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

1/8/2012

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.

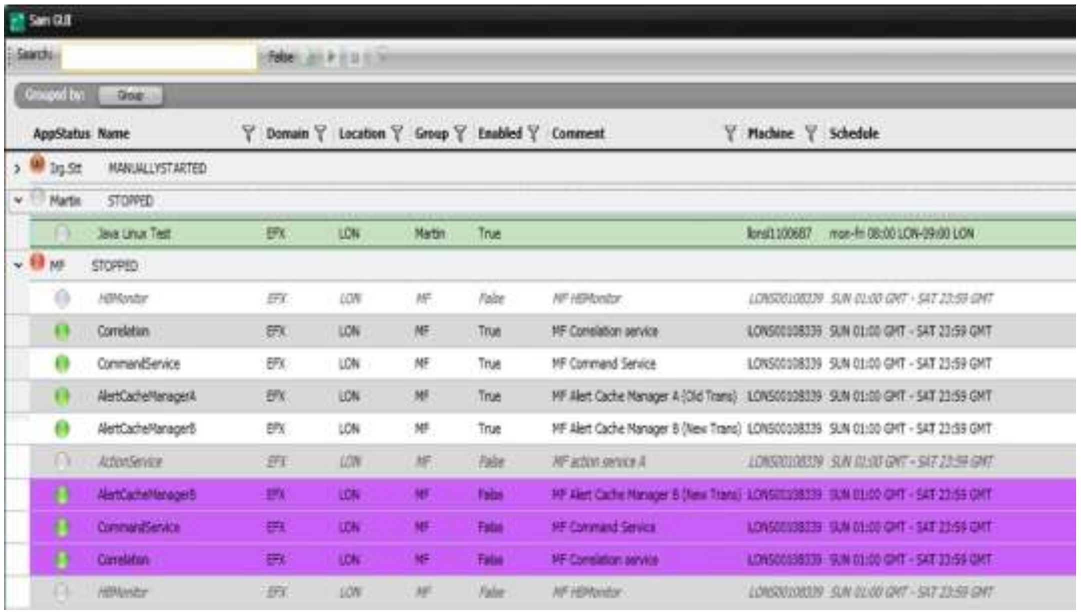
Table of Contents

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.

