not enough documentation or tutorial
Graphviz	<ul style="list-style-type: none"> • open source • searching algorithms • using Dot text language to define a graph • flexible and fancy views • Reliable 	<ul style="list-style-type: none"> • impossible to be directly used without a viewer
Graph#	<ul style="list-style-type: none"> • WPF based • relatively more documentation • completely free • able to customize features 	<ul style="list-style-type: none"> • not enough documentation and tutorial

Table 1: Graph Tool Comparison

Outcome

Project architecture diagram

To start with, it is important to demonstrate project architecture diagram and to understand the information that feeds into the Environment Cockpit. As indicated by the architecture diagram below, information travels a long way before it reaches the cockpit. The blue boxes on the top of the graph are the existing applications that BNP ecommerce team is using to analyze the environment. For example, HB provides process lifetime information, ITRS provides hardware monitoring information, SAM provides processes and services general information, and ADDM manages hardware specification information. Because all these applications present information in different ways, they have to be unified and managed on a common user interface that converts all kinds of information display techniques into one that servers can identify. The API Plug in plays the role of being the common interface, and it is a very important component in passing all the information to the server. After the server gets the information, the data goes through the wire environment that is used to transmit information between client and server. Ultimately, the cockpit receives, organizes and displays the information on the GUI.

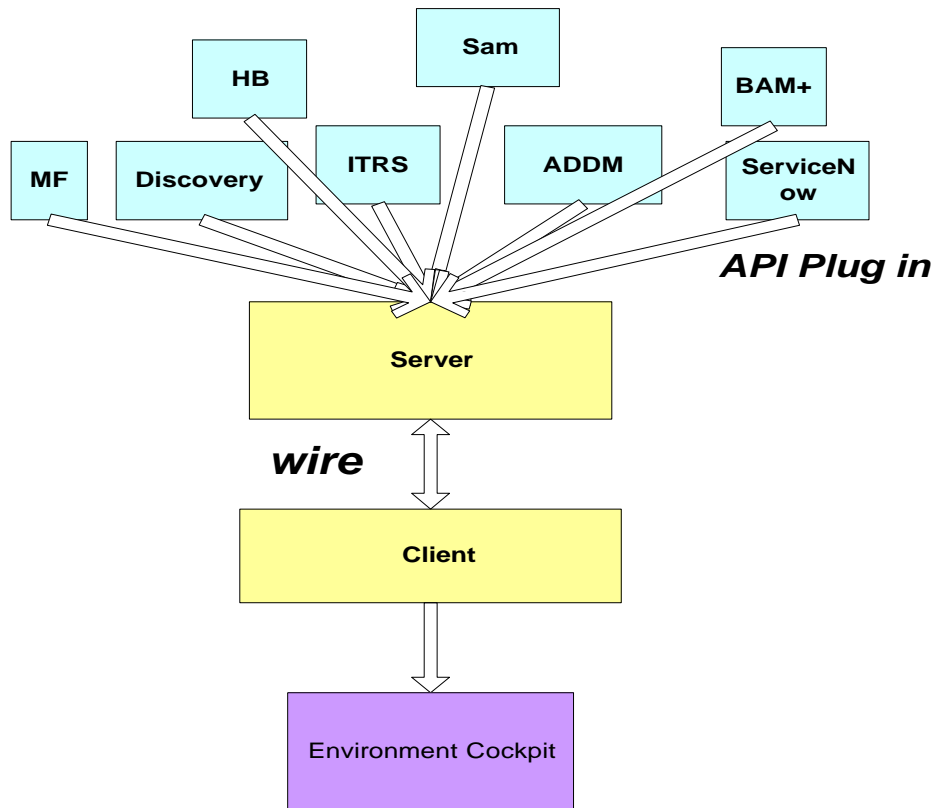


Figure 17 The Environment Cockpit architecture diagram

Project prototype

As the report stated previously, the banking environment is extremely large and complex with a large number of applications, technologies, processes, servers and others statistics concerned. Given the seven weeks of the project timeframe, it is improbable to develop an application that incorporates all the environment information and meets the entire functionality requirements. Therefore, the team decided to build a prototype to explore different aspects of environment cockpit GUI designs, and investigate a few important implementation technologies as a start up practice for the future development. Based on the architecture diagram and the original design ideas proposed by the BNP ecommerce groups, we summarized a list of functionalities that were intended to be

implemented on the Environment Cockpit and these functionalities will potentially revolutionize the way of how the existing monitoring tools manage the environment. These new functionalities include alerting mechanism, data flow representation, hierarchy relationship, logical and physical view filter, adding and deleting connections, saving and loading files and displaying detailed information of different elements in the environment.

In the Environment Cockpit, we successfully configured the wire environment, established both server

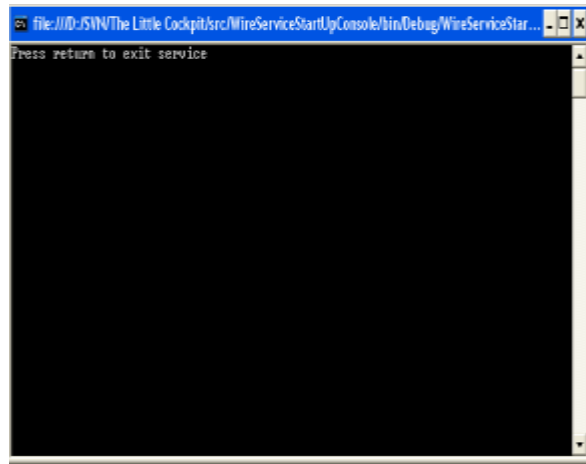


Figure 18 Wire service Window

and client sides, and enabled the communication between the server and the client. The cockpit prototype users need to make sure that they start the cockpit wire service before they can run the cockpit prototype user interface program. The window in figure 17 shows up after the wire service is established successfully. The client side is embedded and used as a libraries reference to the prototype user interface development. When users run the cockpit GUI, the cockpit client subscribes to the cockpit discovery service automatically, and thus enables the communication between the client and the server.



Figure 19 Cockpit Prototype snapshot I

Figure 18 is the cockpit prototype GUI which consists of two tabs. One is the GFit processes and servers view with the real data generated from the Gfit database while the other is the Processes Collection View with dummy data that the team defined. We used dummy data for two reasons. One is the time constraints of the seven weeks period, which makes it impossible to put changes into production to collect all the environment information from the existing monitoring tools; the other is that even if we get the information from the production, there is still the technical difficulty of integrating all the collected information into one universal presenting format that the cockpit server can recognize and take in. Because of these reasons, we implemented the user interface with the dummy data to illustrate the useful functionalities that the real environment cockpit can possibly have in the future.

Prototype functionalities summary

The tab view

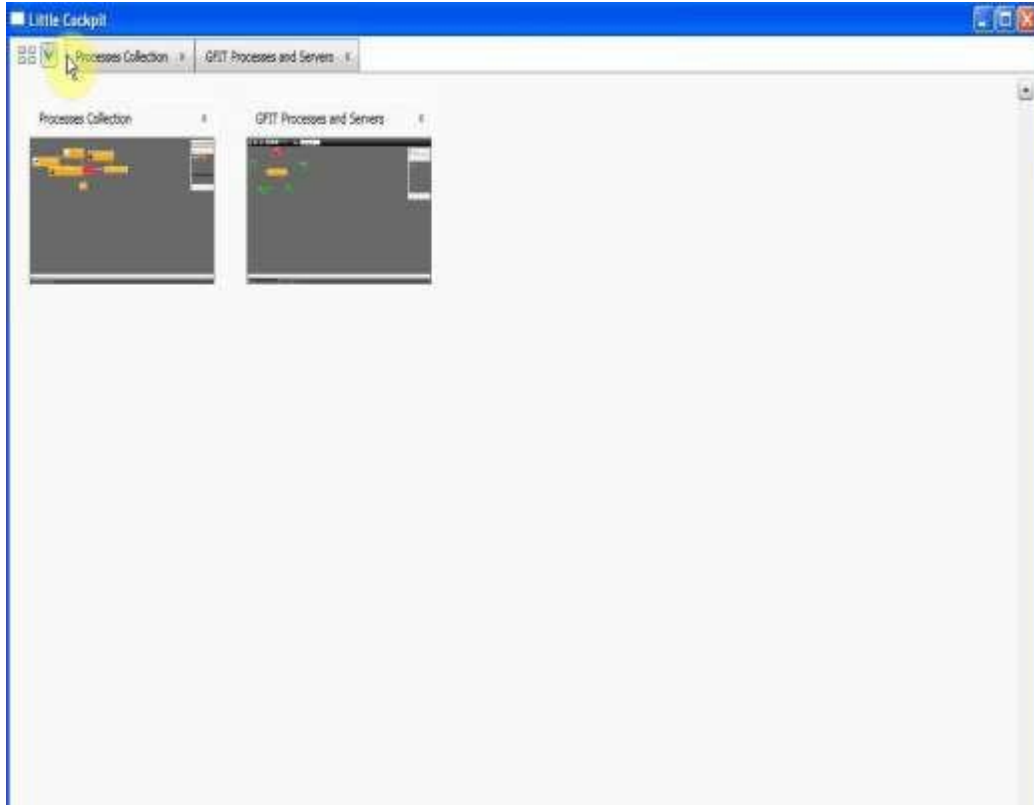


Figure 20 Tab view demonstration

In the Environment Cockpit, tabs can be created by double clicking on the CDP service collection. A new tab will appear to show all the processes within the CDP service collection. Tabs can also be closed by clicking the cross button on the right side of each tab. The leftmost small tab is the tabs overview, which shows the thumbnails and overall condition of all the existing tabs (Figure 19). Users can enter each tab for more information by double clicking its thumbnail.

Abstract items and data flow representations



Figure 21 The cockpit prototype snapshot II

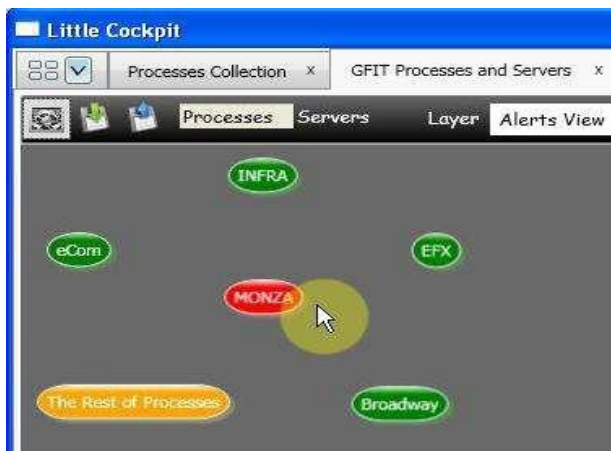


Figure 22 The cockpit prototype snapshot III

Each vertex in the Environment Cockpit represents an abstract item. An item, as mentioned before in the Analysis chapter, can be any processes, process collections, groups, domains, locations and servers. The shape and logo of each vertex are designed to be easily identified as the type of an abstract item. The color indicates the health condition of each domain, group and process. For example, as shown in Figure 21, the red and yellow vertices represent the domains with more than 90% and 50% of alerts-generating processes, respectively. The green vertices mean the domains are in a healthy condition. Each edge represents the dataflow, the correlation and the dependency from one process to the other process. The bidirectional edge stands for the two processes are exchanging data. With the similar design idea to that of the vertices, the edges have different colors and thickness set also to differentiate the connection types (such as RV, MQ or Wire), the health condition of these connections and the data traffic flow. When an alert on a certain process is raised, the Environment Cockpit gets informed from the system. The vertex of the process and the service collection that contains the error process flash red at the same time.

Hierarchical relationships representations

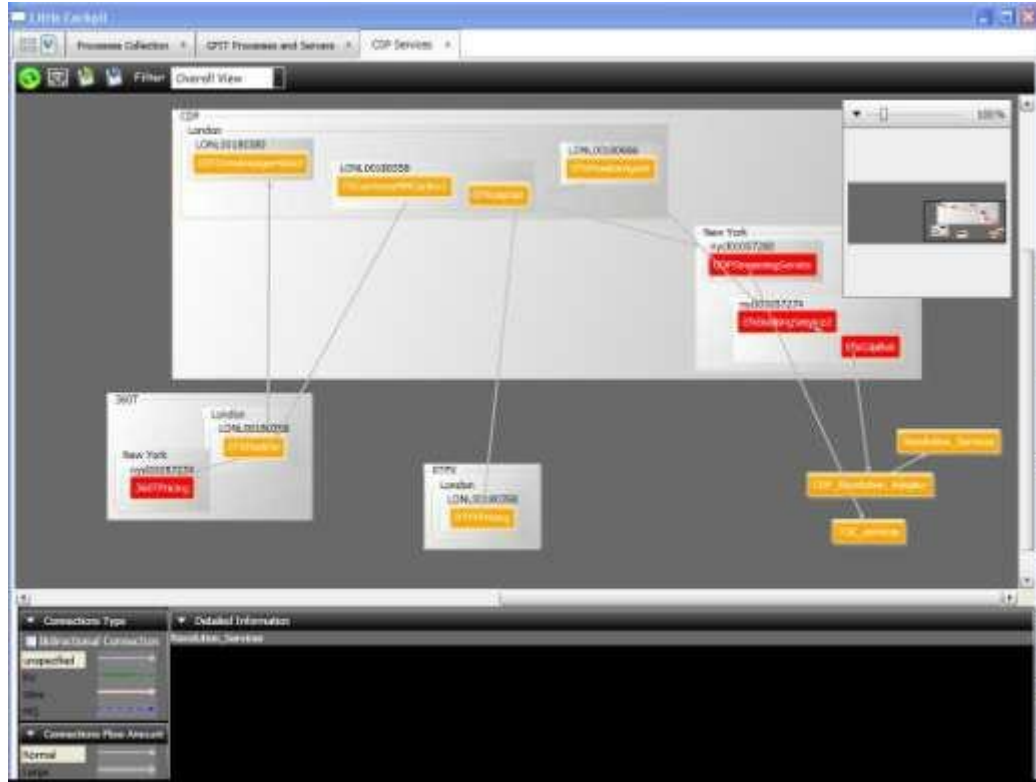


Figure 23 Cockpit Prototype snapshot IV

There are two kinds of relationship for all the abstract items, including hierarchical relationship and dataflow relationship. Hierarchical relationship is a containment connection between abstract items. For example, 360TPricing process belongs to 360T service collections; nycl00057274 server belongs to New York server collections; and 360TPricing process runs on nycl00057274 server. The hierarchical relationships are represented by means of compound boxes (Figure 22). Dataflow relationship is demonstrated with arrow connections. For example, EFXMarkSe process is pointed to 360TPricing, which takes market data from EFXMarkse. It is also very easy to differentiate which items represent processes because only processes can have dataflow relationships.

Logical view and physical view filter

Hierarchical relationship is then classified into two groups--logical hierarchical relationship and physical hierarchical relationship. Logical hierarchical relationship demonstrates the processes' properties logically. For example, the RTFXPricing process runs on the RTFX service collection. While physical hierarchical relationship describes the tangible locations of processes. All of the relationships, consisting of the logical and physical hierarchical relationship and dataflow relationship, can be displayed on the cockpit GUI (Figure 22). However, as we have demonstrated in the Analysis chapter, it is not very easy for users to quickly identify the relationships between items with physical and logical views both present. Especially in a relatively large banking environment, there are numerous processes running, and complex dataflow, which make it very difficult for users to discover the underlying condition in the environment. To address this issue, we implemented a filter that can allow users to choose which view they want to see, logical (Figure 23) or physical (Figure 24), in order to fully understand what is going on in the environment. Users can pick either -Logical View or -Physical View from the combo box on the toolbar.

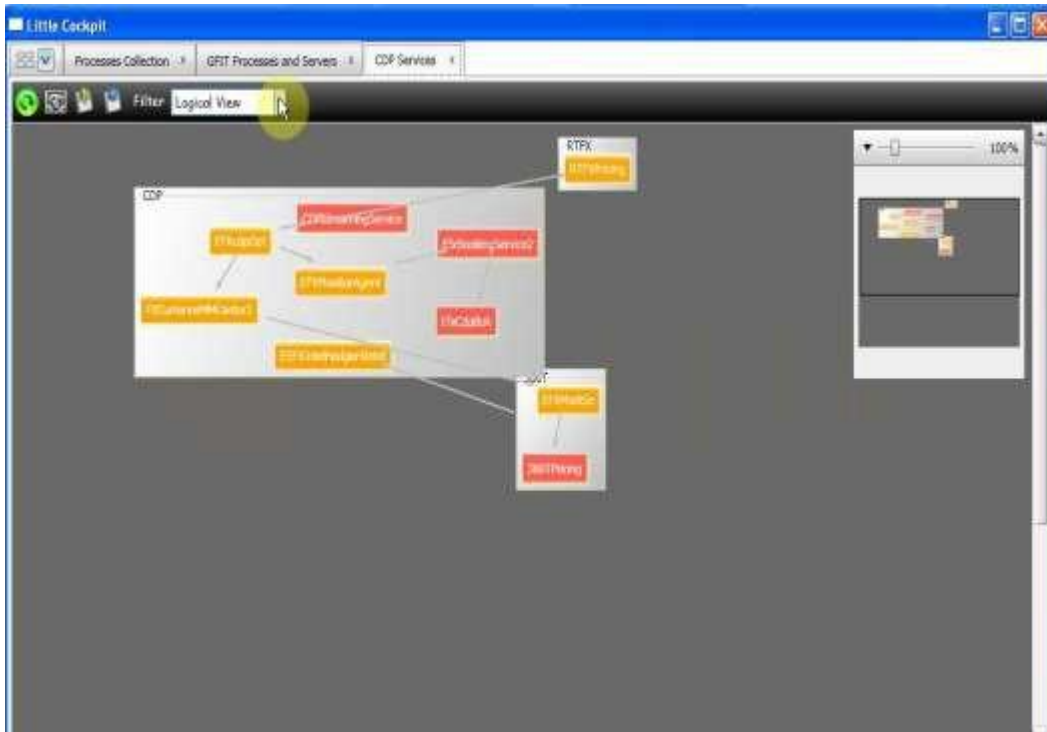


Figure 24 Logical view filter

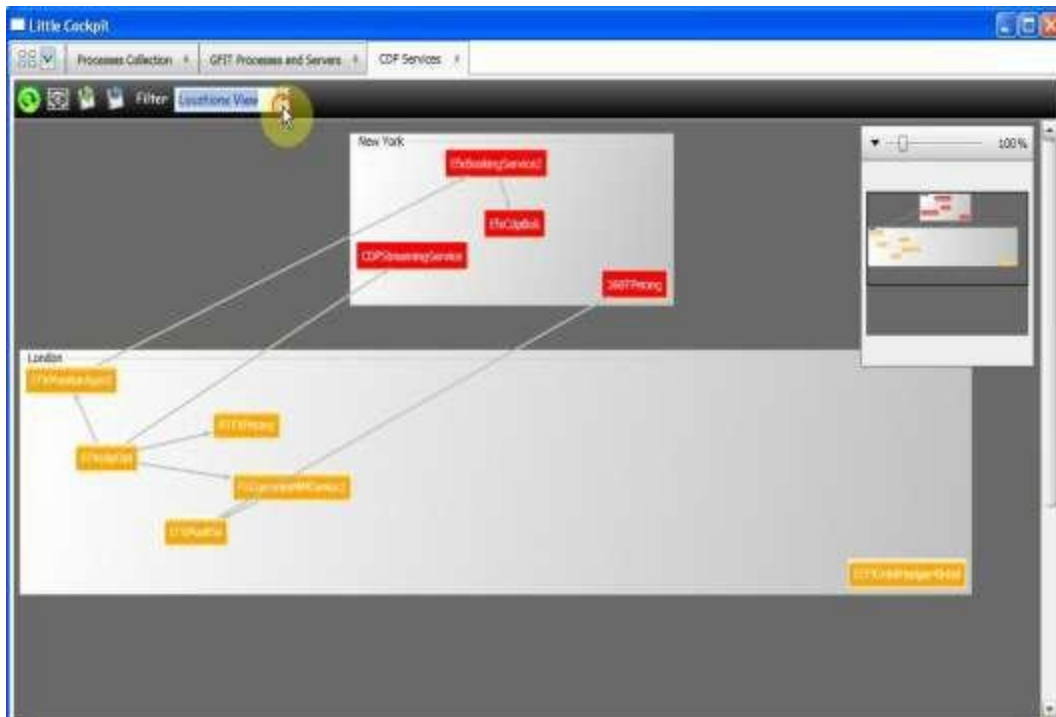


Figure 25 Physical view filter—location view

Adding and deleting different types of connections

As we have previously discussed in the Analysis chapter, it is not possible to automatically retrieve all the logical dataflow



Figure 27
Connectors

relationships from the system, especially for

database connections. Thus, for a better management of the dataflow connections, Environment Cockpit

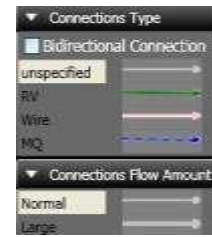


Figure 26
Connection Type

gives users the control to add and delete connections. Users can add

different types of connections, such as RV, Wire, MQ and unspecified, from the

connections type toolkit box on the left bottom side of the screen (Figure 25). The

dataflow connection can be selected as bidirectional, or unidirectional. Users can also

adjust the connection flow amount to stress how large the traffic amount is. With the

desired dataflow connection type and flow amount selected, users can then link the two

processes by clicking any one of the four square controls surrounding each process

(Figure 26). As shown in Figure 27 below, the unidirectional and small amount wire

dataflow from FXCurrenexMMCCantor2 process to EFXMonitorAgent process, the

unidirectional and large amount RV dataflow from FXCurrenexMMCCantor2 process to

EEFXIntelHedge-4Intel process and the bidirectional and large amount RV dataflow

between EEFXIntelHedge-4Intel process and EfxCdpBok process were manually added

by the users.