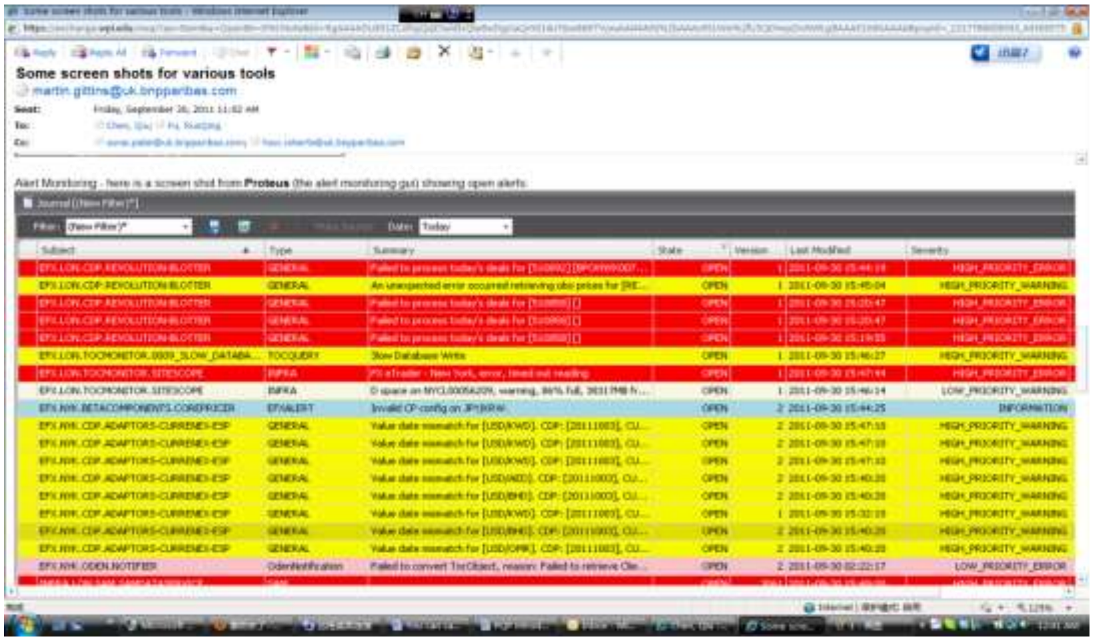


Search: False

Grouped by: Group

AppStatus	Name	Domain	Location	Group	Enabled	Comment	Machine	Schedule
Irg.Stt	MANUALLYSTARTED							
Martin	STOPPED							
	Java Linux Test	EFX	LON	Martin	True		lonsl1100687	mon-fri 08:00 LON-09:00 LON
MF	STOPPED							
	HBMonitor	*EFX*	*LON*	*MF*	*False*	*MF HBMonitor*	*LONS00108339*	*SUN 01:00 GMT - SAT 23:59 GMT*
	Correlation	EFX	LON	MF	True	MF Correlation service	LONS00108339	SUN 01:00 GMT - SAT 23:59 GMT
	CommandService	EFX	LON	MF	True	MF Command Service	LONS00108339	SUN 01:00 GMT - SAT 23:59 GMT
	AlertCacheManagerA	EFX	LON	MF	True	MF Alert Cache Manager A (Old Trans)	LONS00108339	SUN 01:00 GMT - SAT 23:59 GMT
	AlertCacheManagerB	EFX	LON	MF	True	MF Alert Cache Manager B (New Trans)	LONS00108339	SUN 01:00 GMT - SAT 23:59 GMT
	ActionService	*EFX*	*LON*	*MF*	*False*	*MF action service A*	*LONS00108339*	*SUN 01:00 GMT - SAT 23:59 GMT*
	AlertCacheManagerB	EFX	LON	MF	False	MF Alert Cache Manager B (New Trans)	LONS00108339	SUN 01:00 GMT - SAT 23:59 GMT
	CommandService	EFX	LON	MF	False	MF Command Service	LONS00108339	SUN 01:00 GMT - SAT 23:59 GMT
	Correlation	EFX	LON	MF	False	MF Correlation service	LONS00108339	SUN 01:00 GMT - SAT 23:59 GMT
	HBMonitor	*EFX*	*LON*	*MF*	*False*	*MF HBMonitor*	*LONS00108339*	*SUN 01:00 GMT - SAT 23:59 GMT*

Figure 1: Sam



Some screen shots for various tools

martin.gittins@uk.bnpparibas.com

Sent: Friday, September 30, 2011 11:02 AM

To: [illegible]

Cc: [illegible]

Alert Monitoring - here is a screen shot from **Proteus** (the alert monitoring gui) showing open alerts.

Journal [(New Filter)*]