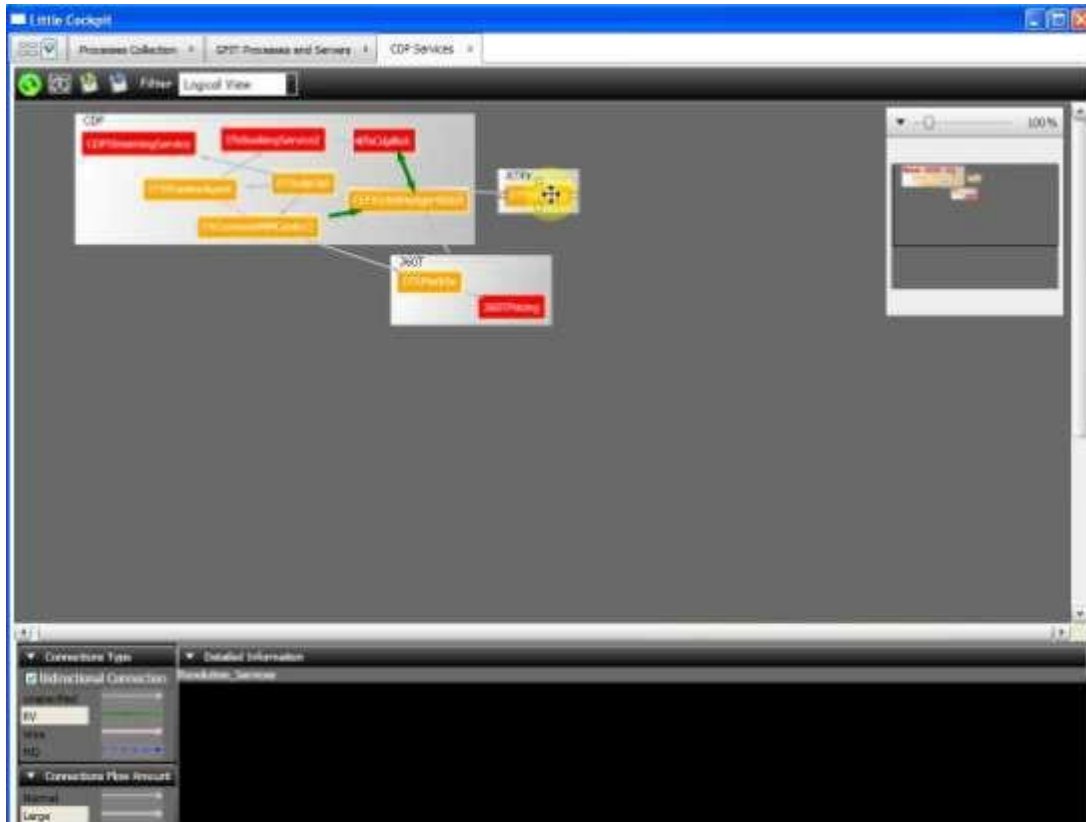


Figure 28 Cockpit Prototype Snapshot V



Figure 29 Connection pop up box

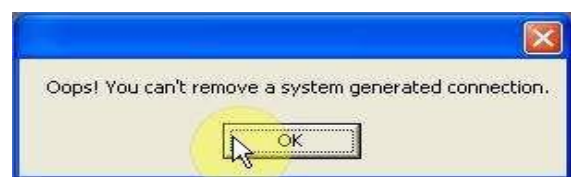


Figure 30 Warning message box

Users can also delete any dataflow by hovering over and then right clicking the connection that they would like to remove. A context menu box will show up. If the users choose the option of -Delete the connection, another message box that writes -Are you sure you want to delete the connection will pop up to confirm users' action (Figure 28). If the users choose -yes, the dataflow connection will then be successfully removed

from the canvas. The whole graph will re-layout automatically again according to the re-layout algorithm. However, users do not have the right to delete any system generated dataflow connections. When they try to delete a system generated connection, a warning box of –Oops, you can't remove a system generated connection will appear to prevent them from doing so because system generated connections are surely correct (Figure 29).

Importing and exporting graph files

The changed graph can be



Figure 31 Load and Save button

saved by clicking the save

button on the toolbar (the

third button from the left in Figure 30). The modifications will then be serialized and

transmitted to the cockpit server. On the opposite, if users choose to reload the display,

(the second button from the left in Figure 30), the changed graph will be de-serialized and

reloaded from the cockpit server into the cockpit prototype GUI. The reloaded graph may

not have the exact same layout as saved before. This is caused by the limitations of

graph# libraries that restricted re-layout algorithm and should be improved in the future

implementation of the Environment Cockpit. The system only button (the first button

from the left in Figure 30) is used for generating system only graph without showing any

changes that the users make.

Detailed information and log files display

When users left click



Figure 33 Detailed Information

Figure 32 Log Information

on a process or service collections vertex, the detailed information box at the bottom of the GUI will be updated, showing different versions, locations, machines and schedules of the clicked process (Figure 31). Users can also right click on each vertex to see the log information (Figure 32) of the process. However, at the moment, log file only contains dummy data since we have not yet built any plug-ins to retrieve the log information from the system. When the users hover over on vertices or connection lines, a yellow tooltip box will appear showing brief information about different vertices and edges.

Zoom box

At the right side of the GUI display, users can control the percentage axis at the top of the zoom box (Figure 33) to expand and diminish the size of the graph. In addition, the zoom box also provides a thumbnail of the whole graph. The framed small black box is used to signify the part that the user is zooming into.

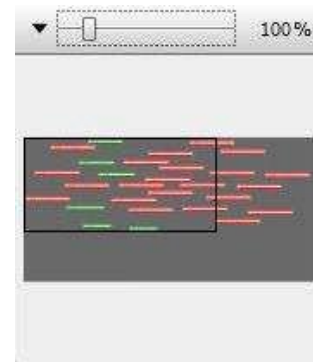


Figure 34 Zoom Box

Double click vertices

Service Collections, Domains and Groups Vertices are double-clickable and can then be expanded to their containing items collections.

User stories

User story I – tracing alerts



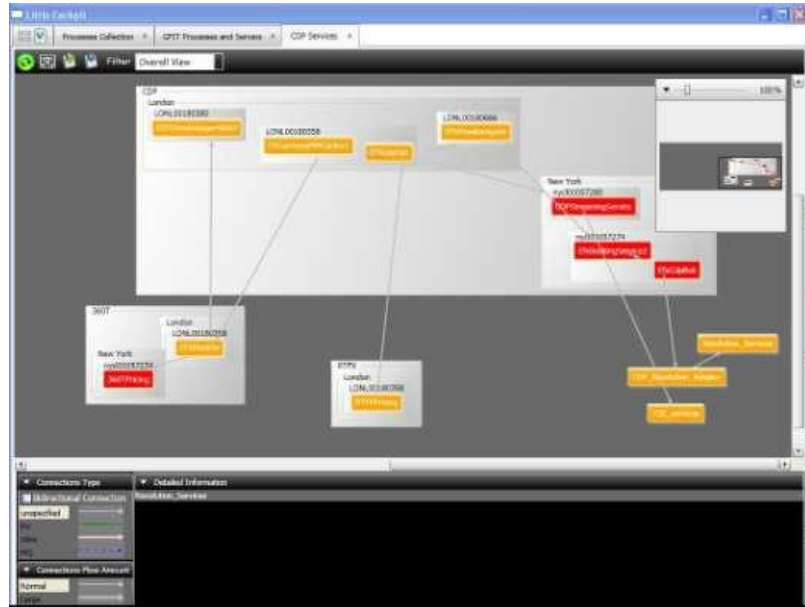
Figure 35

From the processes collection view with dummy data, users can notice that the CDP service is blinking red (Figure 34). This means alerts

were just raised from the system. If users want to troubleshoot the errors inside CDP service

collection, they can simply double click on the CDP services to view all the processes within CDP service

collection. Since the cockpit prototype does not always generate a perfect graph layout, users can click on the re-layout button to generate a clearer view.



From the overall view (Figure 35),

Figure 37 Overall view

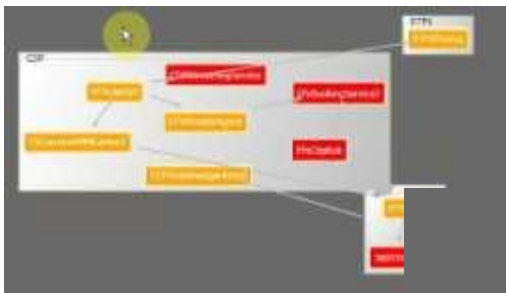


Figure 36 Logical View

view, the cockpit GUI gives information about which processes are sitting in the corresponding service

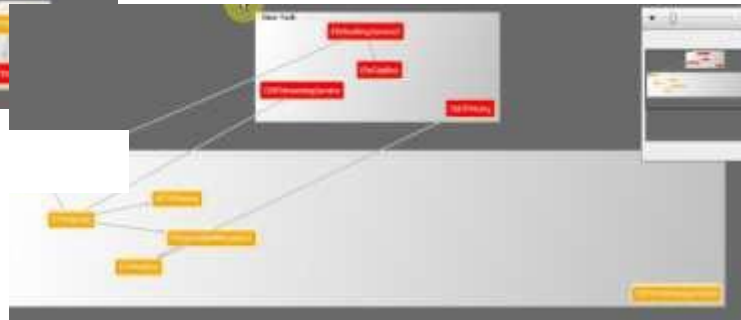


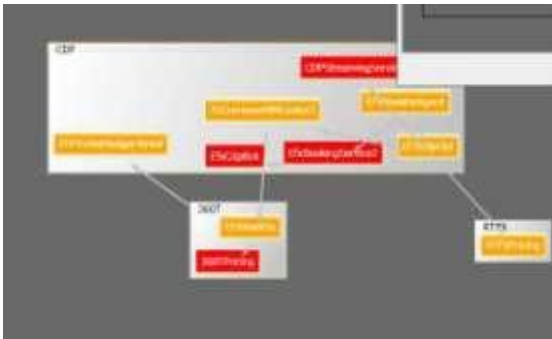
Figure 38 Physical View

collection such as CDP, RTFX

and 360T. In the physical view, the GUI displays which processes are running in different locations, like New York or London. The dataflow is already all over the place with a few processes shown on the graph (Figure 37). Imagine there are many more processes and much more complex data flow in the real world. The view that combines both physical view and logical view gives too much information for users to handle. Furthermore, as users, they can hardly find out why alerts were generated and how these alerts were related. In order to investigate more about the cause of these alerts, users can utilize the built-in filter to separate out the information they are not concerned about and gain a better knowledge of the information they care about. If users choose -logical view (Figure 36), all the processes will be relocated into different service collections. Similarly, if users switch to -location view (Figure 37), all the processes will be relocated into different locations. From the location view, it is obvious that all the processes in New

York are red. This means that the New York servers have something going wrong and generate these alerts.

User story II – adding connections



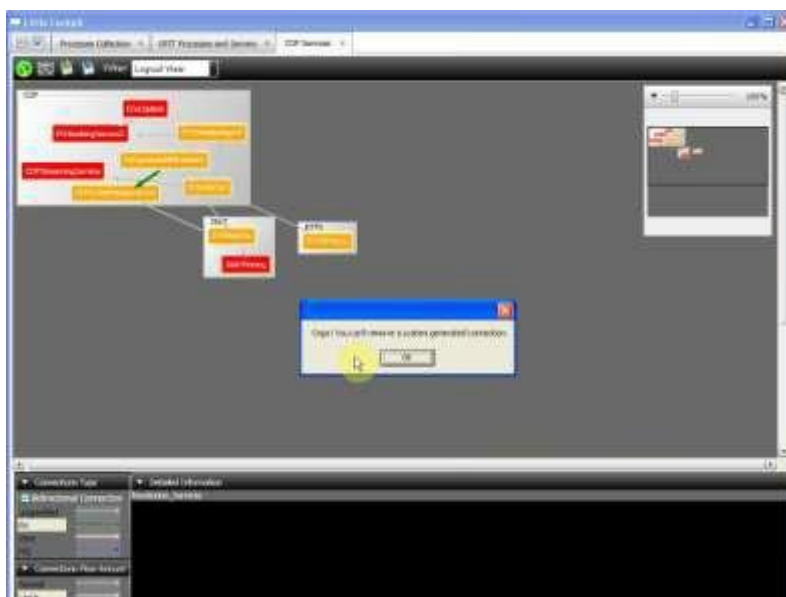
As the APS supporters, if they discover that there should be a small amount wire data flowing from FXCurrenexMMCantor2 process to EFXMonitorAgent process but this connection is not detected or displayed

Figure 39 Adding Connection

by the cockpit prototype. The supporters can

then manually add the connection by selecting –Connection Type as –Wire and then click on the two processes. A connection will be added on the graph (Figure 38).The new connection information will be transmitted into the cockpit server to be saved and distributed throughout all the other users.

User Story III – Deleting Connections



As the APS supporters, if they decide that there shouldn't be any dataflow connections from EFXMarkSe process to 360TPricing process, they can just delete the connections by right

Figure 40 Delete Connection

clicking on the data flow line. However, since the connection between EFXMarkSeprocess and 360TPricing process is system generated, users don't have the right to delete them manually. A pop up warning box will appear to remind the supporters that this connection cannot be removed (Figure 38).

User story IV – Exploring real data



From the GFit processes and servers View (Figure 41), if the APS supporters want to dig out why Monza domain (red on the graph) is in a seriously ill condition, they can double click on the -Monzall domain, and enter the two groups, -COREll and

-MONZAll within the -Monzall

Figure 42 Gfit Processes and Servers

domain. As Figure 40 indicates, Figure 36

both -COREll and -MONZAll groups are red too. If the supporters double click on the

-MONZAll group, all the processes within the group (Figure 41) will

show up in red, meaning every item within -MONZAll group is

generating alerts. This indicates the whole Monza system is down.



Figure 41 Group View of Monza



Figure 41: Double click on "Monza"

Cockpit prototype

implementation strategy

The Environment Cockpit prototype uses WPF as its major coding platform. We implemented both of the GUI and the back end sides, using xaml and C# programming language respectively. The Wire and graph# are the two major technologies used in the prototype. The Wire, an in-house developed middleware tool, and protocol buffer are applied to transmit information between the cockpit server and clients. Graph#, a WPF platform based library, is employed to construct the GUI display.

The two major concepts developed in the cockpit prototype are the idea of abstract item and server with plug-ins. As mentioned before in the Analysis section, an abstract item can be any process, server and service collections. Each item was assigned with a unique ID, type and different attributes. In this way, the wire can transmit information between the server and clients without knowing what type of information it is transmitting. Server with plug-ins can provide a common interface for all the information generated from the different existing applications.

Coding analysis

Two open source libraries

We utilized two online open source libraries downloaded from the CodePro website. They are FabTab, a very popular tab view library, and graph#, a WPF platform supported library based on GraphViz that provides a powerful and fundamental graph

drawing resource for the cockpit prototype. Graph# includes more than ten layout algorithms with one compound layout algorithm, the one that the cockpit prototype is currently using. Although the compound layout is not ideal and sometimes doesn't produce a perfect layout, graph# is still a pioneer library that can provide the technology for compound vertices. However, since graph# cannot manage the compound vertices and update the graph at the same time, the team revised the Graphsharp.control library to make graph# better fit into the cockpit prototype. In the future, the Environment Cockpit development team is recommended to keep track of the development of graph#, a powerful and free graphing tool.

Three self-implemented libraries

We implemented three library packages with about 1146 lines of code. The package consists of GFitDataBase, Service and Client packages. GFitDataBase is a library connecting to the Gfit database and retrieving real life data of the processes and servers basic information from the database. Service library is made up of MockItemSource and LittleCockpitService as two major classes to create dummy data and set up the wire discovery service to transmit both the dummy and real data that GFitDataBase library produces. The dummy data mocks the future ideal data generated from the system. Client is a library creating the wire client to subscribe the dummy data and the real data from the wire discovery service.

One test package

We also implemented a test package with about 90 lines of code. The test package consists of one major test to check if the wire is installed correctly. This is a very useful package during the debugging process because it can analyze whether the error is caused by the wire or by the cockpit prototype GUI.

Two builds

There are two builds within the cockpit prototype, including WireServiceStartUpConsole (26 lines of code) and the LittleCockpit builds. WireServiceStartUpConsole should be run at first to register the cockpit wire service. Users can then run the client side, which is the LittleCockpit build to boot the cockpit prototype GUI.

The Little Cockpit GUI code analysis

Little Cockpit GUI includes four folders, which are GraphBits, Images, Resources and ViewModels folders, five windows and seven separated classes.

GraphBits contains 470 lines of code, including PocEdge, PocVertex and PocGraph scripts, which define the customized vertex, edge and graph, respectively so that each control can bear many self-implemented attributes. For example, in the cockpit prototype, a vertex has not only ID as one of its basic attributes, but also health condition, status, color and double clickable as the other added-on attributes. After processes and processes' attributes are transmitted from the wire, they are then translated into different color and shape of vertices to be displayed on the GUI. Images folder contains image files used in the cockpit prototype. Resources folder defines the style of almost all of the UI elements within the cockpit prototype and provides some useful templates and brushes. The resources folder, with around 1900 lines of code in total, contains the style scripts of expander, scrollbar, scrolling viewer, slider, toolbar, toolbox, tooltip and zoom box, a brushes library with different colors and gradients, and most importantly, a DesignerItem xaml markup that defines the style of vertices and edges. By using data binding provided by WPF, the cockpit prototype binds different attributes of vertices and edges to different vertex color or shapes and edge thickness.

MainwindowViewModel C# script with a total of 1135 lines of code is the part that connects the backend data to the view. To be more specific, MainwindowViewModel's main functionality is to produce the view of the graph with the backend data. With the data transmitted from the wire, MainwindowViewModel calculates how many vertices there should be to be put on the canvas, and determines the attributes of these vertices. MainwindowViewModel also provides methods of producing different types of view, for example, loadlogicalview method will ignore the physical locations aspect of the processes and only display the logical belongings while loadLocationview method will only focus on the location aspect of different vertices.

Five windows include CDPMainWindow (437 lines of code), DummyProcessCollectionMainWindow (111 lines of code), GFitWindow (494 lines of code), InfoLog (60 lines of code) and MainWindow (240 lines of code). The MainWindow window utilizes FabTab library to set up a basic tab view and then adds DummyProcessCollectionMainWindow and GFitWindow for the -Processes Collection and the -GFit Processes and Servers tabs, respectively, within the prototype GUI. When users double click on the CDP service collection vertex, they trigger the CDPMainWindow window to show up in a new tab and the infoLog window is triggered by right clicking on the -Show Log information option on each vertex.

Each of the seven separated classes plays a very important role in the cockpit prototype. Connector class (125 lines) implements the connector control around each vertex. This class incorporates a mouse clicking event control function that can memorize the last two clicked vertices and add the edge between them. MoveThumb class (182 lines) allows vertices to be movable and resizes the canvas according to the whole

vertices layout. In addition, MoveThumb class implements the functionalities of left button clicking to update the detailed information box and double clicking to dig into a deeper level of the graph. Singleton class (49 lines) is another crucial class to expose several essential entities so that they can be used globally, for example, the previously clicked vertex, the current clicked vertex, the main window and so on. PocSerializeHelper class (92 lines) is to facilitate the save and reload process and EdgeRouteTopathConverter (134 lines) is used to draw the arrow head shape of the connections. Toolbox (29 lines) and Zoom box (117 lines) classes implement the toolbox and zoom box functionalities respectively.

Within the past seven weeks, the team learned WPF, C# and xmal by themselves, worked with graph#, the wire and FabTab libraries and then implemented around 5000 lines of code. At the end of the project, they successfully realized several crucial functionalities, such as filter, connection management and alerting system within the cockpit prototype GUI, and proved the idea and feasibility of the Environment Cockpit. Because of the size of the code, it is not feasible to attach all of the cockpit prototype code in this report. Thus only several important pieces of sample code are demonstrated in Appendix A.1 – Cockpit Prototype Code and a small piece of the serialized cockpit item is illustrated in Appendix A.2 – Serialized Cockpit Item

The Environment Cockpit GUI

In order to define a clearer Environment Cockpit GUI design for future implementation of the real Environment Cockpit, the architecture group and our team

actively interacted with UX team and the APS team supporters. The UX team and our team came up with the following GUI design.



Figure 43 UX team Gui Design I

From figure 42, on the top of the toolbar, the Environment Cockpit is currently showing FX Global Environment. The green boxes indicate the health of each service collection and the lines demonstrate the dataflow. From the toolbar on the top, users can choose to view different layers of information, such as processes status or processes connection status. Users can also click on -Edit| button to add and remove connections within the graph. In the bottom of the GUI, users can view the alerts grouped by difference source, process group, application, or type and also the alerts log that shows the history record of the alerts. From the graph canvas, users can tell the blackbird service is going wrong with one red box. Users can then double click on the service collection to get a detailed logical view about why the Blackbird service is giving error. From the

processes view below in figure 43, it is clear that the red connection of the data flowing from the RMDS is actually giving error information into the Blackbird process.

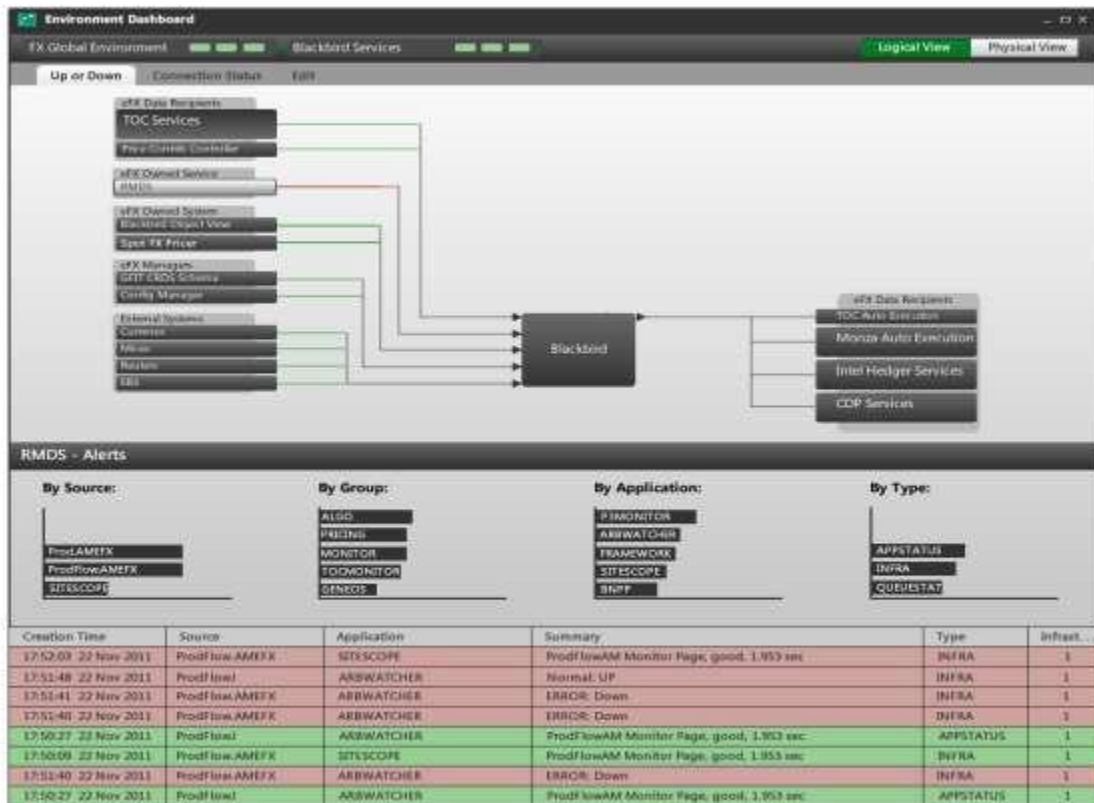


Figure 44 UX team Gui Design II

From the top right hand side, users can also switch to the physical view of the Blackbird services collection (Figure 44). The middle panel shows different black boxes of servers or server collections sitting in grey boxes of zones within London or New York locations. From the toolbar on the top, users can filter out different types of connections by picking -All connections, -Manual Connections, -System Connections, tabs. In the bottom, users can view the detailed information of servers that the Blackbird service collection is running on, which includes location, zone, memory, CPU and disk space of

the servers. If the users double click on a server collection, for example, Nirvana Servers Collection, the bottom detailed information box will be updated to the servers' information that is only related to the Nirvana Servers Collection.



Figure 45 UX team Gui Design III

The Environment Cockpit proof of concept

The Environment Cockpit is designed by the architecture team and our team to fill in the blanks of the current existing monitoring tools and to accommodate various requirements from the APS supporters. Within all the functionalities proposed in the Environment Cockpit, developers were especially concerned with the functionality of giving users the right to add and delete connections. Developers generally doubted that

the users will actually take the time to add and delete connections and maintain the Environment Cockpit dataflow by themselves. Through many interviews and discussions, there were several APS supporters confirmed that they would like to see and utilize this functionality; however, there were also opposed voice from others. Thus, we strongly recommend to investigate more and to collect more precise users' responses regarding to this functionality.

Except the main functionalities described in the Environment Cockpit GUI section above, several other functionalities were very popular among the APS supporters and it will be very beneficial to consider the implementation of them in the future. First is placing layers on vertices. For the APS supporter, the most important information that matters to them is the status of applications and the status of alerts. Thus showing these two parts of information straightforwardly by placing extra information layers on top of each vertex within the Environment Cockpit will be very useful. Second is dynamic alerts notification. Because users usually customize several filters to focus on the areas of the environment that they are interested in, they will often neglect the health condition and alerts in the other part of the environment. Thus, it is recommended to have a small spot in the GUI to indicate the health condition and the alerts happening in the part of the environment other than the users' focus area. Third is automation. The Environment Cockpit is going to include enormous complex data of the whole environment and the maintenance of this giant data entity is for sure going to be problematic. Because of this, the Environment Cockpit is designed to use automation as much as possible. For example, whenever a team deploys a new process, the Environment Cockpit should get informed and display the newly added process at real time.

Some other user stories were also depicted by the APS supporters and developers, who expressed the hope for the Environment Cockpit to have these abilities in the future. One user case is to have detecting ability of ineffectiveness in the dataflow. For example, if process A locating in London acquires data from process B in New York, while there is process C in London offering exactly the same service as process B, it is quite obvious that using process C will remove the data transmission overseas and, as a result, improve efficiency. The environment cockpit should be able to display, detect and highlight this kind of inefficiency for users so that users can make improvements of the environment dataflow. The other user case is to show server resource usage within the system. For example, if there are several processes providing the same service and only one of these processes is actually requested by many other clients, this process will be way overloaded and the speed of computing will decrease; while the other processes providing the same service are just left idle. The Environment Cockpit should again have the ability of detecting and demonstrating this kind of inefficiency in the system and alerting users about it.

Throughout the requirements collection period, our team not only gathered suggestions specifically for the environment cockpit, but also pulled together some other ideas of improving the monitoring system in the bank as a whole. For example, the existing tools name the same applications and processes differently and this is very confusing for users when they want to evaluate the same process across different platforms. Thus, there is an urge demand to have a unified application name within the system across all the monitoring platforms. The unified status type of the application is also recommended because different tools display different types of status for the

processes. The current alert message was also said to be not accurate and straightforward by the APS supporters. Thus, developers are highly suggested to write good alert message for their products.

The technology of Graph# was proven to be a cheap, useful, appealing and suitable for the Environment Cockpit through the prototype development. However, there are a couple points that need to be noticed by future developers. First, Graph# is still under development. The current version used in the cockpit prototype is the most recent Oct 2011 version and future developers should follow the Graph# Code Project homepage to retrieve the updated version with more methods. Second, as we have mentioned in the prototype section before, Graph# does not always generate a perfect graph layout with compound boxes. Thus, efforts can be made to improve the existing compound graph layout algorithm in graph#. Third, when people save graph, they expect to reload the exact same layout as the saved graph. Therefore, the Environment Cockpit in the future should have the ability of disabling the automatic layout algorithm imposed by Graph#.

Through the seven weeks' work, the Environment Cockpit is proven to be not only a powerful monitoring tool and highly expected by users, but also feasible in implementation through the development of the prototype. As the progress of the Environment Cockpit continues after the leave of our team, the cockpit developers can refer to the cockpit prototype for some useful technology, such as Graph# and the Wire. They can also implement according to the architecture diagram and follow the GUI design illustrated in the above Outcome section. They can consider the other user requirements collected by our team and keep users actively involved throughout their

implementation of the Environment Cockpit to create a well-designed, popular and powerful monitoring tool.

Methodology

A major difficulty of the project was the long distance development between the two WPI team members. Qiu worked in the BNP Paribas London site while Linda worked in the BNP Paribas New York site. The time difference and the communication difficulties at first obstructed the project progress. The major communication tool used in the bank, Windows Communicator, took some time to be set up in New York. The first several meetings were spent resolving technical difficulties involving international conference calling. In addition, Qiu and Linda had to overcome the time difference to work together on the project. The mid-project presentation and one final presentation were successful although the two team members were in different countries.

The first few days of working on the project, we familiarized ourselves with the internal environment monitoring software such as Sam, BlackBird, TOC and so on. Then, we met with the APS group to learn about the limitations of the existing applications. The aim was to gain a full understanding of the existing monitoring tools in order to better design the Environment Cockpit.

Since the Environment Cockpit is a start up project in the bank, the first task was to define the project scope and gather user requirements. We had a few meetings with the stakeholders and different teams, including the APS group and the architecture team, to discuss what features they would like to include in the cockpit and their ideas on the implementation of the project in the future. Although we were in different locations, we

still adopted the Agile development working method as the project proceeded. On a daily basis, we managed to speak to our sponsors to ensure the functionality requirements and project scope suited business needs. Meanwhile, we held daily discussions in order to separate daily tasks, and discuss about development strategies and implementation plans. These frequent communications greatly helped project development and allowed the project to be adaptable to the ongoing client requests.

Soon after the project scope was cleared, we started implementing the user interface prototype. The goal was to create visuals that are practical and useful in accordance with functionality, but also pleasing to the user. This required not only excellent coding skills, but also the use of a proper graphing tool. The team investigated numerous graphing technologies such as Nshape, graphviz, Yworks, graph# and others. Graph# turned out to be the best fit for the project though it was relatively new and there was not much documentation available. The options were considered carefully given the constraints of the limited project timeline and the restricted option of programming languages (the programming languages were restricted to C# and WPF).

We encountered many technical challenges as the project went on. For example, the physical view and logical view were difficult to combine and display graphically. There was so much information that it was a struggle to create a user interface that was both practical and thorough. The visual aesthetics of the user interface were limited by practicality. As the Outcome chapter stated, data flow between different elements in the environment would become messy and disordered with every piece of information shown. To address this issue, we vigorously interacted with the architecture team in the eCommerce group, brainstormed, experimented and finally discovered the solution of

implementing a filter that would allow users to choose views. The views can be either logical or physical. This solution was rather successful because it practically presented all the important information without ruining the aesthetics of the GUI.

WPI student team also worked with the UX group for user interface design, and outlined a solid proof of concept regarding how environment cockpit project could be developed and who else it could benefit in the future. The future steps are to establish the back end connection, develop environment control and implement UX group's user interface design. A timeline of the implementation and workflow steps of the WPI student team's project work is attached in Appendix C – Timeline.

Conclusion

The seven weeks of working on the Environment Cockpit project was a fantastic learning experience. We were honored to be given such a great opportunity to work with so many intelligent and friendly people from a world-class top-notch bank. We were extremely appreciative of all the given guidance, support and inspiration along the way.

The aim of the Environment Cockpit project was to deliver a business solution to help supporters efficiently monitor and maintain the system. Inspired by the original idea, bubble view and grid view, proposed by Wells Powell and Huw Roberts respectively, we developed a user interface that could visually present all the environment information and allow users to control, manage and maintain the environment. Environment Cockpit centralizes all the environment information, provides a dashboard overview of the system status, and displays the general health condition of the environment. In addition, Environment Cockpit is able to display the complex data flow among all the elements, which no other existing application currently does.

The Environment Cockpit can revolutionize the way supporters monitor the environment. With the Cockpit, they do not have to open all the discreet existing monitoring tools, fit them into a handful of screens, and analyze the environment's overall health condition from all the complex and huge tables of existing tools. Instead, they could just open up one application, the Environment Cockpit, view the system health graphically, dig out the real cause of the alerts, understand environment dynamic, and have better control of the system. Specifically, if anything in the environment goes

wrong, alerts would be generated to inform the users. Users could quickly locate the error, what caused the error, and quickly discover the other applications that the error may affect. This was achieved by displaying two forms of information in the Environment Cockpit— the physical view and the logical view. Depending on what information they would like to know, users can choose which type of view they want to see. Users are also given certain controls such as adding/deleting connections to monitor system data flow and saving/loading files from the cockpit server. A log file, tooltip and information box were also provided to reflect the system situation in different ways.

Upon the completion of the Cockpit prototype, we recommended a few steps that the future Environment Cockpit development group could take to continue the project and summarized the other user requirements that haven't been implemented in the prototype or designed in the GUI, but were strongly expressed by the supporters. By defining the project scope, investigating graphing tools, constructing the prototype, and providing an implementation strategy for the future, WPI's student team intended to initiate the huge environment cockpit project for the bank. Ultimately, the Environment Cockpit will improve the supporters' efficiency and provide better assistance for monitoring trading events.

Acknowledgements

From BNP Paribas

- Wells Powell – Head of eComm Group
- Huw Roberts – eComm Architecture
- Sunai Patel – Configuration Management Lead
- Nicolas Wright - Configuration Management --New York
- Martin Gittins – Architecture Developer
- Daniel Slater – Architecture Developer
- Mohammed Abu Sharikh—Architecture Developer
- Nick Matterson – User Experience Team
- Emmanuel Philipon – Application Production Support
- Amine Bakkali Yedri – Application Production Support—New York
- Sebastien Dubuisson – Market Data Team

From WPI

- Professor Arthur Gerstenfeld—WPI School of Business
- Professor Daniel Dougherty— WPI Department of Computer Science
- Professor Xingming Huang—WPI Department of Electrical and Computer Engineering
- Professor Jon Abraham—WPI Department of Mathematical Science

References

-The server., Comer, Douglas E.; Stevens, David L. (1993). *Vol III: Client-Server Programming and Applications*. Internetworking with TCP/IP. Department of Computer Sciences, Purdue University, West Lafayette, IN 47907: [Prentice Hall](#). pp. 11d. [ISBN 0134742222](#).

"Client" Wikipedia, Dec 20th. Web. <
http://en.wikipedia.org/wiki/Client_%28computing%29>.

Borysowich, Craig. "Overview of Dependency Diagrams." *IT Communities - Share Knowledge at Toolbox.com*. 25 May 2007. Web. 29 Dec. 2011.
<<http://it.toolbox.com/blogs/enterprise-solutions/overview-of-dependency-diagrams-16493>>.

"Developer Guide" Protocol Buffers Google Code, Dec 20th. Web. <
<http://code.google.com/intl/zh-CN/apis/protocolbuffers/docs/overview.html>>.

Roberts, Huw. *ECommerce Cockpit Straw Man*. Tech. 1.0st ed. London: BNP Paribas ECommerce, 2011. Print.

Bucanek, Jame. *Learn Objective-C for Java Developers*. Apress, 2009. Print.

"Discovery Service." *BNP Paribas Wiki*. BNP Paribas London, 2009. Web. 26 Oct. 2011.

Patel, Sunai. Personal interview. Oct 30th. 2011.

Beal, Vangie. "What Is A Server Platform? ?? Webopedia.com." *Webopedia: Online Computer Dictionary for Computer and Internet Terms and Definitions*. 28 Jan. 2005. Web. 29 Dec. 2011.
<http://www.webopedia.com/DidYouKnow/Hardware_Software/2005/servers.asp>.

"The Wire." *BNP Paribas Company Wiki*. Web. Nov 15th. 2011. <
<http://wiki.london.echonet/display/DC/Discovery+Service+Overview> >

"Heartbeat." *BNP Paribas Company Wiki*. Web. Nov 15th. 2011.

Storey, Sean. Personal interview. Nov 30th. 2011.

- Dubuisson, Sebastien. Personal interview. Nov 30th.2011.
- "Sam." *BNP Paribas Company Wiki*. Web. Nov 15th. 2011.
- West, Douglas B. *Chaos: Introduction to Graph Theory*. Prentice Hall, 2001. Print.
- "*Windows Presentation Foundation*." Microsoft, Dec 20th. Web. <
<http://msdn.microsoft.com/en-us/library/ms754130.aspx>>.
- Martin, Robert Cecil. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall, 2003. Print.
- Fruchterman, Thomas M.J. Reingold, Edward M. "Graph drawing by force-directed placement" *Software: Practice and Experience* 21.11 (1991): 1129-1164. Print.
- Brent, R.P. Kung, H.T. "A Regular Layout for Parallel Adders" *IEEE Transactions on Computers* C-31.3 (1982): 260-264. Print.
- Ashcraft, B.Adam, and Til Schuermann. "Understanding the Securitization of Subprime Mortgage Credit." *Federal Reserve Bank of New York Staff Reports* no. 318. Print. Mar. 2008. Feb. 2012
- Bank of China. *Underwriting of Structured Product Financing*. Retrieved February-17th, 2012 http://www.boc.cn/cbservice/cb2/cb24/200807/t20080710_817.html
- Barnett-Hart, K. Anna. "The Story of the CDO Market Meltdown: An Empirical Analysis." Harvard College Cambridge, Massachusetts. Print. 19 Mar. 2009. Feb. 2012
- Bigio, Saki, and Jennifer La'O. "The 2008 Financial Crisis: Institutional Facts, Data and Economic Research." Print. 29 Aug. 2011. Feb. 2012.
- Black, Fischer and Myron Scholes. "The pricing of Options and Corporate Liabilities." *Journal of Political Economy*. Vol 81, No. 3 (May – June., 1973) 637-654, Print.
- Capone, P. Elisa. "Collateralized Debt Obligations(CDOs): An Introduction." *RGE monitor*. Print. 7 Mar.2007. Feb. 2012.
- Dowd, Kevin. "Moral Hazard and The Financial Crisis." *Cato Journal*, Vol.29, No. 1. Cato

- Institute. Print. Winter. 2009. Feb. 2012
- Fabozzi, Frank J., Henry A. Davis and Moorad Choudhry. *Introduction to Structured Finance*. The United States of America: John Wiley & Sons, Inc, 2006. Print.
- Fagan, Mark, and Frankel Tamar. "MBS, ABS, SPV, CDS, ARM, BBB+: Understanding the Alphabet Soup of Securitization." Copyright Fagan and Frankel. Print. Oct. 2008. Feb. 2012.
- Fender, Ingo, and John Kiff. "CDO rating methodology: Some thoughts on model risk and its implications." *Bank for International Settlements Working Papers No. 163*. Monetary and Economic Department. Print. Nov. 2004. Feb. 2012
- Gonzalez, Carlos. "Why Securitization?" Stewart title Latin America. Print. Feb. 2012.
- "House of Cards" CNBC. 12 Feb. 2009. Media. Feb. 2012
- Jobst, Andreas. "What Is Securitization?" *Financial & Development*. Print. Sept. 2008. Feb. 2012
- Katz, Jonathan, Emanuel Salinas, and Constantinos Stephanou. "Credit Rating Agencies." *Crisis response*. Print. Oct. 2009. Feb. 2012
- Library of Economics and Liberty. *Takeovers and Leveraged Buyouts*. Retrieved Feb-23rd, 2012
<http://www.econlib.org/library/Enc1/TakeoversandLeveragedBuyouts.html>
- Lowenstein, Roger. *The End of Wall Street*. New York: Penguin, 2010. Print. Feb. 2012
- O'Kane, Dominic and Stuart Turnbull. "*Valuation of Credit Default Swaps*." Lehman Brothers, Quantitative Credit Research April 2003: Print.
- Song, Shin Hyun. "Securitization and Financial Stability." *The Economic Journal*, 119 (March), 309-332. Royal Economic Society. Print. 2009. Feb. 2012
- Stewart, Frances. "The 2008-9 crisis and developing countries: implications for poverty." Print. Feb. 2012

"2008 Financial Crisis." *Concept*. Web. 29 Feb. 2012.

<http://www.wikinvest.com/concept/2008_Financial_Crisis>.

"Report on securitization incentives". *Joint Forum*. Web. Feb.

2012.<http://www.bis.org/press/p110713.html>

Appendix

Appendix A.1 – Cockpit Prototype Code

CockpitService.cs

```
using System;using System.Collections.Generic;

using System.Linq;

using System.Threading;

using Bnpp.Gfit.Proto.Wire.Samples;

using Bnpp.Gfit.Wire;

using Bnpp.Gfit.Wire.Entities;

using Bnpp.Gfit.Wire.Services;

using Bnpp.Gfit.Wire.Transport;

namespace QiuLindaService{

    class Program{

        private const string WireServiceName = "EfX.Wire.BlotterSample";

        private const int ChunkSize = 100;

        private readonly String name;

        private readonly ReaderWriterLockSlim rwLock = new ReaderWriterLockSlim();

        private WireServer wireServer;

        public QiuLindaService(String name)
```

```

{
    this.name = name;
}

private static void InitialisewireEnvironment(){

    if (WireEnvironment.Current == null)
    {
        WireEnvironment.Default.Location = "LON";
        WireEnvironment.Default.Environment = "Dev";

        WireEnvironment.Default.SubEnvironment = Environment.MachineName
+ Environment.UserName;

        WireEnvironment.Default.Initialise();

        this.wireServer = new WireServer(WireServiceName);

        this.wireServer.SetMessageHandler<GetQiuLindaSubscription>(this.HandleQiuL
indaSubscription);

        this.wireServer.Error += (s, error) => Console.WriteLine("WireListener
Error:" + error.Exception);

        this.wireServer.Start();
    }

}

private void HandlePingPongSubscription(GetPingPongSubscription message,
Channel channel)
{

```

```

        this.rwLock.EnterReadLock();

        do
        {
            GetPingPongSubscription name = new GetPingPongSubscription();

            Console.WriteLine("Received " +
GetpingPongSubscriptionMessage.Message)
                channel.SendMessage(name);
        }
        finally
        {
            this.rwLock.ExitReadLock();
        }
    }
}

```

CockpitServerStartupConsole.cs

```

using System;

using Bnpp.Gfit.Proto.Wire.Samples;
using Bnpp.Gfit.Wire;
using Bnpp.Gfit.Wire.Entities;
using Bnpp.Gfit.Wire.Services;
using Bnpp.Gfit.Wire.Transport;

```

```

namespace QiuLindaService{

class Program
{
    static void Main(string[] args)
    {
        InitialiseWireEnvironment();

        using (var server = new QiuLindaService())
        {
            server.SetMessageHandler<PingPong>(HandlePingPongSubscription);

            server.Start();

            server.handlePingPongSubscription();

            Console.WriteLine("Press return to exit");

            Console.ReadLine();

        }
    }
}

```

EdgeControl.xaml

```

<Style TargetType="{x:Type graphsharp:EdgeControl}">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type graphsharp:EdgeControl}">

```

```
<Grid DataContext="{Binding RelativeSource={RelativeSource  
TemplatedParent}}" >
```

```
<Path
```

```
  MinWidth="1" MinHeight="1"
```

```
  ToolTip="{TemplateBinding ToolTip}"
```

```
  x:Name="edgePath">
```

```
<Path.Stroke>
```

```
  <Binding RelativeSource="{RelativeSource TemplatedParent}"
```

```
    Path="Edge.Type"/>
```

```
</Path.Stroke>
```

```
<Path.StrokeThickness>
```

```
  <Binding RelativeSource="{RelativeSource TemplatedParent}"
```

```
    Path="Edge.Thickness"/>
```

```
</Path.StrokeThickness>
```

```
<Path.StrokeDashArray>
```

```
  <Binding RelativeSource="{RelativeSource TemplatedParent}"
```

```
    Path="Edge.IsDashed"/>
```

```
</Path.StrokeDashArray>
```

```
<Path.Data>
```

```
<PathGeometry>
```

```
  <PathGeometry.Figures>
```

```
    <MultiBinding Converter="{StaticResource  
routeToPathConverter}">
```

```
<Binding RelativeSource="{RelativeSource TemplatedParent}"
    Path="Source.(graphsharp:GraphCanvas.X)" />
<Binding RelativeSource="{RelativeSource TemplatedParent}"
    Path="Source.(graphsharp:GraphCanvas.Y)" />
<Binding RelativeSource="{RelativeSource TemplatedParent}"
    Path="Source.ActualWidth" />
<Binding RelativeSource="{RelativeSource TemplatedParent}"
    Path="Source.ActualHeight" />
<Binding RelativeSource="{RelativeSource TemplatedParent}"
    Path="Target.(graphsharp:GraphCanvas.X)" />
<Binding RelativeSource="{RelativeSource TemplatedParent}"
    Path="Target.(graphsharp:GraphCanvas.Y)" />
<Binding RelativeSource="{RelativeSource TemplatedParent}"
    Path="Target.ActualWidth" />
<Binding RelativeSource="{RelativeSource TemplatedParent}"
    Path="Target.ActualHeight" />
<Binding RelativeSource="{RelativeSource TemplatedParent}"
    Path="RoutePoints" />
<Binding RelativeSource="{RelativeSource TemplatedParent}"
    Path="Edge.IsBidirectional"/>
</MultiBinding>
</PathGeometry.Figures>
</PathGeometry>
</Path.Data>
</Path>
```

```

    <Grid.ContextMenu>
        <ContextMenu ItemsSource="{ Binding
RelativeSource={RelativeSource TemplatedParent}, Path=Edge.Commands}">
            <!--<ContextMenu ItemsSource="{ TemplateBinding Vertex, }">-->
            <ContextMenu.ItemContainerStyle>
                <Style TargetType="MenuItem">
                    <Setter Property="Command" Value="{ Binding }"/>
                    <Setter Property="Header" Value="{ Binding
SimpleCommandText }"/>
                </Style>
            </ContextMenu.ItemContainerStyle>
        </ContextMenu>
    </Grid.ContextMenu>
</Grid>
</ControlTemplate>
</Setter.Value>
</Setter>
<Setter Property="graphsharp:GraphElementBehaviour.HighlightTrigger"
    Value="{ Binding RelativeSource={ RelativeSource Self},
Path=IsMouseOver}" />
<Setter Property="MinWidth"
    Value="1" />
<Setter Property="MinHeight"
    Value="1" />
<Setter Property="Background"

```



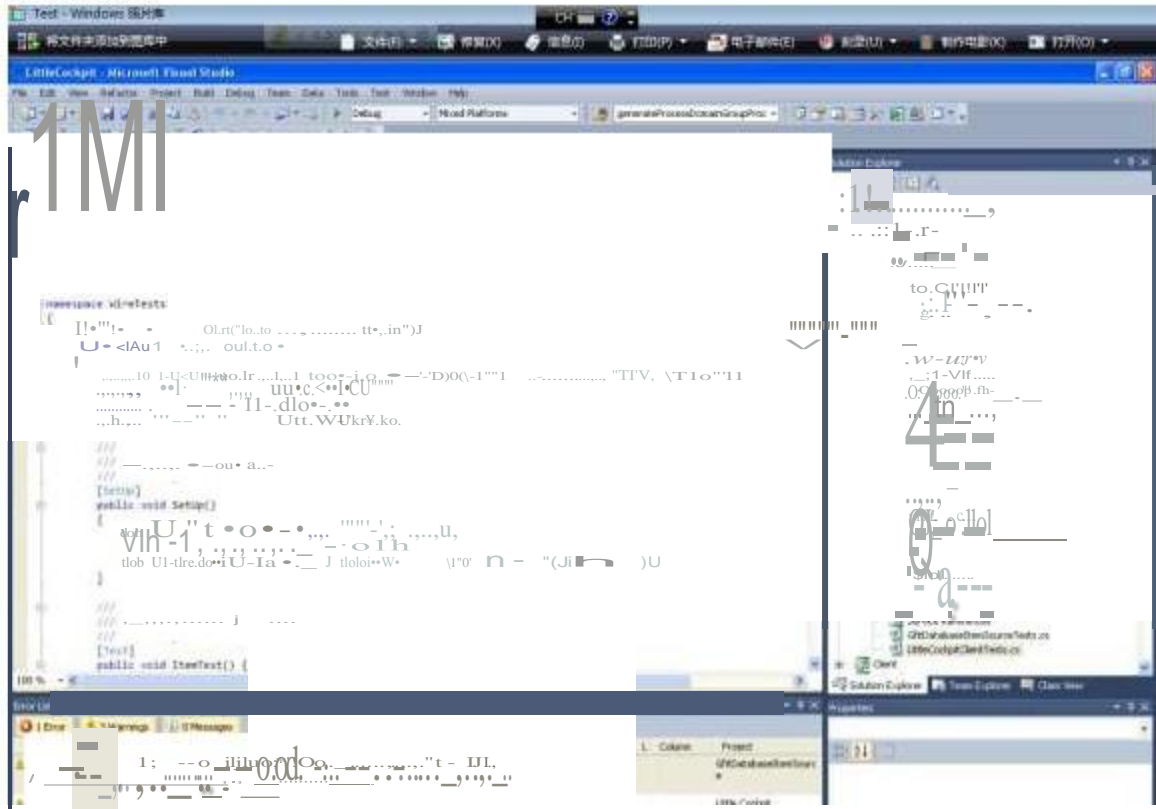
```

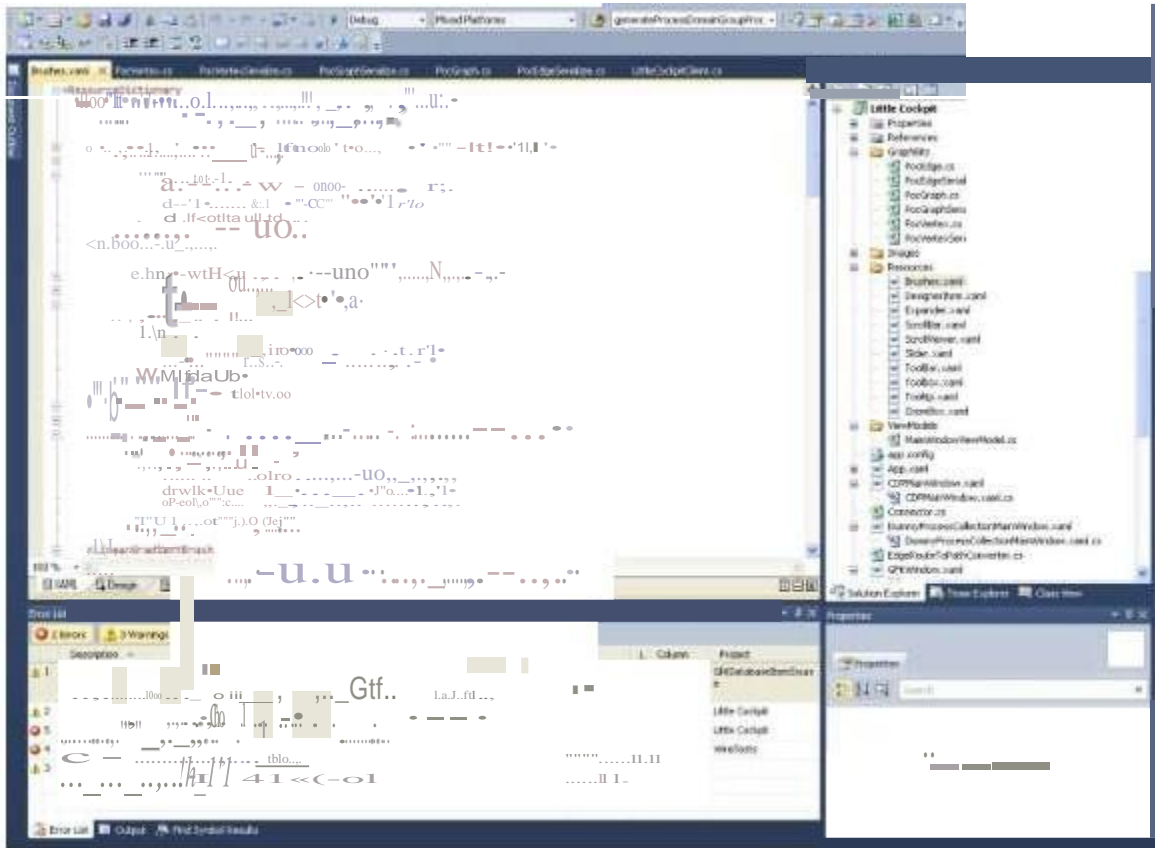
        Value="Red" />
    <Setter Property="Foreground"
        Value="Silver" />
    <Setter Property="Opacity"
        Value="0.5" />
    <Style.Triggers>
        <Trigger Property="graphsharp:GraphElementBehaviour.IsHighlighted"
            Value="True">
            <Setter Property="Foreground"
                Value="Black" />
        </Trigger>
        <Trigger Property="graphsharp:GraphElementBehaviour.IsSemiHighlighted"
            Value="True">
            <Setter Property="Foreground"
                Value="Yellow" />
        </Trigger>
        <MultiTrigger>
            <MultiTrigger.Conditions>
                <Condition
Property="graphsharp:GraphElementBehaviour.IsSemiHighlighted"
                    Value="True" />
                <Condition
Property="graphsharp:GraphElementBehaviour.SemiHighlightInfo"
                    Value="InEdge" />
            </MultiTrigger.Conditions>
            <Setter Property="Foreground"

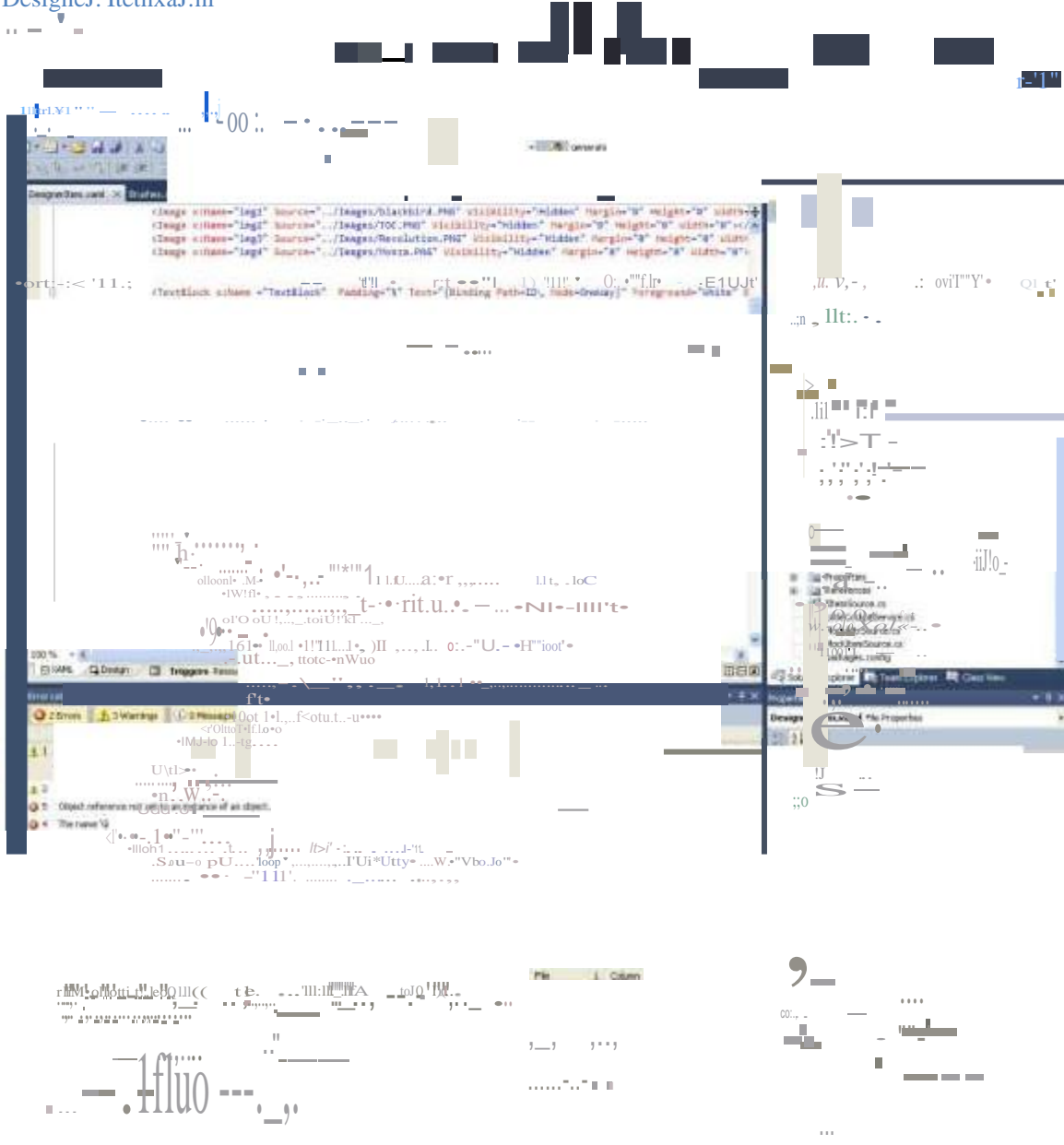
```

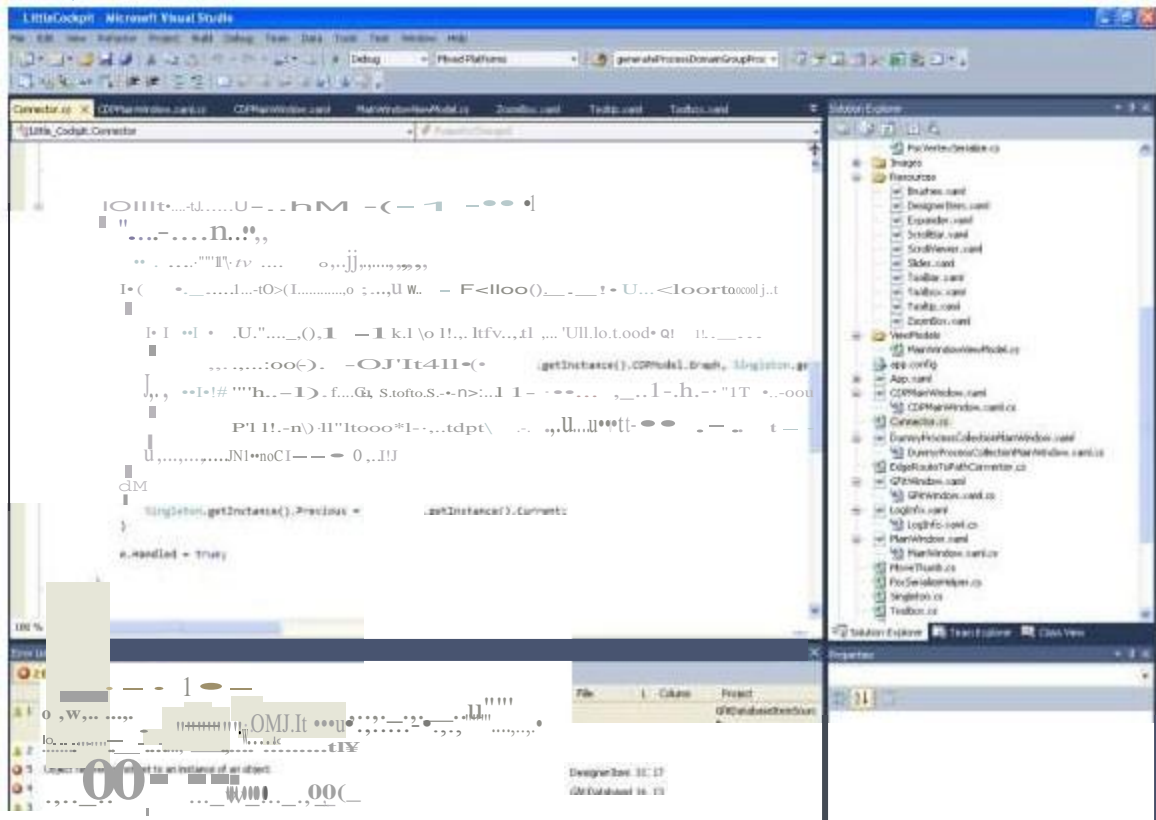
```
        Value="Red" />
    </MultiTrigger>
    <MultiTrigger>
        <MultiTrigger.Conditions>
            <Condition
Property="graphsharp:GraphElementBehaviour.IsSemiHighlighted"
                Value="True" />
            <Condition
Property="graphsharp:GraphElementBehaviour.SemiHighlightInfo"
                Value="OutEdge" />
        </MultiTrigger.Conditions>
        <Setter Property="Foreground"
                Value="Blue" />
    </MultiTrigger>
</Style.Triggers>
</Style>
```

ClientTests

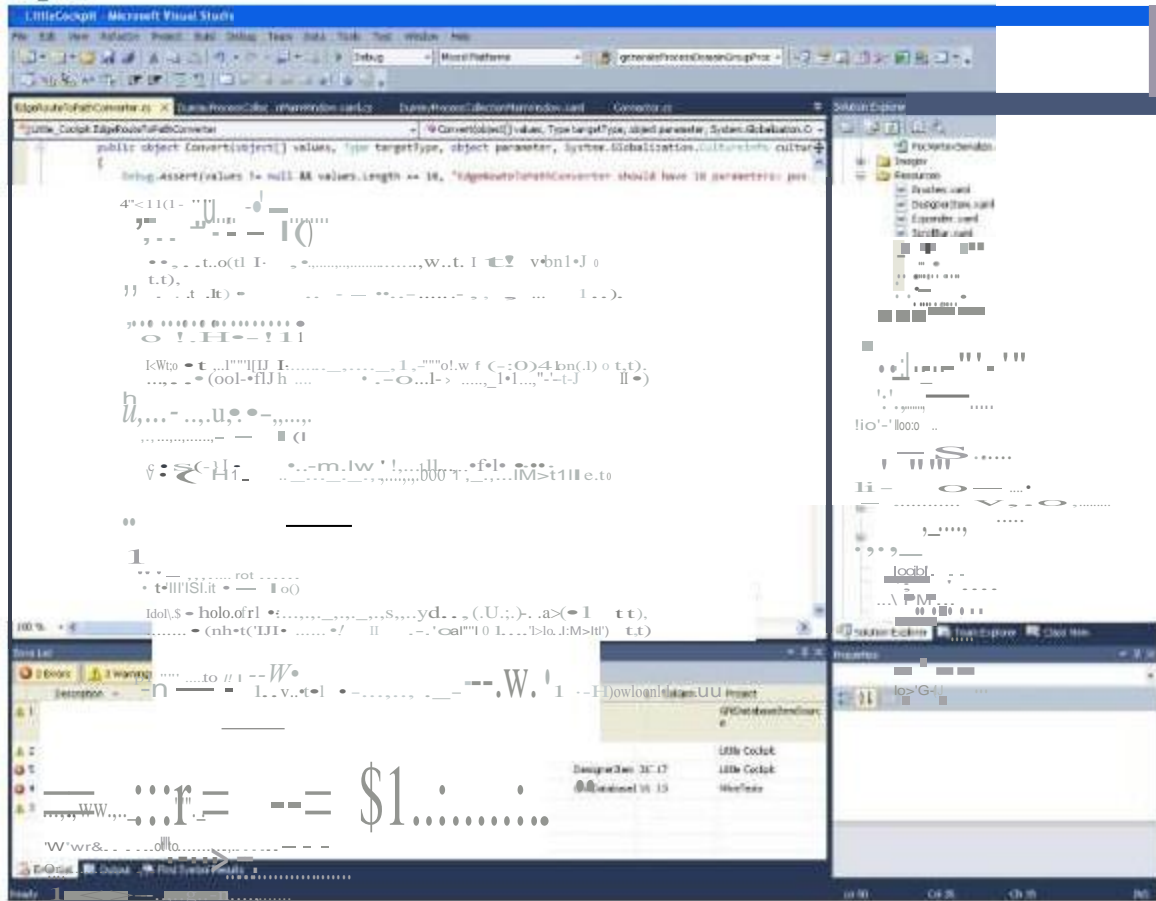




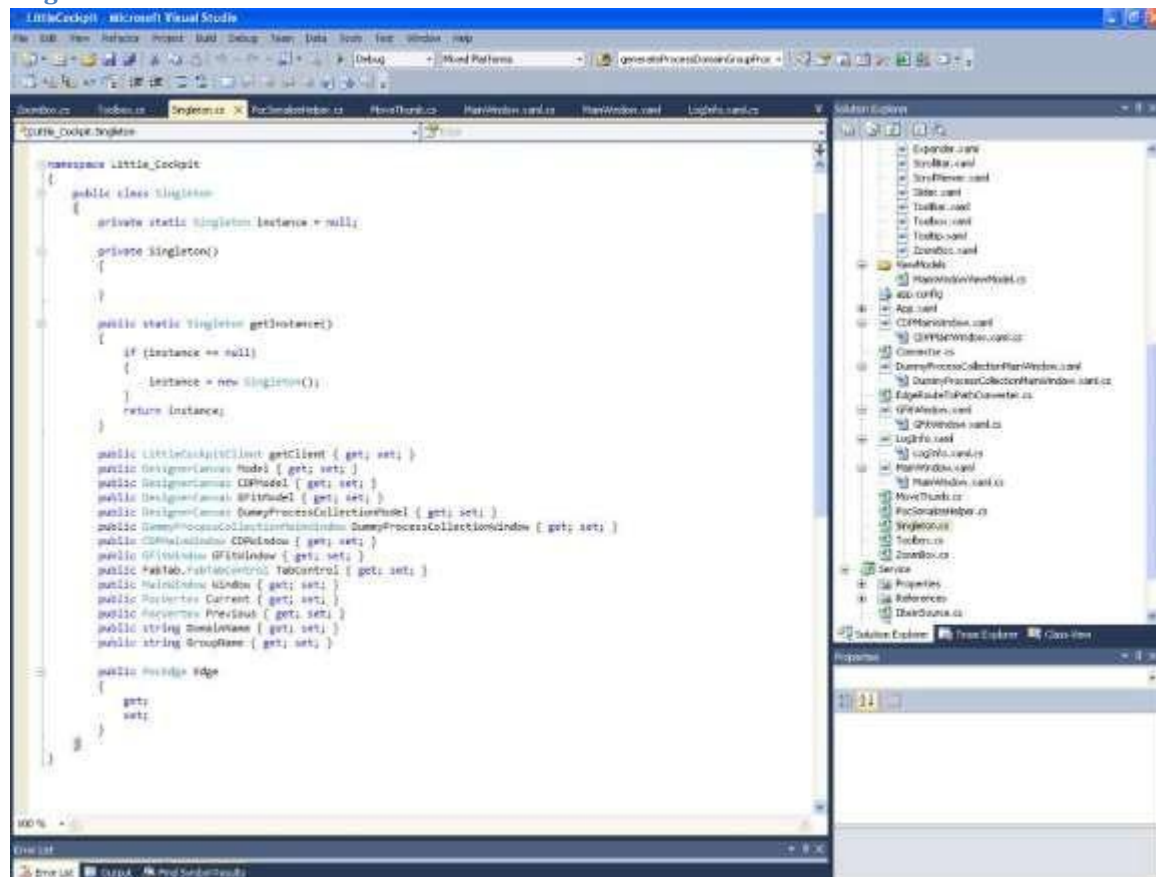




EdgeRouteToPathCon.vetteJ:".cs



Singleton.cs



```
namespace Little_Cookpit
{
    public class Singleton
    {
        private static Singleton instance = null;

        private Singleton()
        {
        }

        public static Singleton GetInstance()
        {
            if (instance == null)
            {
                instance = new Singleton();
            }
            return instance;
        }

        public LittleCookpitClient GetClient { get; set; }
        public DesignerCanvas Model { get; set; }
        public DesignerCanvas CIPModel { get; set; }
        public DesignerCanvas @IPModel { get; set; }
        public DesignerCanvas DummyProcessCollectionModel { get; set; }
        public DummyProcessCollectionForm DummyProcessCollectionWindow { get; set; }
        public CIPModelWindow CIPWindow { get; set; }
        public IPModelWindow IPWindow { get; set; }
        public WinTab.WinFormsControl.TabControl { get; set; }
        public MainWindow Window { get; set; }
        public HostPort Current { get; set; }
        public Assembly Preview { get; set; }
        public string DomainName { get; set; }
        public string GroupName { get; set; }

        public void Edge
        {
            get;
            set;
        }
    }
}
```


Appendix A.2 – Serialized Cockpit item

```
<?xml version="1.0" ?>  
┆ <ArrayOfCockpitItem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
┆ <CockpitItem>  
    <id>LONS00109273</id>  
    <type>Server</type>  
┆ <Attributes>  
┆ <Attr>  
    <attributeKey>Application</attributeKey>
```

```

: <attributeValue>
: <AttrValue>
  <attributeValueValue>Revolution</attributeValueValue>
  <attributeValueType>String</attributeValueType>
  </AttrValue>
</attributeValue>
</Attr>

: <Attr>
  <attributeKey>Description</attributeKey>
: <attributeValue>
: <AttrValue>
  <attributeValueValue>Revolution HA External Nirvana LIVE</attributeValueValue>
  <attributeValueType>String</attributeValueType>
  </AttrValue>
  </attributeValue>
  </Attr>

: <Attr>
  <attributeKey>Env</attributeKey>
: <attributeValue>
: <AttrValue>
  <attributeValueValue>Prod</attributeValueValue>
  <attributeValueType>String</attributeValueType>
  </AttrValue>
  </attributeValue>
  </Attr>

```

```

=> <Attr>
  <attributeKey>Agent_status</attributeKey>
=> <attributeValue>
=> <AttrValue>
  <attributeValueValue>agent is alive</attributeValueValue>
  <attributeValueType>String</attributeValueType>
  </AttrValue>
  </attributeValue>
  </Attr>
=> <Attr>
  <attributeKey>Location</attributeKey>
=> <attributeValue>
=> <AttrValue>
  <attributeValueValue>LON</attributeValueValue>
  <attributeValueType>String</attributeValueType>
  </AttrValue>
  </attributeValue>
  </Attr>
  </Attributes>
  </CockpitItem>
=> <CockpitItem>
  <id>reuters-autoquote-k1bpqq.us.net.intra</id>
  <type>Server</type>
=> <Attributes>
=> <Attr>

```

```
<attributeKey>Application</attributeKey>
: <attributeValue>
: <AttrValue>
  <attributeValueValue>Reuters Autoquote</attributeValueValue>
  <attributeValueType>String</attributeValueType>
  </AttrValue>
</attributeValue>
</Attr>
: <Attr>
  <attributeKey>Description</attributeKey>
: <attributeValue>
: <AttrValue>
  <attributeValueValue>NY MDFD Reuters Keystations-K1BPQQ</attributeValueValue>
  <attributeValueType>String</attributeValueType>
  </AttrValue>
</attributeValue>
</Attr>
: <Attr>
  <attributeKey>Env</attributeKey>
: <attributeValue>
: <AttrValue>
  <attributeValueValue>Prod</attributeValueValue>
  <attributeValueType>String</attributeValueType>
  </AttrValue>
</attributeValue>
```

```

    </Attr>
  : <Attr>
    <attributeKey>Agent_status</attributeKey>
  : <attributeValue>
  : <AttrValue>
    <attributeValueValue>agent is alive</attributeValueValue>
    <attributeValueType>String</attributeValueType>
    </AttrValue>
  </attributeValue>
  </Attr>
  : <Attr>
    <attributeKey>Location</attributeKey>
  : <attributeValue>
  : <AttrValue>
    <attributeValueValue>NYK</attributeValueValue>
    <attributeValueType>String</attributeValueType>
    </AttrValue>
  </attributeValue>
  </Attr>
  </Attributes>
  </CockpitItem>
  : <CockpitItem>
    <id>nycs00057562</id>
    <type>Server</type>
  : <Attributes>

```

```

: <Attr>
  <attributeKey>Application</attributeKey>
: <attributeValue>
: <AttrValue>
  <attributeValueType>String</attributeValueType>
  </AttrValue>
  </attributeValue>
  </Attr>

```

```

: <Attr>
  <attributeKey>Description</attributeKey>
: <attributeValue>
: <AttrValue>
  <attributeValueValue>Xiphias Swap</attributeValueValue>
  <attributeValueType>String</attributeValueType>
  </AttrValue>
  </attributeValue>
  </Attr>

```

```

: <Attr>
  <attributeKey>Env</attributeKey>
: <attributeValue>
: <AttrValue>
  <attributeValueValue>Dev</attributeValueValue>
  <attributeValueType>String</attributeValueType>
  </AttrValue>
  </attributeValue>

```

```

</Attr>
: <Attr>
  <attributeKey>Agent_status</attributeKey>
: <attributeValue>
: <AttrValue>
  <attributeValueValue>agent is alive</attributeValueValue>
  <attributeValueType>String</attributeValueType>
</AttrValue>
</attributeValue>
</Attr>

```

Appendix B – Items List and Attributes

Items	Items Attributes
Environment	<ul style="list-style-type: none"> • Name • Function/Owner • Hardware Location • Name in Config • Share • App server • DNS • Cube1/Cube2
Server	<ul style="list-style-type: none"> • Usage • OS • Location • Server • Cores • Memory
Process	<ul style="list-style-type: none"> • Status • Status Change Time • Enabled • Version • modifiedBy

	<ul style="list-style-type: none"> • TimeModified • Monitored? • Process Descriptor History • Invocation History • Status History • Log • Service history
Process descriptor	<ul style="list-style-type: none"> • Name • Sub Environment • Group • Location • CommandLine(Name of Service) • Is service? • Machine • Schedule • RetrySchedule • KillAfter • Is Wire Service • Mf Service Id • Mf Heartbeat Subject • Working Directory
Alerts	<ul style="list-style-type: none"> • Type • Subject • State • Creation Time • Last Modified Time • Severity • Instance number • Login • Date • STAR LON/ STAR TYO/STAR SIN • Version • Owner • Server Name • Description •
Heartbeat	<ul style="list-style-type: none"> • Subject • Server Name • Server id • Process name • Service Type • Expected HBs • Received HBs • Last HB

	<ul style="list-style-type: none"> • Status
--	--

Appendix C – Timeline

10/24/11(Monday)-10/28/11(Friday)	Get familiar with the existing monitoring tools, set up communication devices , request all the potentially used softwares, investigate graphing tool technology and gather user requirements
10/31/11(Monday)-11/4/11(Friday)	MileStone1: Define the scope of environment cockpit
11/7/11(Monday)-11/10/11(Thursday)	MileStone 2:Configure the Wire environment
11/10/11(Thursday)-11/15/11(Tuesday)	MileStone 3: Build basic frame of environment cockpit user interface prototype
11/16/11(Wednesday) - 11/ 21/11(Monday)	Milestone 4: Add simple arrows Report: Background first draft
11/21/11(Monday)	Finish arrow toolbox
11/22/11(Tuesday) – 11/23/11(Wednesday)	Finish adding different types of arrows
11/24/11(Thursday) -11/25/11(Friday)	Milestone 5: Delete arrows Report: Background finished, Current difficulties first draft
11/28/11(Monday) - 11/29/11(Tuesday)	Finish Pop up box
11/30/11(Wednesday)	Milestone 6 : Show information in the detailed information box

12/1/11(Thursday) -12/2/11(Friday)	Organize code Finish save and load , design user interface with UX team Report: Current difficulties finished, Outcome and methodology first draft
12/2/11(Friday)	Finish the project requirement!!!
12/5/11(Monday) – 12/9/11(Friday)	Refine codes &add additional features Finish Filter ,tab view, compound view improve the overlook look of the user interface Report: Finish Report first draft.
12/12/11(Monday) – 12/14/11(Wednesday)	Wrap up presentation Report: Finish Report by 12/14/11

Appendix D – Structured Finance

Definition

Structured Finance is a broad term. A summarization of some commonly used concepts is that structured finance is used to describe a sector of finance that was created to help transfer risk using complex legal and corporate entities. Typically the process of risk transferring involves activities that will not heavily affect the balance sheet. The term “structured finance” implies that the corporations or entities are using special purpose entities or special purpose vehicles (SPE or SPV) to exchange future cash flow from an existing asset or portfolio while financing the corporation or entity leveraged by the asset. This risk transfer as applied to securitization of various financial assets (e.g. mortgages, credit card receivables, auto loans, etc.) has helped to open up new sources of financing to consumers. However, the asset allocation from certain assets of corporation to cash flows in the balance sheet is risky, as the increasing cash flow comes from the condition that the asset-liability ratio is unchanged. Structured finance also includes the innovation of new financial instruments which allows for re-transfer funds to investors (as asset-backed securities)

Classification

Usually, structured finance is divided into two categories: Asset Financing and Capital Financing. Asset Financing is further divided into Current Asset and Fixed Asset.

Examples of Asset Financing:

Current Asset classes: cash financing (a loan-deposit); accounts receivable financing (factoring, payments); inventory financing (warehouse financing); order financing (credit packing, red clause letter of credit), etc.

Fixed Assets classes: mortgage, hire purchase, and finance lease of fixed assets.

The main types of Capital Financing are stock and equity financing; swaps; leveraged buyouts

Terminology used in this section:

Factoring: A business sells its accounts receivable to a third party at a discount rate in order to hasten the finance process.

Warehouse financing: A form of inventory financing. Loans are usually made to manufacturers and processors on the basis of goods or commodities held in trust as collateral for the loans.

Credit packing: A loan given to the beneficiary by the bank to enable the individual to purchase raw materials. The beneficiary is usually requested to deposit the DC with the bank as security.

Red clause letter of credit: A specific type of letter of credit which enables a buyer to extend an unsecured loan to a seller. Red Clause Letters of Credit permit documentary credit beneficiaries to receive funds for any merchandise outlined in the letter of credit.

Hire purchase: A persons usually agree to pay for goods in parts or a percentage at a time at an amount of interest.

Leverage buyout: A type of acquisition that acquires a controlling interest in a company's equity with a small amount of cash flow, but a significant percentage of the purchase price is financed through leverage (borrowing)

Specification

Companies can finance themselves in a variety of ways based on different financing structures, but mainly by traditional finance or structured finance. Traditional finance is based on assets, while structured finance is based on credit: it fulfills the purpose of financing by constructing a rigorous transaction model. Companies can use integrated finance methods which may involve both traditional finance and structured finance. The best capital structure allows the limited assets to generate maximum value of present cash flow.

Traditional ways of financing are usually achieved by increasing corporate debt (debt financing) and increasing equity (equity financing) two ways. Debt financing and equity financing reflect the activities of the both left-hand-side and right-hand-side of the balance sheet. Structured finance, unlike the other two, mainly involves with relatively small activities on the balance sheet. Traditional finance involved mainly with fixed assets, such as a house or a newly started company, while structured finance mainly involves with financial assets, such as securities or derivatives.

In order to better illustrate the differences between traditional finance and structured finance, here is a scenario of a small bank: this bank grant loans to multiple individuals, and then the bank uses these loans to construct CDO products and sell these CDOs to investors. In the process, the bank realizes it has problems with its cash flow, so the executives of the bank decide to borrow money from another institution. The borrowing and loans here are ways of traditional finance, and CDOs are examples of structured finance. The observation towards changes in the balance sheet is provided:

The balance sheet of the bank at the beginning would be like this. They have large amounts of accounts receivable, meaning that the bank has released many mortgage loans to its clients:

Balance Sheet							
							22-Feb-12
Assets							
	Cash						\$5,000,000.00
	Accounts Receivable (AR)						\$20,000,000.00
	Property, Plant and Equipment (PP&E)						\$5,000,000.00
	Total Assets						\$30,000,000.00
Liabilities							
	Accounts Payable (AP)					\$5,000,000.00	
	Debt					\$0.00	
	Total Liabilities					\$5,000,000.00	
Shareholders' Equity							
	Common Stock and Additional Paid-In Capital (APIC)					\$10,000,000.00	
	Retained Earnings					\$15,000,000.00	
	Total Shareholders' Equity					\$25,000,000.00	
	Total liabilities and shareholders' Equity					\$30,000,000.00	

Figure 1: The initial balance sheet

All of a sudden, the bank has a new business opportunity, but the deal requires a 20 million dollar investment. Unfortunately, due to its poor liquidity, the bank doesn't have \$20m cash on hand. In order to secure this profitable deal, the bank turns to a mutual fund MFLA and borrows \$20m from the MFLA. Therefore the bank has more cash now, as well as accounts payable (for the easiness of the balance sheet, the interest is set to 0 here):

Balance Sheet							
							23-Feb-12
Assets							
	Cash						\$25,000,000.00
	Accounts Receivable (AR)						\$20,000,000.00
	Property, Plant and Equipment (PP&E)						\$5,000,000.00
	Total Assets						\$50,000,000.00
Liabilities							
	Accounts Payable (AP)					\$25,000,000.00	
	Debt					\$0.00	
	Total Liabilities					\$25,000,000.00	
Shareholders' Equity							
	Common Stock and Additional Paid-In Capital (APIC)					\$10,000,000.00	
	Retained Earnings					\$15,000,000.00	
	Total Shareholders' Equity					\$25,000,000.00	
	Total liabilities and shareholders' Equity					\$50,000,000.00	

Figure 2: Balance sheet after the bank received money from the mutual fund

Everything is going smoothly, and the bank is waiting the mortgage payment from those individuals. However, executives at the bank heard bad news about their clients. If their clients are about to default, the bank will experience heavy losses and may even go bankrupt. The balance sheet for the bank after the default would be like the following (they might experience heavy losses in their retained earnings and common stock, but that’s not the topic we are concerned with here):

Balance Sheet			23-Feb-12
Assets			
	Cash		\$25,000,000.00
	Accounts Receivable (AR)		\$0.00
	Property, Plant and Equipment (PP&E)		\$5,000,000.00
	Total Assets		\$30,000,000.00
Liabilities			
	Accounts Payable (AP)	\$25,000,000.00	
	Debt	\$0.00	
	Total Liabilities	\$25,000,000.00	
Shareholders' Equity			
	Common Stock and Additional Paid-In Capital (APIC)	\$10,000,000.00	
	Retained Earnings	(\$5,000,000.00)	
	Total Shareholders' Equity	\$5,000,000.00	
	Total liabilities and shareholders' Equity	\$30,000,000.00	

Figure 3: If those clients default, the bank will have its account receivable sets to \$0

A \$20m loss is unaffordable for a small bank. The executives of the bank won’t allow the defaults to happen. But what can they do to reduce its loss?

Here is where the CDO becomes crucial. The executives of the bank are very smart, and they have thought about the probability of default in the past. In early days, they have created a CDO product and put it into the market. The CDO has following rules: if the clients didn’t default, whoever has the CDO would enjoy the free cash; but if the clients default, then the holder of CDO would pay back the loss of the bank.

Investment bank IBLB has shown great interest in the very beginning. So the small bank had a deal with IBLB, with the price of CDO was finalized at \$1m. Now it is time for IBLB to pay the losses of the bank, which is \$20m. Clearly, without the help of CDO, the bank could be bankrupted because of the default of its clients.

The last balance sheet would be look like this:

Balance Sheet							
							23-Feb-12
Assets							
	Cash						\$24,000,000.00
	Accounts Receivable (AR)						\$20,000,000.00
	Property, Plant and Equipment (PP&E)						\$5,000,000.00
	Total Assets						\$49,000,000.00
Liabilities							
	Accounts Payable (AP)					\$25,000,000.00	
	Debt					\$0.00	
	Total Liabilities					\$25,000,000.00	
Shareholders' Equity							
	Common Stock and Additional Paid-In Capital (APIC)					\$10,000,000.00	
	Retained Earnings					\$14,000,000.00	
	Total Shareholders' Equity					\$24,000,000.00	
	Total liabilities and shareholders' Equity					\$49,000,000.00	

Figure 4: Last balance sheet

Compare figure 1 and 2, we can see that the balance sheet has changed a lot during a traditional finance process. However, compare figure 2 with figure 4, we could see that balance sheet only changed a little. Here is one of the most important advantages of structured finance: it allows the corporate to use their limited assets to generate great revenue.

Advantages of structured finance are list below:

1. Provide clients long term future cash flows
2. Improve the client's asset turnover ratio

3. Reduce client's asset liability ratio (assuming the structured finance product will not 100% default)
4. Credit enhancement, lower financing costs, diversifies investment products for investors.
5. Self-liquidating, one of the most important specifications of structured finance. Unlike other finance methods, structured finance products typically do not require additional mortgages or warranties (or requires a little), allows corporate to use their limited mortgage assets into other finance activities.

Process

Typically, a structured finance product will go through following process:

1. Divestiture of current assets, identify the suitable vehicle for issuing the bond/security, establish the asset pool.
2. Finish the writing for required documents for offering, complete every preparation work for issuing.
3. Obtain the approvals from the central bank and other regulatory authorities to issue bonds and/or securities.
4. Debt/equity/mortgage pay back;
5. Registration for claims and liabilities, disclosure of obligatory information according to the agreement.

Parts from the above, specific tasks might require different processes and operations. For example, partial credit guarantees usually involves with the lender utilizes their credit rating to help clients expanding their financing channels, while securitization includes the

collection and the actual sale of certain financial assets, then issue securities which aims to generate cash flows used to repay these assets.

No matter what product is constructing, the credit supply process is always the key factor in the whole process. Any potential instability of credit supply process would jeopardize the whole product severely.

Environmental Requirements

The United States has a good tradition of growing structured finance market and highest revenue among the world. If we looking back the U.S. structured finance market in terms of history, we can conclude that there are eight elements have played an important role to maintain the growth potential of both the market and revenue. These factors often complement each other, and development of one element could promote the development of other elements, therefore improve the overall quality of U.S. structured finance system.

Well-established regulatory system

There should be a well-established legal framework to protect investors and their legitimate underlying assets for each securitization transaction. Most importantly, when the sponsor / seller in the situation bankruptcy, the law/regulation must protect investors' right to recover the asset and cash flows in the securities of the bankrupted entities. Therefore, speaking of securitization transactions, a special purpose entity which will not bankrupt and have good credits must be established. In addition, the regulation and laws should have clear items about the responsibilities and obligations among issuers, trustees, credit managers and people from service side.

Accurate analysis of cash flows

From the perspective of financing, securitization is essentially a process to help the

credit side to raise an amount of capital equals to present value of current assets in future cash flows after deducting the cost of securitization. In the beginning of the securitization, the issuer must conduct a careful cash flow analysis to determine whether the special purpose entity could fully repay the debt on time.

To conduct an analysis about pricing of future cash flow requires a lot of prerequisite assumptions. In addition to using assumed discount rate to calculate present value and pricing of future assets, a well-rounded cash flow analysis must have their own assumed indexes, such as the general economic conditions within the loan period, borrower's ability to repay loans, and the probability of default. Only by researching a large amount of historical data can the researcher be able to come up with reliable assumptions for analysis of cash flows.

Clear and reliable accounting activities

During the process of asset securitization, there might be three relatively important accounting issues: firstly, the accounting activities towards the interest income of securities. As the securities issued by special purpose entities to which the loans have interest, it should be taxed on the special purpose entity accordingly. In order to avoid double taxation, as long as the special purpose entity meet the requirement of being a "grantor trust ", it will be deemed "grantor trust" and will be granted federal tax exemption in the United States; secondly, to protect investors by clarifying the periodic cash flow between the loans (including interest and principal) of the accounting records.; thirdly, all cash flows generated by transactions should be subject to the strict inspection from professional accountants.

Accredited public rating organizations

As the securities issued from lender are supported by its sponsored loans or commercial loans, stock investors will be more concerned about credit risks of loans and corresponding securities. For example, the housing mortgage-backed securities are credited by

the Government National Mortgage Association (Finnie Mae), Federal National Mortgage Association (Fannie Mae) and Federal Home Loan Mortgage Corporation (Freddie Mac). If any housing mortgage-backed securities, commercial mortgage-backed securities and asset-backed securities didn't receive credit from any of these three institutions, then either the security will try a variety of ways to improve its credit rating, or the investors will be in big trouble.

In general, there are four meaningful ways to improve one's credit rating: first/secondary structure of cash flow (or over-collateralization), parent company guarantee, Guarantee Bond (Performance Bond), and letters of credit. Every security wants to get triple-A rating from big 3 credit rating agencies, which are Standard & Poor, Fitch and Moody's.

Comprehensive Investment Banking Service

IPO's and sales of new securities underwriting are the responsibilities of investment bankers. Investment banks acts like a bridge between issuers and investors. However, during the securitization process, the investment banks are essential to the success of a securitization. Investment Bank is responsible for coordinating and helping issuers to deal with legal, accounting, tax and cash flow analysis issues. In addition, investment banks also play the role of dealers: securities pricing, purchasing all issued securities and sold them to individual and corporate investors. After the IPO, investment banks are served as the "market maker" in the secondary market: they actively trade securities in order to ensure the liquidity.

Healthy Treasury Bills and Notes market

In order to ensure the healthy development of securities markets, a healthy and stable bonds and treasury bills market is necessary. Because the government bonds are almost risk-free (exception includes bond governments in turmoil such as Greek Government and Italian Government), so the trading of such bonds on the market will be the beacons for the risk-free

return rates in different terms. In terms of risk-free, the curve that describes the relationship between terms and yield was called the 'U.S. Treasury Yield Curve'. This yield curve provides a benchmark for the pricing of fixed-income securities with different risk and different terms in the primary market and secondary market.

Active secondary market

A successful primary market securities must supported by an active secondary market. An active and healthy secondary market should provide useful information for the upcoming IPOs and pricing information. Once after the issue of new securities, investors need an active secondary market to trade securities and securities need these active markets for its own liquidity. By doing so, investors will be able to trade in a relatively stable the market.

However, it should be noted is that investors are the key to a very active secondary market. If investors in a market simply hold securities rather than engage in any transaction, it will be very hard for those investment banks to keep the active of the market. To rely on investment banks to ensure the liquidity in the secondary market is very difficult.

Diversified investors

An important factor that contributes the success in the current structured finance market is the rapid growth of investor groups. Investor group includes various types of investors, from short-term money market investors, to the portfolio managers of commercial banks, or the long-term pension fund managers.

In addition, with the increasing globalization of capital market development, foreign investors will be more and more important. Also, the success in the current structured finance market can also be attributed to the innovative development of structured finance securities themselves. With products developed in the different kinds, different credits levels and different terms, structured finance securities are able to meet the demand from all investors.

High-yield securities can attract "income-oriented" investors, securities with various period of maturity can attract the "term-oriented" investors, and securities with different levels of credit rating can attract those "credit-oriented" investors.

Valuation and Assessment

This section will explain on a macro level of how to evaluate a structured finance product. There will be a section later to detailed explain the valuation of specific products.

Typically, the target being evaluated is an amount of assets of mortgage securities. As discussed above, the operation of a structured finance product usually doesn't involve the with the management activities of original asset holder, therefore a key element to value a structured finance product is to separate the credit/mortgage with original assets.

Besides, there are certain factors that may affect the pricing model of a structured finance product. First of all, the quality of the collateral adequacy matters. Usually the better of quality of collateral, the better of asset turnover ratio is, the better of previous payment history is, and the better the product is. In most cases, the quality of product contributes most in the pricing of structured finance. In addition, the structure of involved transaction's cash flow determines. Investors need their money before the deadline, so the better cash flow structure the transaction associates, the more likely the investor would like to pay for a higher price. Finally, regulation system is a crucial part of valuing a structured finance product, because of those the legal and tax considerations. The legitimacy of collateral representatives', the legitimacy of the cash flow, impacts of cash flow from tax perspectives, everything above will affect the pricing and assessment of a structured finance product. Besides, the political economy will affect the valuation of the product. Generally, in a political stable nation, the price of a structured finance product would be higher than those of

political unstable countries.

Futures Contracts

Future contract is by definition a standardized contract between two parties to exchange a specified asset of some quantity and quality for a price agreed today. The assets are going to be delivered at a specified future date. This contract itself theoretically costs nothing to enter. The buyer expects that the price to increase which means in the future they can receive the asset at cheaper price than the future market price. However the seller wants the price to decrease. If so, in the future the seller will be able to sell their asset above the future normal asset price. Because the future contracts are standardized, they are traded in exchanges. The future contracts are introduced by Japan in 1930s and became very popular later on. The underlying asset used to be traditional commodities. Now the assets expand to currencies, securities, financial instruments and intangible assets.

Here is a timeline to better demonstrate future contracts.

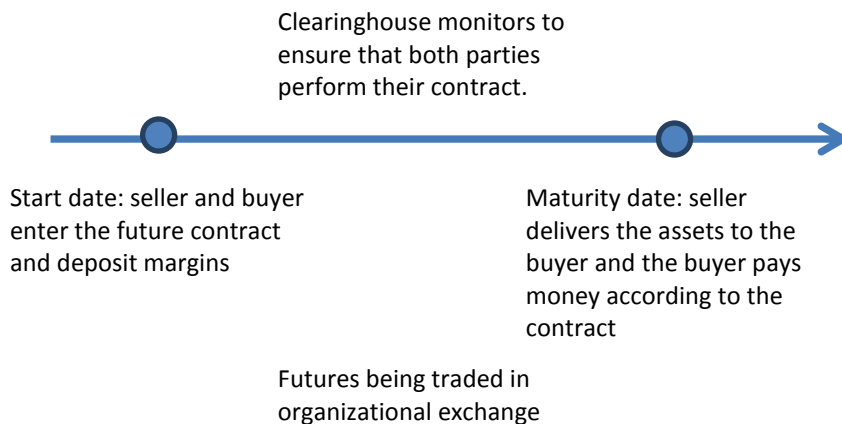


Figure 5: Timeline for future contracts

In the current finance world, the future contracts are carried out in the following way. Even though we mentioned before that theoretically the future contract costs nothing to enter, in real world, in order to minimize the counterparty risk, both buyer and seller need to deposit some money, called margin, which is normally proportional to the asset that they are

contracted on. Since future contract is marked to market daily. The daily profits and losses will be shown in the traders' account operated by a clearinghouse. At the settlement day, the futures will be settled by either commodities or cashes.

Pricing Model

$$F(t) = S(t)e^{r(T-t)}$$

Here T means maturity and r is risk-free return. F(t) is the expected future price of the asset and S(t) is the current price of the asset. This formula assumes continuous compounding and the future asset price equals to the current asset price with continuous compound interest.

Risk Exposure

In addition to the fluctuation of the market price of the assets, both parties in the future contract are subject to counterparty risk, which is the risk that the other party doesn't deliver the goods or doesn't pay the money according to the contract. And a clearinghouse is the one who guarantees both parties performance.

Some Commonly Traded Future Contract

Eurodollar CD future's underlying instrument is 3-month (90 days) Eurodollar CD. It is currently traded on Chicago Mercantile Exchange and London International Financial Futures Exchange. It has \$1 million face value with cash settlement contract. This means at the maturity date, the Eurodollar CD future is settled in cash for the value of a Eurodollar CD based on London Interbank Offered Rate - LIBOR. Many people use Eurodollar CD futures for hedging (Fabozzi, 24).

Interest Rate Swaps

Interest rate swap is a contract for two parties who agree to exchange interest payments during certain future period. Normally one party agrees to pay a fixed interest

payment periodically. They are called fix-rate payers and while the other party will pay at a floating rate according to some reference rate (the most commonly used reference rate is London Interbank Offered Rate – LIBOR). They are called fixed-rate receivers. The interest rate swap is an over-the-counter instrument, and thus has a lot of varieties. It can be in different currencies and except for the previously mentioned most common kind of fixed-for-floating-rate swap. There is also floating-for-floating-rate swap. The interest rate swap is majorly used by companies who desire to change its financing structure from fixed rate to floating or from floating to fixed-rate in order to reduce funding costs. Interest rate swaps are a very popular instrument and it is now the largest component of global over-the-counter derivative market.

The timeline is demonstrated as below:

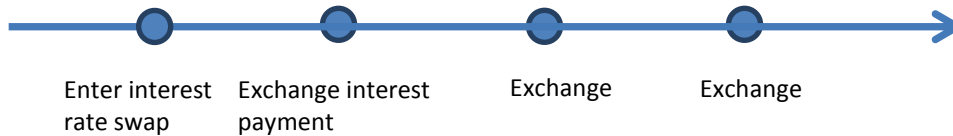


Figure 6: Timeline for interest rate swaps

When interpreting the interest rate swap, you can view it as a package of forward contracts with each payment as one forward contract between the fixed-rate receiver and fixed-rate buyer. Interest rate swap can also be viewed as a package of cash market instruments. For example, interest rates swap which exchanges LIBOR rate with 10% fixed interest rate paid annually for 5 years. An investor as a fixed rate receiver entering this interest rate swap equals to buy a 5-year fixed rate bond and financing this purchase by borrowing the notional amount of money for 5 years with LIBOR rate interest paid every year.

Pricing Model

The value of the fixed leg is:

$PV_{\text{fixed}} = C * \text{present value of the sum of the future payments (C is the swap rate)}$

$$PV_{\text{fixed}} = C \times \sum_{i=1}^M \left(P \times \frac{t_i}{T_i} \times df_i \right)$$

Here P is the notional amount, t_i is the number of days in period i, T_i is the basis according to the day count convention (It may possibly be 365 or be 360 for calculation convenience) and df_i is the discount.

For the present value of floating interest rate, since we don't know what the future interest rate will be. Thus we predict the future interest rate from forward rates which are derived from the yield curve. And the value of the floating leg will be:

$$PV_{\text{float}} = \sum_{j=1}^N \left(P \times f_j \times \frac{t_j}{T_j} \times df_j \right)$$

In this formula, N is the number of payments, F_j is the forward rate, P is the notional amount, t_j is the number of days in period j, T_j is the basis according to day count convention and df_j is the discount factor. The present value of both fixed and floating legs are essentially the sum of the present value of future payments. And when the PV_{fixed} and PV_{float} equal to each other, there will be no upfront payment from any party.

Risk exposure

There are two kinds of risks that traders may expose to. One is interest rate risk, which related to the fluctuation of the interest rate and the other is counterparty risk, which mainly concerns about the in-the-money party facing the risk of possible default of the other party. Since interest rate swap is traded over-the-counter without any clearinghouse in between. The counterparty risk of defaulting may be significant (Fabozzi, 26).

Case study – how interest rate swap benefit both parties

Next we are going to demonstrate a very interesting case to better explain the incentives of entities which are willing to involve in an interest rate swap.

There are two corporations. Corporation Good is a great company and it was rated A by more than 8 rating agencies. It issues a \$1000 million fixed-rate bonds for 5 years at 6%. While Corporation Bad is not very promising and it was rated C- by rating agencies last year. Thus Corporation Bad can only issue high-yield debt, borrows \$1000 million from a bank at 6-month LIBOR plus 2%. Here is some more background information. The interest rate that must be paid by the Corporation Good and Corporation Bad in both floating rate and fixed-rate markets are as below. For Corporation Good, it needs to pay 6-month LIBOR + 40 basis points in floating rate market and 6 percent in fixed rate market; while for Corporation Bad, it needs to pay 6-month LIBOR + 200 basis points in floating rate market and 10 percent in fixed rate market.

Now a very smart financial analyst of Corporation Good figures out a way to lower its funding cost by swapping into floating rate debt and another very brilliant financial analyst of Corporation Bad also sees an opportunity to lower its cost by entering this interest rate swap that is offered by Corporation Good. Their interest rate swap is agreed as following. For Corporation Good, it is going to pay floating rate of 6-month LIBOR and to receive fixed rate of 6.2%; for Corporation Bad, it is going to pay fixed rate of 6.45% and receive floating rate of 6-month LIBOR. The 0.25% that is paid by Corporation Bad but is not received by Corporation Good is for swap dealer. Let's hear the reasons from the two smart financial analysts.

For Corporation Good, it needs to pay fixed-rate bonds issued of 6% and the interest rate swap of 6-month LIBOR. It will then receive 6.2% from the interest rate swap. Thus in total it pays $6\% + 6\text{-month LIBOR} - 6.2\% = 6\text{-month LIBOR} - 20\text{bp}$. This number is lower compare to what Corporation Good should pay within floating rate market of 6-month LIBOR + 40bp as we mentioned in the background. Thus the interest rate swap is a benefit contract

for Corporation Good.

For Corporation Bad, it needs to pay 6-month LIBOR + 200 bp to the bank and 6.45% to the swap dealer. It will receive 6-month LIBOR from Corporation Good. Thus in total it pays 6-month LIBOR + 200 bp + 6.45% - 6-month LIBOR = 8.45%. This is also a number lower than what Corporation Bad should pay within fixed rate market of 10% as we mentioned before in the background. By entering the interest rate swap contract, although both company exposed to certain amount of counterparty risk, they can achieve lower financial costs than by directly borrowing from the market and this is the fascinating power of the interest rate swap (Fabozzi, 33).

Options

An option is a contract that grants an option, not an obligation for the option buyers to buy or sell things such as commodities at a specific rate to the option sellers, also called as option writer, at a specific future time. The buyer will need to pay premium to the option writer in order to have this kind of option in the future. The specific rate that is agreed upon the option contract is called strike price and the specific future time when the seller needs to exercise the option is called the expiration date.

The timeline is demonstrated as below:

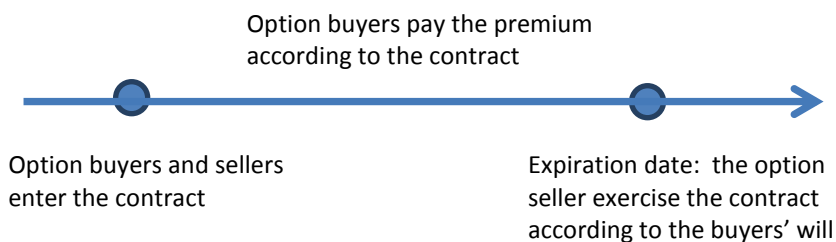


Figure 7: Timeline for options

Options can be classified by four different ways. First, if an option grants buying right

to the option buyer, it is called a call. If an option grants selling right to the option buyer, it is called a put. Second, if an option can be exercised at any time up to the expiration date, it is called an American option. If it can only be exercised at the expiration date, it is called a European option. If the early option is possible but restricted, it is called a Bermuda option, which is a hybrid between American and European options. Third, there are also exchange-traded options and over-the-counter options. Exchange-traded options have standardized contract and there is a clearinghouse connecting the buyers and sellers. Over-the-counter options are tailor-made for big corporation or institution and usually investment banks and commercial banks will act as brokers in over-the-counter market. These options can be very complex and less liquid than the exchange-traded options. As the option product evolved, in order to compete with over-the-counter market, the flexible Treasury futures option was introduced in 1994 in Chicago Board of Trade (CBOT) which allow investors to customize option with certain limitations. Fourth, options can be classified according to its underlying. If the underlying is a fixed-income security, then the option is called options on physicals. If the underlying is interest rate futures, the option is named as futures options (Fabozzi, 36).

Pricing Model

The value of the options is the intrinsic value, which is the economic value if the option is exercised immediately. For example, for a call option, if the current price of the underlying security is smaller than the strike price, the intrinsic value is negative and vice versa. A more mathematical way of valuing an option is by using the famous Black-Scholes option pricing model, which gives the valuation for European-style options. The most important concept to derive the Black-Scholes option is that we can reconstruct an option by a risk-free bond and a stock and we can get the same pay-off as the given option. Below are some important assumptions:

- The stock price follows a geometric Brownian motion process
- There is no transaction costs or taxes
- No dividends during the life of the option
- No risk-free arbitrage opportunities

Let's say the price of the option is a function of S_t , the price of the stock, and t , the time. We represent the option price as $c(S, t)$. The risk-free bond B with risk-free rate r will hold: $dB = rBdt$ and the stock with geometric Brownian motion will hold: $dS = \mu Sdt + \sigma Sdz$. According to Ito's lemma¹, the option price will hold:

$dc = \left(C_t + \mu SC_s + \frac{1}{2} \sigma^2 S^2 C_{ss} \right) dt + \sigma SC_s dz$. First we need to reconstruct an option with x shares and y bond.

$$c = xS + yB$$

Different the above formula, we get:

$$dc = xdS + ydB = (x\mu S + yrB)dt + x\sigma Sdx$$

With the formula for dc above, we get two equations as results:

$$(x\mu S + yrB) = \left(C_t + \mu SC_s + \frac{1}{2} \sigma^2 S^2 C_{ss} \right)$$

$$x\sigma S = \sigma SC_s$$

We can easily get $x = C_s$. Plugging this into $c = xS + yB$, we get $y = \frac{1}{B}(c - SC_s)$. Plugging

both $x = C_s$ and $y = \frac{1}{B}(c - SC_s)$ into $(x\mu S + yrB) = \left(C_t + \mu SC_s + \frac{1}{2} \sigma^2 S^2 C_{ss} \right)$. We finally

obtain the Black-Scholes equation (Black, 637):

$$C_t + rSC_s + \frac{1}{2}\sigma^2S^2C_{ss} = rc$$

-
1. Ito's lemma for Brownian motion is $(x, t) = \left(f_t + a(x, t)f_x + \frac{1}{2}b^2(x, t)f_{xx} \right) dt + b(x, t)f_x dz$. In this equation $a(x, t)$ and $b(x, t)$ are deterministic function of x and t , and z represents a standard Brownian motion.

Risk exposure

The risk exposure of option is a little different from the risk exposure for futures. It is asymmetric. The largest loss an option buyer will suffer is the premium and the gain that the buyer will get is going to be offset by the premium that he paid before. While for an option seller, he will gain at most the premium and the premium will also offset the downside risk. Concerning about counterparty risk, after the option buyers finish paying all premiums, he fulfilled his entire obligation to the option sellers. In contrast, option sellers are required to put down margin according to their position marked to market to ensure they will carry out the contract if the option buyers choose to exercise their right.

Credit Default Swaps

In credit default swaps, there are two parties and a reference entity. One can be called the protection buyer who pays a fee to the protection seller so that when any credit event happens to the reference entity, the protection buyer will receive payments from the seller. If nothing happens by the end of the contract, then the protection seller will win the fee that protection buyer pays at the beginning and doesn't need to pay out anything. There are normally eight types of credit events, which include bankruptcy, credit event upon merger, cross acceleration, cross default, downgrade, failure to pay, repudiation or moratorium and restructuring. Here for the convenience of explanation, we will assume that the credit event happens to a company is bankruptcy, also called as defaults. In real world, Credit Default Swap is normally five years and the protection buys pays the fee separately rather than upfront. Each swap premium payment can be calculated by multiplying notional amount, swap rate and the percentage of days within one payment period over 360. The protection buyer is possibly the one who holds bonds of the reference entity and exposes to the default risk of the reference entity. There is another condition for the protection buyer to buy credit

default swaps, which is for speculative reasons and they think that the reference entity is very risky and going to default. For these buyers, they are like holding short position of the reference entity's bonds and the protection sellers are like buying reference entity's bonds. And credit default is like a tool that can create short position of bonds for individual, who is not very possible in real world without CDS and create a leveraged credit exposure for the protection seller since they are bearing similar risk as holding reference entities' bonds, while not paying the principal. When reference entity defaults, there are two kinds of settlement. One is cash settlement, which means the protection seller will pay the amount of money that is determined by the decline of the reference entity's bond price to compensate the loss for the protection buyer. The other method is physical settlement. The protection buyer will give the bad bonds of the reference entity to the seller and the seller is promised to pay the protection buyer the par value of the bonds. The Credit Default Swap is currently taking the largest part of the credit derivatives market.

Below is a timeline to demonstrate credit default swap:

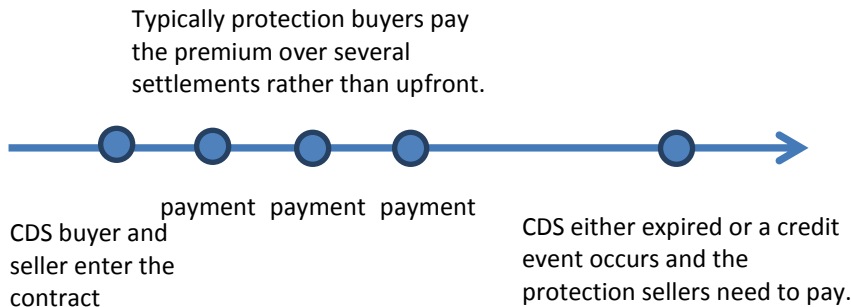


Figure 8: Timeline for CDSs

Except from the customized credit default swap arrangements between two counterparties, Dow Jones also manages Credit default swap index, which is essentially a standardized basket of credit risk of many corporations as reference entities. The biggest difference between a normal credit default swap and a credit default swap index is that the premium payments stop when a credit event happens to a normal credit default swap; while

for CDS index, since it has a basket of reference entities, when one of the corporation defaults, the index buyers need to continue paying premium, but just with less money because the notional amount decreases as a result of the corporation defaulting (Fabozzi, 48).

Pricing Model

Let's start exploring from a simple example. Suppose we have a CDS with swap rate of 300 basis points and face value of \$10 million. This means the protection buyer needs to make quarterly payments of $\$10\text{million} \times 0.03 \times 0.25 = \$75,000$. Then let's assume after 1 month, the reference entity suffers a credit event. We also know the recovery price as well, which is \$45 per \$100 of face value (recovery price can be interpreted as the remaining value of the reference entity after the credit event). After the credit event, the protection seller then needs to pay the protection buyer for the loss, which is $\$10\text{million} \times (100\% - 45\%) = \5.5 million , and the protection buyer needs to pay the 1-month accrued premium, which is $\$10\text{million} \times 0.03 \times \frac{1}{12} = \$18,750$.

Next we are going to explain how CDS mark-to-market value works. Let's consider a protection buyer purchases 5-year protection on a corporation with swap rate of 60 basis points and tries to value his position after one year. On the date after one year, the 4-year CDS is quoted with 170 basis points in the market. Then the

Mark-to-Market Value

= current market value of 4-year Protection - expected present value of 4-year premium leg at 60 basis points

= expected present value of 4-year premium leg at 170 basis points - expected present value of 4-year premium leg at 60 basis points.

$$=170bp \times Risky PV01 - 60bp \times Risky PV01$$

$$=110bp \times Risky PV01$$

The Risky PV01 (RPV01) is defined as the expected present value of 1bp paid on the premium leg until a credit event happens or the CDS expires. In order to calculate this RPV01, we will need a more complex model because we need to consider the possibility of a credit event happening over the CDS contract period which will essentially terminate the premium paying. Now for the protection buyer to realize this mark-to-market value gain, he can unwind it with the protection buyer for a cash rewind value, which will be equal to the mark-to-market position.

The most common approach to model the probability of credit events of the reference entities is the reduced-form approach. The probability of a credit event is modeled as Poisson counting process, which means the probability of a credit event happening within $[t, t + dt]$ conditional on the surviving to time t and is proportional to a function $\lambda(t)$, which is called as hazard rate.

$$\Pr[\tau < t + dt | \tau \geq t] = \lambda(t)dt$$

We can interpret this model as the reference entity defaulting in a time dt with probability $\lambda(t)dt$ or it surviving through the time dt with probability $1 - \lambda(t)dt$. We are also going to simply assume that the hazard rate process is deterministic, which also means it is independent of interest rates or recovery rates. Here is a picture that can clearly demonstrate this model.

Figure 4. The equivalent of a binomial tree in the modeling of default in which the tree terminates and makes a payment K at default

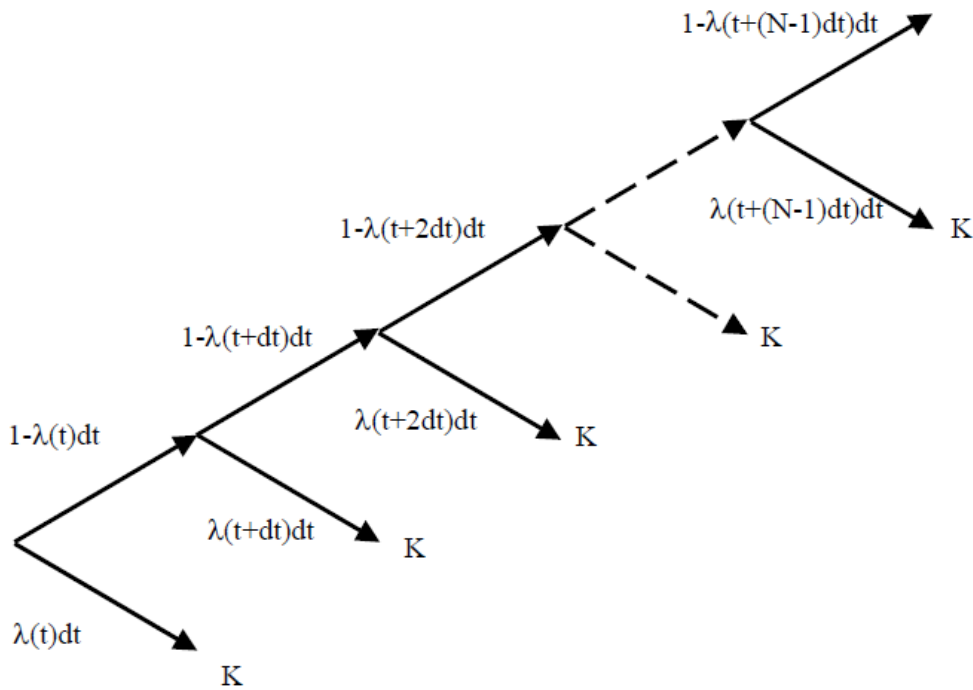


Figure 9: The equivalent of a binomial tree in the modeling of default.

According to this model, we can compute the continuous time survival probability to time T conditional on surviving to time t_v by having $dt \rightarrow 0$. And then

$$Q(t_v, T) = \exp\left(-\int_{t_v}^T \lambda(s) ds\right)$$

Next we are going to value the premium leg, which includes all the premiums made until the credit event happens. Let's assume there is in total N payments if the CDS makes to maturity and the swap rate is $S(t_0, t_N)$. We are also going to ignore premium accrued for now. Then the premium leg of existing contract is:

$$\text{Premium Leg PV}(t_v, t_N) = S(t_0, t_N) \sum_{n=1}^N \Delta(t_{n-1}, t_n, B) Z(t_v, t_n) Q(t_v, t_n)$$

$\Delta(t_{n-1}, t_n, B)$ is the day count fraction, $Z(t_v, t_n)$ is the Libor discount factor and $Q(t_v, t_n)$ is the arbitrage-free survival probability of the reference entity conditional on

surviving to t_V . Next let's consider the premium accrued, which we will need to consider the possibility of defaulting between the two payments. Then formula will look like:

$$\text{Premium Leg PV}(t_v, t_N) = S(t_0, t_n) \sum_{n=1}^N \int_{t_{n-1}}^{t_n} \Delta(t_{n-1}, s, B) Z(t_v, s) Q(t_v, s) \lambda(s) ds$$

The above formula can be approximated as the following equation by taking the average accrued premium as half of the full premium which is set to be paid at the end of the payments period.

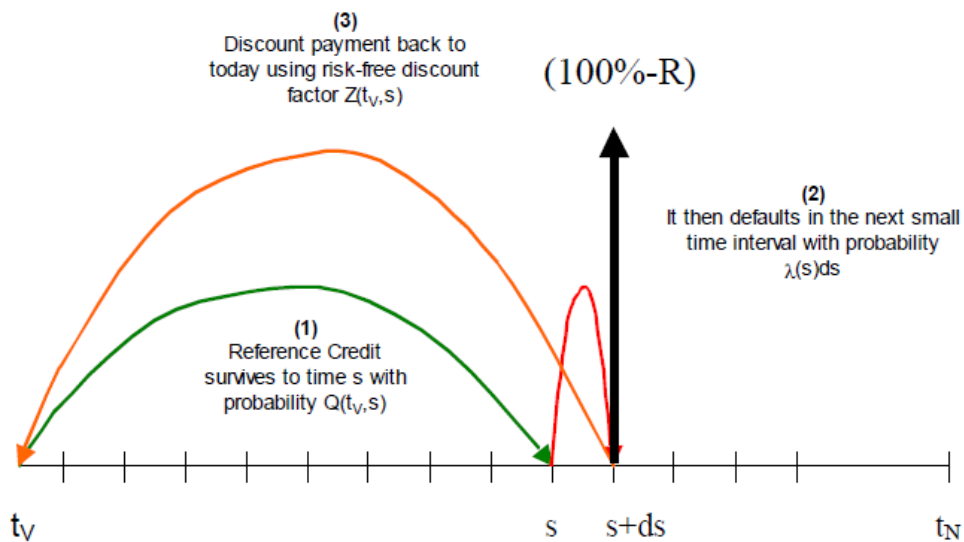
$$\begin{aligned} \text{Premium Leg PV}(t_v, t_N) &= S(t_0, t_n) * RPV01 \\ &= \frac{S(t_0, t_n)}{2} \sum_{n=1}^N \Delta(t_{n-1}, t_n, B) Z(t_v, t_n) (Q(t_v, t_{n-1}) - Q(t_v, t_n)) \end{aligned}$$

Thus we get equation for RPV01 for accrued premiums as

$$RPV01 = \frac{1}{2} \sum_{n=1}^N \Delta(t_{n-1}, t_n, B) Z(t_v, t_n) (Q(t_v, t_{n-1}) - Q(t_v, t_n))$$

Next we are going to value the protection leg, which is contingent payment of (100% - recovery rate) of the face value of the CDS depending on the credit event. The following timeline can clearly demonstrate the calculation logic of the protection leg.

Figure 6. Steps in the calculation of the expected present value of a recovery rate which is paid at the time of a credit event



The formula can be written as the following:

$$Protection\ Leg\ PV = (1 - R) \int_{t_V}^{t_N} Z(t_V, s) Q(t_V, s) \lambda(s) ds$$

Here R represents the expected recovery rate if a credit event happens. Z is the Libor discount factor and $Q(t_V, s)$ is the arbitrage-free survival probability of the reference entity living to time s .

With the present value of both protection leg and premium leg available, we can then equate them to get the breakeven swap rate, plug in the RPV01 value to get the mark-to-money value and find out the current market spread to maturity (O’Kane, 1).

Risk exposure

The protection buyer is facing the risk of not receiving the payments promised by the seller. Thus a common way to prevent such a risk is asking protection the seller to post collateral for its responsibility to pay the protection buyer whenever a credit event happens.

Cash Flow Collateralized Debt Obligations

A collateralized debt obligation (CDO) typically has a CDO portfolio manager. He is in charge of first raising money by issuing its own bonds and then invests the money he raises into either bonds, loans, or other assets. The assets the manager invests are the underlying assets of the portfolio. Then the payments on the asset portfolio can be used to repay the bonds that the CDO raises its money from. The manager will also collect fees for actively managing the portfolio. This is only the basic idea of CDO. The most important feature of CDO “credit tranching”, which means a CDO can issue different classes of bonds, for example, senior debt, mezzanine debt, subordinate debt and equity. Each class is exposed to a different level of risk. The most senior debt is the least risky debt because the more senior class will have higher priority to receive the asset repayments than the less senior class. Thus naturally, the less senior class demands higher interest rate because it is exposed to greater risk. CDO is invented in 1987 and then became a fast growing sector. There are two kinds of bankers who can take advantage of CDOs. First kind is called arbitrage CDO, which bankers can make money from the spread between the yield that CDO earns on the underlying asset and the payments that CDO pays out for CDO investors. The other kind is called balance sheet CDO, which can help a bank move its assets from the balance sheet to a CDO portfolio so that there will be less asset on the bank’s balance sheet.

CDO life-cycle

First, ramp – up phase is when the CDO manager raises money from issuing CDO bonds and then uses the money to set up the initial portfolio. There are regulations from CDO governing documents to constrain the portfolio’s average maturity date and other parameters. Second is revolving period when the CDO manager receives the payments from underlying assets and repays CDO bonds. When an underlying asset reaches its maturity date, the CDO

manager doesn't have to amortize some of the CDO bonds, and instead he can reinvest the money into other assets. Third phase is the amortization phase when the manager amortizes all the CDO's asset and finishes repaying all the bonds that the CDO issues.

Different types of CDOs

CDOs can be classified by the underlying assets. Below is a table for this kind of classification:

Underlying Portfolio	Different types of CDO
Bonds	Collateralized bond obligation (CBO)
Loans	Collateralized loan obligation (CLO)
Asset-backed Securities or Mortgage- backed securities	Structured finance CDO
Mix of bonds, loans, asset-backed Securities	Multisector CDO
Other CDOs	CDO squared
CDS	Synthetic CDO

Table 1: Classification of CDOs

CDO can also be classified by its repayment method. If the CDO repays its bonds with cash and a set interest rate, this is called cash flow CDO. If the CDO repays its bonds depending on the market value of the underlying assets, this is called market value CDO.

Synthetic CDOs

Synthetic CDO's underlying portfolio consists of credit default swaps. Synthetic CDO works slightly different from the normal CDOs. The CDO portfolio acts as a protection seller of a credit default swap. As we mentioned before, the CDO portfolio can then receive

payments from protection buyers. The CDO may also invest money on low risk securities and then the payments from the underlying CDS and interest from low risk securities can pay the bonds that the CDO issues. The reason that synthetic CDO invests money on low risk securities is because when a credit event happens in the underlying CDS, CDO manager can use the money in the low risk securities to pay the protection buyer according to the CDS contract. The effect of the credit event in synthetic CDOs is similar to the effect of when the asset defaults within regular CDOs because both events will potentially decrease the ability for the CDO to repay its bonds and decrease the CDO's rating.

In the synthetic CDO, there is an additional class called unfunded class. The investors who buy this class will act like a protection seller themselves. They don't need to pay anything upfront and they will receive payment from the CDO portfolio as a protection seller. However, when a credit event happens, the unfunded class investors will need to pay money to the protection buyer through the synthetic CDO (Fabozzi, 119).

The year 2008 marked down an unprecedented global financial crisis in human history. From mid-2007 to 2008, the world experienced a series of collapse of financial institutions, bailout of banks by governments, and downturns in stock market worldwide.

Three out of the five largest investment banks failed – Lehman Brothers filed for bankruptcy protection, Merrill Lynch was purchased by Bank of America, and Bear Stearns was absorbed by JP Morgan. In addition to this, Fannie Mae and Freddie Mac were completely taken over by the federal government. American International Group (AIG) survived only after receiving an \$85 billion capital injection from the government. Starting from Monday October 6th, the stock market declined for a week straight in which the Dow Jones Industrial plunged 1874 points.

Although the exact cause of this crisis was still under debate, this crisis was largely coined with three words: Housing Bubble, Securitization, and Subprime Mortgages. This section was devoted to explore the cause of the 2008 crisis around these three concepts, examine the major game players, and understand how they interacted with each other.

Overview

The crisis was largely the net product of the three interactive factors and it is hard to isolate any one of them to account for the crisis. The three factors affected and reinforced each other in an endless loop only to worsen the situation. Their interaction can be illustrated as follows.

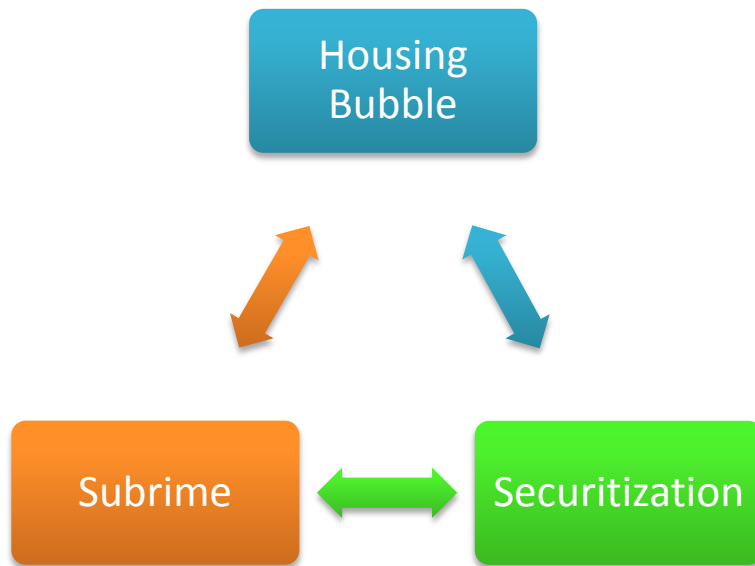


Figure 11: Three concepts in 2008 crisis

As the start, housing bubble planted the seed of evil. Initially fueled by a healthy increase in natural demand after World War II, rising house prices was soon artificially boosted by well-intended government policies. In order to increase homeownership, the fed government broke the law of natural demand and supply and created a favorable environment for home buyers fueled by “cheap money”. Urged by the government, mortgages lenders began to lower their standards to include more people with weaker credit. Subprime mortgages were created to cater to the growing appetite of the real estate industry. House prices continued to skyrocket. Soon after investment banks on Wall Street saw profits in subprime, they joined mortgages lenders. In order to be less exposed to the risk of subprime, banks securitized subprime mortgages and created complicated second level securities backed by these mortgages. They then shipped thousands of them to investors all over the world. With rapid growth in economy, developing countries were more than happy to purchase these financial products with their excess capital. Capital flew back into U.S. As securitization proved successful in diversifying and managing risk, more funds went to subprime loans. In turn, more subprime loans increased demand for housing, leading to even higher house prices.

The chain effect of housing bubble, subprime mortgages, and securitization process moved in a cycle just like electricity moved in a short circuit. They ran freely without disturbance from outside and overheated each other until the “wire” could no longer bear the heat. The crisis then broke out.

Key Concepts

Housing Bubble

One thing noticeable about the outbreak of the crisis was the coinciding collapse of U.S real estates in 2007. Before then, house prices had been skyrocketing at an unusual pace for a decade. Below are some quick facts.

- House prices rose steadily from the 1990s to 2006.
- The appreciation in house prices exceeded 10% per year from early 2004 to early 2006
- The home-ownership rate rose to a record level of 68.6% of households by 2007
- In Boston, the median home, which had sold for a reasonable 2.2 times median income in the mid-90s, rose to 4.6 times income in 2000s.

Notably, the last piece of information revealed a very interesting fact about the U.S house market – its growth went far ahead of the growth of household income. As Fannie Mae’s managers pointed out (Lowenstein), if we overlay a graph of house prices with a graph of incomes, the two lines tracked each other from 1976 to 1999. During that period, home prices grew in response to income growth. As a matter of fact, every blip in income growth was reflected in a corresponding change in home prices. As time moved into 20th century, however, the two lines deviated. Household income experienced a growing rate of only 2 percent while home prices rocketed alone. The growth of home prices by itself was very unnatural.

Generally, a mere rise in the price of an asset does not necessarily give rise to an inflation bubble (Lowenstein). As long as the increased price is aligned with changes in the assets' demand or supply, the rise in value is healthy to the economy. A bubble market, on the other hand, is one that lost its connection to the natural demand and supply. Since there was no sign of increased buying power in the 2000s – household income did not gain much, rapid appreciation of houses turned the housing market into a bubble market. The bubble market then fostered the rapid growth of subprime mortgages, with the latter transformed into securities which were traded all over the world and widely spread out the seed of the crisis.

If it was not the income to back the home price up, it must have been something else that could account for the expanding house market. As analysis later in this report showed, the U.S. government, along with Wall Street banks and investors overseas, was responsible for the start and growth of the housing bubble.

Subprime Mortgages

Subprime mortgages, to the opposite of prime mortgages, are a type of mortgages that are made to borrowers with shaky credit ratings. The word “Subprime” refers to the low credit rating of the borrower. Subprime mortgages borrowers usually have a credit rating below 600 on a scale of 300 to 900 while a typical consumer can have as high as 700.

Since the borrowers of subprime mortgages are riskier – the chance that they make late payments or even go bankruptcy is much higher, interest rates charged on subprime mortgages are usually higher than prime mortgages. Still, they can vary wildly based on factors such as the actual credit score of the borrower, size of down payments and number of late payments.

Subprime Mortgages played a very important role in the 2008 financial crisis. During

the pre-crisis period, especially into the 2000s, subprime mortgages issued to home buyers enjoyed a sharp rise in response to the increase in the U.S. house prices. The percentage of low-quality subprime mortgages rose from 8% to approximately 20% within 2004-2006 timeframe. Among them over 90% were adjustable-rate mortgages which are mortgages with an initial low interest rate that would grow much higher in a process called mortgage reset.

Subprime mortgages would normally have added much instability to the banking system and the U.S. economy. However, when house prices continued to rise, no problem surfaced in the early 2000s. Nobody questioned about subprime mortgages even if the borrower paid down little or even paid up. It was largely believed that, if a borrower was short of money paying his first mortgage, he could always take on a second one because higher value of his home gave him more collateral. After all, he could sell his house at a higher price to pay off the first mortgage.

As subprime mortgages defaulted moderately and subprime lending appeared profitable, Wall Street banks joined mortgage lenders in the game of subprime. Instead of being exposed directly to subprime mortgages, investments created securities backed by these mortgages through the process of securitization. They also introduced Credit Default Obligation (CDO) as the second layer of securitization. Through these products, subprime mortgages were packaged and shipped to different parts of the world and traded as completely different products to various investors.

Subprime mortgages began to fall back to their real value soon after house prices declined steeply after 2006. Refinancing became much more difficult to home buyers without the backup of real estate boom. They soon found themselves paying even higher interest than what they already could not afford. Securities backed by subprime mortgages slumped in their

value, deeply wounding investment banks. Global investors no longer purchased mortgage-related products as a result of loss of credit confidence. Subsequent to the deflation of housing bubble, subprime mortgages generated waves of default and failure throughout the global financial system.

Securitization

Unlike the previous two concepts which are widely accepted as ill-advised, securitization gained its bad reputation only recently, just because of the subprime mortgage crisis. Before then, it was thought as an innovative financial practice to manage risk.

Definition

Securitization is the process of pooling various types of assets and repackaging them into interest-bearing securities. The interest and principal payments from the assets (i.e. houses) are passed through from the originator (i.e. home mortgage issuer) to purchasers of the securities (i.e. investors).

Historical Review

Securitization was started in the 1970s. Back then, U.S. government-backed agencies were the only players in the business. Fannie Mae and Freddie Mac, two giants in the industry, securitized only prime mortgages. Starting from the 1980s, other income-producing assets began to be securitized. The market of securities backed by risky subprime mortgages also grew drastically. Entering into the 2000s, increasing numbers of financial institutions joined the game of securitization as they saw lucrative profits out of the business of subprime mortgages. Investment banks, insurance companies, pension funds, and hedge funds were eager to bite away their share. The scale and prevalence of the crisis explained how broadly and wildly securitization was practiced in the decade preceding the outbreak of the crisis.

Process Illustration

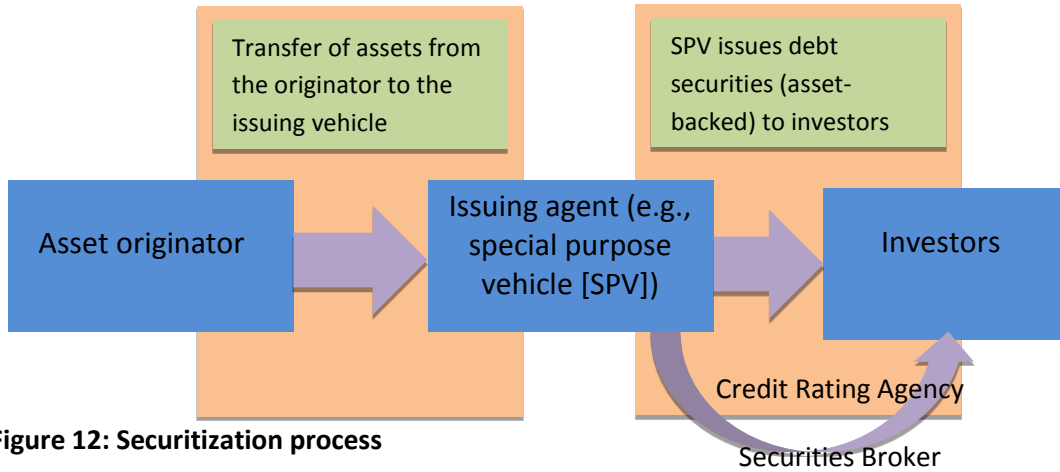


Figure 12: Securitization process

The process basically involves two steps. It starts with a company which wants to remove the loans or other assets they hold onto out of its balance sheet. The company, also called the originator in this process, pools these unwanted assets into so-called reference portfolio. It then sells the asset pool to an issuer (arranger), such as a special purpose vehicle (SPV). SPV, a bankruptcy-remote trust set up by a financial institution, has its specialty in acquiring these assets and writing them down the balance sheet for the originator. At the second step, the issuer finances the acquisition of the pooled assets by issuing tradable, interest-bearing securities that are sold to capital market investors (Jobst 2008). The investors receive fixed or variable rate payments from a trustee account funded by the cash flows generated by the reference portfolio (Jobst 2008). Thus, credit risk is transferred from issuers to investors.

In the real world, rating agencies and securities brokers also facilitate the process. They help to bridge the information asymmetry between SPV and investors. Rating agencies assign investment grade to the securities traded in the market. The grades provide guideline to investors. Securities brokers provide counseling to investors.

Tranches

Tranching is an innovative way issuers created to market the reference portfolio.

Instead of selling the entire portfolio as a big chunk, they slice it into tranches, each of which has a different level of risk and marketed separately. Investment return varies dramatically among different tranches. The safer the tranche is, the higher its priority to get its share of income generated by the underlying assets. On the other hand, the riskier the tranche is, the higher the rate of return.

Based on the seniority of the risk, tranches are classified as junior, mezzanine, and senior tranches. The most risky junior tranche, usually the smallest of the three, bears most of the credit exposure and receives the highest return. The senior tranche, to the opposite, are the least expected to default. However, the expectation is very sensitive to changes in the quality of the underlying asset. When the borrowers of subprime mortgages began to have problems making payments, junior tranches were first to be affected but loss of confidence among senior tranche holders fired panic among investors. That caused a fire sale of such securities.

Rationale for Securitization

Securitization represents a new way for companies to raise money, the way that actually increases capital availability. Suppose a leasing company wants to raise cash. Under traditional procedures, the company has to do so by issuing bonds or loans. Its ability to do so, poorly or well, and the cost incurred, is largely determined by the overall financial health of the company and its credit rating. If the company is not functioning so well recently or if it does not meet some standard criteria, it will not be able to issue quality bonds or loans that are attractive to investors at a low cost. If only it could sell some of its leases directly to potential buyers, that would have solved its problem of issuing bonds or loans. However, there is no secondary market where individual leases can be traded. But by securitization, the company can raise the cash it wants by pooling the leases and selling the package to an issuer, with the latter convert the pool into a tradable security.

Moreover, securitization lowers the cost of capital. Since the assets are now detached from the originator's balance sheet, the credit grade of the security issued is no longer tied to the overall credit rating of the originator. This means issuers can finance the pool of assets they purchase more cheaply than normal and in turn costing originator less to sell the pool. For example, by securitization, a company with an overall "B" rating with a triple-A rated asset pool is able to raise funds at the rate for triple-A instead of "B". Also unlike conventional debt, securitization does not inflate a company's liabilities because the assets are now off-balance-sheet.

Investors also benefit from securitization because it creates a broad array of attractive investment options. From the same asset pool, people who want steady and stable return with the least risk exposure can take away senior tranches while speculators can gamble with their risky junior tranches at extremely high rate of return. The flexibility of securitization transactions also help pension funds and other collective investment schemes which require a diverse range of highly rated long-term fixed-income investments beyond what the public debt issuance by governments can provide (Jobst 2008).

Securitization and 2008 Financial Crisis

Despite its great attributes in raising capital, lowering cost, and diversifying investment options, securitization played its negative to the extreme in the crisis.

The process of securitization encouraged predatory lending in the relationship between the mortgagor and originator and the relationship between the originator and the issuer.

Predatory lending describes fraudulent and deceptive practices by the loan lender that aim to mislead and take advantage of the borrower. In particular, subprime mortgagors could be very financially unsophisticated. They were either unaware of the variety of mortgage products available to them or unable to make a choice between the available options. That opened the

door for predatory lending. Especially when the originator saw the huge profits generated from subprime mortgage securitization, it had stronger tendency to trick the borrower into buying unsuitable mortgages. Predatory lending also occurred between the originator and the issuer. When the pool of mortgage loans was bought by the issuer, it had the responsibility to check the originator's credit status and the quality of the underlying mortgage pool. The detail of the deal between them was then finalized by the result of the examination. However, with the information asymmetry existing between the originator and the issuer, the originator had the tendency to misrepresent the quality of the mortgage borrower in order to write more mortgages off their balance sheet. This led to mortgage fraud. Predatory lending and mortgage fraud clearly knocked financial soundness off the chain of securitization and brought unpredictable instability to the whole process. Theoretically the issuer could put due diligence on the originator to help prevent the spread of mortgage fraud but in reality, especially in those days when securitization was fervently practiced by Wall Street investment banks, the close check was usually skipped.

Securitization also created moral hazards. A moral hazard refers to a situation where one party holds responsibility for another but has the incentive to put his or her own interests first (Dowd 2009). Most moral hazards involve excessive risk-taking: why not take the gambling if "heads I win, tails you lose" is the rule? After all, I do not have to assume any loss since you bear the risk for me. Moral hazards, if not controlled properly, often lead large-scale risk-overtaking just as what happened in the crisis.

Traditionally when a mortgage lender granted a loan to the borrower, he or she held onto it until its maturity. If the mortgage holder defaulted, the lender assumed all the loss. Therefore, it was natural for the lender to scrutinize the borrower before granting a mortgage. In this way, not many subprime borrowers would have been successful getting a mortgage.

However, under the new securitization process, the originator did not have the incentive to make serious check on the borrower because they did not expect to hold the mortgage for very long. They were only concerned about the payment it got for originating the mortgage. Now even the doziest mortgage broker could originate subprime mortgages for the least creditworthy borrowers (Dowd 2009). Unfortunately, this giant Ponzi scheme could only last as long as the housing bubble continued to inflate and new entrants continued to come into the market. Once interest rates started to rise and house prices began to fall, the supply of suckers inevitably dried up and the whole edifice began to fall in on itself (Dowd 2009). In the 2008 financial crisis, investment banks and investors all over the world helped to “relay” the crappy subprime mortgages in a string of securitization. When the head of the string burned soon after the collapse of housing market, the entire string got fire in a flash.

Furthermore, securitization introduced complex financial products which attracted misuse and abuse. Securitization was initially used to finance simple, self-liquidating assets such as mortgage. However, potentially all types of assets with a steady cash flow could be structured into a reference portfolio which after pooling could be converted to securities. In addition to mortgages, corporate and sovereign loans, consumer credit, project finance, lease/trade receivables, and individualized lending agreements could all be used to back up securities. The securities created this way generally are called asset-backed securities (ABS) though those securities backed by mortgage loans are called mortgage-backed securities (MBS) more precisely. Moreover, a variant called collateralized debt obligation (CDO) was created more recently to include an even more diverse range of assets.

Given the complexity and variety of instruments, it was very difficult for the investors to figure out what was the right choice for them. It was even difficult for rating agencies or securities brokers to understand the instruments they were grading or offering counseling on.

Commented by Allen Greenspan, former chairman of Federal Reserve, “I have not shallow background in mathematics and I have access to hundreds of top math PhDs. But even I could not completely understand those CDOs. Could anybody in the world possibly understand them? I doubt.” (House of Cards 2009)

Since nobody could have a thorough insight, buyers could largely be manipulated by sellers. That was exactly how Narvik, Norway, a town far above the Arctic Circle, was fooled to buying CDOs in 2006 (House of Cards 2009). The town of 17,000 people, suffering from a shrinking population and a growing budget deficit, needed money. So when sales people showed up, selling what they claimed “safe with high yield” products from Citigroup, the mayor and her counseling were overjoyed. From everything they examined they found no sign of problems except the content of the product remained mysterious. The product was rated triple-A – that was all they need to know to invest. Narvik thought their budget problem was over. However, when the subprime mortgage broke out, the products they held suddenly turned into nothing. Now the town is crashing and its residents have to pay the price of this poor investment for over a decade.

Securitization can be a valuable tool. It was the oversight in regulation and greedy in human nature that spoiled the good will of the tool. The lesson is: don’t just throw out the water at the baby; monitor the temperature and change the water accordingly. Water can sustain you as well as kill you!

Major Game Players

The U.S. Government – Over interfered the housing market

The initial rise of the housing bubble was thought to be a product of government

interference with the natural demand and supply.

During 1930's Great Depression, the United States had experienced the greatest mortgage crisis ever in its history. About half of mortgage was in default and the amount of mortgage lending had fallen by about 80 percent. In response to this, in 1938, Fannie Mae was created as a government agency to help with home mortgage lending market. In addition, Congress chartered Freddie Mac in 1970. These two companies functioned as private corporations but were sponsored by the government. Their intimate relationship with the government resulted in the companies' dominance in the industry on the one hand. On the other hand, however, it restrained the companies' freedom from political influence.

Initially Fannie and Freddie only held onto prime mortgages. They set strict industry standards that only those with strongest credit would be issued home mortgages. Starting from the 1990s, however, they were pushed by Congress to accept documented loans available to borrowers with spotty credit history. The goal claimed by the government, was to increase home ownership. By lowering standards, it was hoped that more people could afford loans issued by Fannie and Freddie.

Originally backed by the Democrats and the Clinton Administration, the policy was further pushed by the Bush Administration. As time moved on into year 2001, the overall mortgage environment became more and more favorable for home buyers.

The terrorist attack on September 11th 2001 also unintentionally helped with the on-going housing boom. As the result of the attack, the country was immersed in a mood of terror and depression. The economy halted as people were afraid to go out shopping. Allen Greenspan, the Chairman of Federal Reserve at that time, feared that a financial crisis of decades would come. The only way to prevent the crisis, as he believed, was to encourage

people spending. Starting from 2001, Greenspan made a series of cuts for short-term interest rates all the way to 2003 until the rates finally dropped to only 1 percent – history lowest of the generation. The inflation rate was at a time equal to or even greater than the interest rate. Banks were effectively borrowing money for nothing. People were spending far more than they could afford.

Investment Banks

Among all that experienced huge loss in the crisis, investment banks on Wall Street have won the least sympathy. Quite to the contrary, they were blamed intensely for their irresponsible conducts and insatiable greedy. The oversight in the risk they were taking in securitization and subprime mortgages not only cost themselves high price but also blew the storm of credit crisis over the globe.

Investment banks helped to supply capital from oversea investors to the U.S. housing market and sustained the housing bubble for quite a long time. Breaking the exclusive right of Fannie & Freddie to securitize, investment banks offered an alternative for mortgage lenders to write the loans off their balance sheet in the 2000s. They soon became a steady source for these lenders. By issuing securities converted from subprime mortgages and other kinds of other debt overseas and recycling capital back to the domestic housing market, investment banks expanded the appetite of the housing industry and subprime mortgage business dramatically.

To be less exposed to direct risk transfer from subprime market, investment banks sought ways to further enhance the practice of securitization. They soon invented CDOs – a vehicle to offload unwanted risk and make a fortune in the process. The table below lists the top CDO underwriters and how many deals they had from 2002 to 2007.

Underwriter	2002	2003	2004	2005	2006	2007	TOTAL
Merrill Lynch	0	3	20	22	33	18	107
Citigroup	3	7	13	14	27	14	80
Credit Suisse	10	7	8	9	14	6	64
Goldman Sachs	3	2	6	17	24	7	62
Bear Stearns	5	2	5	13	11	15	60
Wachovia	5	6	9	16	11	5	52
Deutsche Bank	6	3	7	10	16	5	50
UBS	5	2	5	10	16	6	35
Lehman Brothers	3	4	3	6	5	6	35
Bank of America	2	2	4	9	10	2	32`
TOTAL DEALS	47	44	101	153	217	135	697

Table 2: CDO underwriters

Sometimes it was hard to sell the mezzanine CDO tranches because unlike the most senior tranches, they usually did not get an investment grade. To solve this problem, investment banks repackaged them into new CDOs. In this way, the mezzanine CDO tranches were turned into part of new AAA bonds. This development was the notorious “CDO squared” or sometimes “CDO cubed”. On the one hand, it held investors more distant from

the underlying mortgages they were actually investing in. Investors were literally investing in something riskier than they thought. On the other hand, it deteriorated the CDO quality and made investors more exposed to the risk without them knowing it.

The banks went further and further down the road of securitization. They often conducted many iterations of securitization on their products. It was shown that Merrill Lynch created “CDO²” with as much as 15% of the assets from their prior CDO transactions. It also bought 59% of its CDO tranches that were resold into CDO². The table below summarizes the amount of repackaging done by the banks. In particular, Merrill Lynch topped with an average 4.79 iterations on their CDO assets. As the amount of repackaging increased, the complexity involved in these products multiplied. It became more and more difficult to perform analysis on their underlying collaterals.

Bank	Largest CDO Buyer of Bank’s CDOs	Largest CDO Supplier to Bank’s CDOs	LEVEL
BoA	Citigroup	Bank of America	3.00
Barclays	Merrill Lynch	Barclays Capital	2.79
Bear Stearns	Citigroup	Bear Stearns	3.94
Citigroup	Citigroup	Citigroup	4.17
Credit Suisse	Merrill Lynch	Credit Suisse	2.07
Deutsche Bank	Merrill Lynch	Deutsche Bank	1.62
Goldman Sachs	Goldman Sachs	Goldman Sachs	2.32
JP Morgan	Merrill Lynch	JP Morgan	2.79

Lehman	Merrill Lynch	Lehman Brothers	2.99
Merrill Lynch	Merrill Lynch	Merrill Lynch	4.79

Table 3: CDO repackaging level (Barnett-Hart 2009)

Investment banks soon felt tired of relying on mortgage lenders and other loan originators to provide them with the collateral required for CDOs. Instead, they wanted to be their own originator. They began to repackage their own collateral into CDOs. Bear Sterns underwrote CDOs with as much as 30% of the collateral issued by their in-house RMBS (*Residential Mortgage-Backed Securities*) business. Merrill Lynch bought 32% of all its in-house RMBS used in CDOs. Playing the role of both an originator and an issuer enabled the banks create and trade ABS or MBS more freely. Now that nobody could restrain them generating huge profit out of the business, nobody could also stop them over-taking risks.

While the Wall Street was wild packaging securities they created and enjoyed “riskless” profits from CDOs, they were winding up tremendous amount of risk due to so-called “super senior” tranches. Super senior tranches were created by chopping the uppermost tier of the AAA portion of a CDO. The tier held “super” low credit risk and all the lower tiers could be sold for higher yield than original. Many banks kept these super senior tranches to themselves because: 1. Very least capital was required to keep AAA securities. 2. It was difficult to sell these super senior tranches because of low yield. A JP Morgan report revealed that banks held around \$216 billion worth of super senior tranches of ABS CDOs in 2006 and 2007 (Barnett-Hart 2009).

The banks did not worry much about their increasing exposure to SS tranches because they assumed the risk of default was zero. In order for super senior tranches to default, the economy had to turn down completely from bottom to surface, which was very unlikely to

happen. Under such assumption, banks treated their SS CDOs as fully hedged even if they were only partially hedged – usually by credit-default swaps. However, this method of hedging of hedging left the banks with counter-party risk from other financial institutions (Barnett-Hart 2009). It turned out that later it was these positions that caused the majority of bank’s write-downs. As Merrill put it,

“The bottom line is that we got it wrong by being over-exposed to subprime. As the market for these (subprime) securities began to deteriorate, we began substantially reducing our warehouse risk by constructing CDOs and retaining the highest parts of the capital structure, which we expected then to be more resistant to market disruptions in terms of both liquidity and price...our hedging of the higher-rated tranches was not sufficiently aggressive nor was it fast enough.” (Barnett-Hart 2009)

In November 2008, Merrill Lynch, Citigroup, and Lehman Brothers, took write-downs of \$51.2, \$46.8, and \$15.3 billion, as the banks with the highest combined amounts of CDO and subprime assets. (Asset-Backed Alert, Nov. 18, 2008) The massive write-downs destroyed many of banks including Merrill and Lehman, pushing others to the brink of disaster. Furthermore, CDO losses have spread far beyond the investment banks on Wall Street, affecting very pool of investment money from pension funds to Norwegian villages. (House of Cards 2009) It remained still unsolved how much impact exactly investment banks had brought to the global economy.

Rating Agency

The violent crash of the asset-backed (especially the subprime mortgage backed) structured finance market was believed as one of the major catalysts for the 2008 financial crisis. Credit Rating Agencies have drawn much criticism for their role in fueling this

unsustainable and problematic market.

Rating Agencies play a crucial role in financial markets. They have the responsibility to assign an investment grade to various debt-related financial instruments, i.e. bonds. Investors rely on these ratings to make their investment decisions. Due to the lower transparency and higher complexity in the structured finance market, investors have an even heavier reliance on rating agencies than any other market. Inflation of credit ratings easily boosted the market to grow dramatically in a short period of time and subsequent downgrades in ratings accelerated the collapse of the market. As many highly rated securities defaulted in the crisis and rating agencies had no choice but to downgrade them, it was clear that these agencies did not correctly place their rating at first place.

There were many reasons why these weathercocks made such huge miscalculation this time. The data used to develop ratings was different from what was available before. Traditionally, when rating agencies rated corporate debt, they based their ratings on publicly available, audited financial statements. In contrast, structured debt ratings were based on nonpublic, nonstandard, unaudited information supplied by the originator or issuer (Katz 2009). There could be potentially a lot of misrepresentation in the information, especially when the entire industry went insane in subprime mortgage business. Moreover, since rating agencies had no obligation to perform due diligence to check the accuracy of the information and the new mortgage-backed securities and CDOs were so complex to examine, rating agencies tended to and sometimes had to rely on representation and warranties provided by the originator or issuer. This largely undermined rating agencies' independence.

Rating agencies also fell behind updating their rating method to better fit the structured financial products. More than often, they lacked extensive historical data to make

the correct distribution assumption of the innovative products. In order to rate anyways, they used older models that were created for traditional products. The models turned out to be inadequate and inappropriate. For example, they failed to account for default correlation rise in the pool of assets in response to declines in housing prices. Rating agencies were also reluctant to invest in newer databases and rating models because they were costly and hurt profits.

At the root, there is always this conflicting interest in the nature of credit rating. Rating agencies run their business by charging securities issuers. The more securities they can get to rate, the more they get paid. If the issuers are consistently unsatisfied with the grade they get for their instruments, they will go to other agencies to get a better chance. This “issuer pays” business model encourages rating agencies to relax their own criteria in order to maintain or attract more market share. Moreover, a few large investment banks which controlled much of the deal flow made it even worse. They often “shopped around” for the highest ratings on their lucrative issuance deals by playing one rating agency against. Continuing pressing rating agencies, these banks often landed the privilege to consult agencies informally on structures they could create to achieve high ratings. The practice inevitably caused an inflation of credit rating and adversely hurt the industry standard.

Misrepresented information provided by issuers, complex structured finance products to rate, outdated rating models in use, and pressures to lower standard from peers and clients all led credit rating agencies to creating a rating bubble in the period of pre- crisis. According to a report issued by the BIS and Basel Committee’s joint forum (Report on securitization incentives), between 1990 and 2006, assets with the highest credit ratings rose from a little over 20 per cent of total rated fixed-income issues to almost 55 percent. It means more than half of the world’s debt securities were considered risk-free. The report also says during the

same period, Asset-Backed Securities (ABS) accounted for 64 percent of the total growth in the amount of AAA-rated fixed income while public debt, corporate debt, and other debt contributed only 27, 2, and 8 percent to the total, respectively.

The bubble did not sustain for very long. It soon crashed as house prices went down and mortgages defaulted. As of 2007, triple-A rated CDOs were downgraded an average of 16 rating points (1 means AAA, 22 means D) (Barnett-Hart 2009). It was shocking that as many as 70% of CDOs defaulted among all rated by some agencies. Numerous investors bore loss. Structured finance market became as volatile as ever. With its careless use of power to blow a credit rating bubble up and smash it in a flash, credit rating agencies helped to spill out the evil.

Global Investors

It is never the heat itself that causes a fire. It is also the fuel. Global investors played the role of fuel in the crisis. Without their help, the crisis would never become so widely spread out and so profoundly influential. It might not have even happened.

Low interest rates not only encouraged spending instead of saving among consumers. It also turned investors to higher-yielding securities. A variety of investors ranging from professional investors and corporate CEOs to hospital funds and state pension funds began to look at loans to private equity deals. They were assured that these loans were safe as well as the mortgage pools. After all, when interest rates were low, the only way to earn more was to assume more risk.

Low interest was not limited to the United States. It was a worldwide phenomenon at the time. China, Japan, Germany, and oil-exporting nations were all experiencing the same issue. Unlike the U.S., they were spending less than their income and thus had extra money to

lend. Thus they became the investors who purchased higher-yielding securities backed by private equity loans from the United States.

It was these investors' dollars that fueled the credit binge. In 2000s, when borrowing reached its peak, the United States alone sopped up 70 percent of the surplus capital flowing from the developing countries. It was largely argued that America was borrowing to spend only because other countries were lending. Although it sounded like America was blaming others for their own mistake, it was undeniable that global investors played their role in the development of the crisis.