Filter: (New Filter)* Date: Today

Subject	Type	Summary	State	Version	Last Modified	Severity
EFX.LON.CDP.REVOLUTION.BLOTTER	GENERAL	Failed to process today's deals for [[illegible]] [[illegible]...	OPEN	1	2011-09-30 15:44:19	HIGH_PRIORITY_ERROR
EFX.LON.CDP.REVOLUTION.BLOTTER	GENERAL	An unexpected error occurred retrieving obo prices for [RE...	OPEN	1	2011-09-30 15:45:04	HIGH_PRIORITY_WARNING
EFX.LON.CDP.REVOLUTION.BLOTTER	GENERAL	Failed to process today's deals for [[illegible]] []	OPEN	1	2011-09-30 15:20:47	HIGH_PRIORITY_ERROR
EFX.LON.CDP.REVOLUTION.BLOTTER	GENERAL	Failed to process today's deals for [[illegible]] []	OPEN	1	2011-09-30 15:20:47	HIGH_PRIORITY_ERROR
EFX.LON.CDP.REVOLUTION.BLOTTER	GENERAL	Failed to process today's deals for [[illegible]] []	OPEN	1	2011-09-30 15:19:55	HIGH_PRIORITY_ERROR
EFX.LON.TOCMONITOR.0009_SLOW_DATABA...	TOCQUERY	Slow Database Write	OPEN	1	2011-09-30 15:46:27	HIGH_PRIORITY_WARNING
EFX.LON.TOCMONITOR.SITESCOPE	INFRA	FX eTrader - New York, error, timed out reading	OPEN	1	2011-09-30 15:47:44	HIGH_PRIORITY_ERROR
EFX.LON.TOCMONITOR.SITESCOPE	INFRA	D space on NYCL0005A209, warning, 86% full, 3831 MB fr...	OPEN	1	2011-09-30 15:46:14	LOW_PRIORITY_WARNING
EFX.NYK.BETACOMPONENTS.COREPRICER	EFXALERT	Invalid CP config on JPY/KRW	OPEN	2	2011-09-30 15:44:25	INFORMATION
EFX.NYK.CDP.ADAPTORS-CURRENEX-ESP	GENERAL	Value date mismatch for [USD/KWD]. CDP: [20111003], CU...	OPEN	2	2011-09-30 15:47:15	HIGH_PRIORITY_WARNING
EFX.NYK.CDP.ADAPTORS-CURRENEX-ESP	GENERAL	Value date mismatch for [USD/KWD]. CDP: [20111003], CU...	OPEN	2	2011-09-30 15:47:10	HIGH_PRIORITY_WARNING
EFX.NYK.CDP.ADAPTORS-CURRENEX-ESP	GENERAL	Value date mismatch for [USD/KWD]. CDP: [20111003], CU...	OPEN	2	2011-09-30 15:47:10	HIGH_PRIORITY_WARNING
EFX.NYK.CDP.ADAPTORS-CURRENEX-ESP	GENERAL	Value date mismatch for [USD/AED]. CDP: [20111003], CU...	OPEN	2	2011-09-30 15:40:20	HIGH_PRIORITY_WARNING
EFX.NYK.CDP.ADAPTORS-CURRENEX-ESP	GENERAL	Value date mismatch for [USD/BHD]. CDP: [20111003], CU...	OPEN	2	2011-09-30 15:40:20	HIGH_PRIORITY_WARNING
EFX.NYK.CDP.ADAPTORS-CURRENEX-ESP	GENERAL	Value date mismatch for [USD/KWD]. CDP: [20111003], CU...	OPEN	1	2011-09-30 15:32:19	HIGH_PRIORITY_WARNING
EFX.NYK.CDP.ADAPTORS-CURRENEX-ESP	GENERAL	Value date mismatch for [USD/BHD]. CDP: [20111003], CU...	OPEN	2	2011-09-30 15:40:20	HIGH_PRIORITY_WARNING
EFX.NYK.CDP.ADAPTORS-CURRENEX-ESP	GENERAL	Value date mismatch for [USD/OMR]. CDP: [20111003], CU...	OPEN	2	2011-09-30 15:40:20	HIGH_PRIORITY_WARNING
EFX.NYK.ODEN.NOTIFIER	OdenNotification	Failed to convert TocObject, reason: Failed to retrieve Clie...	OPEN	2	2011-09-30 02:22:17	LOW_PRIORITY_ERROR

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

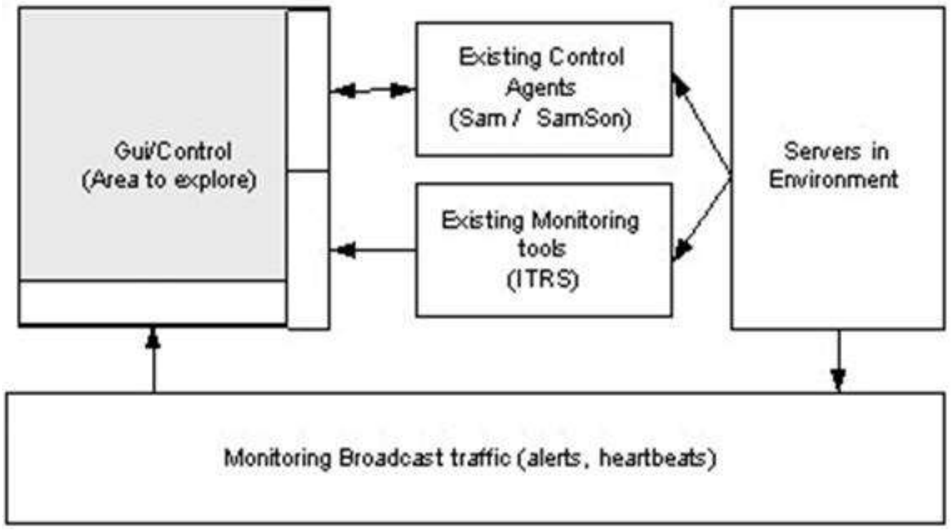


Figure 3 Block diagram of the Environment Cockpit (Sunai Patel, Project Description)

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 more” (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.

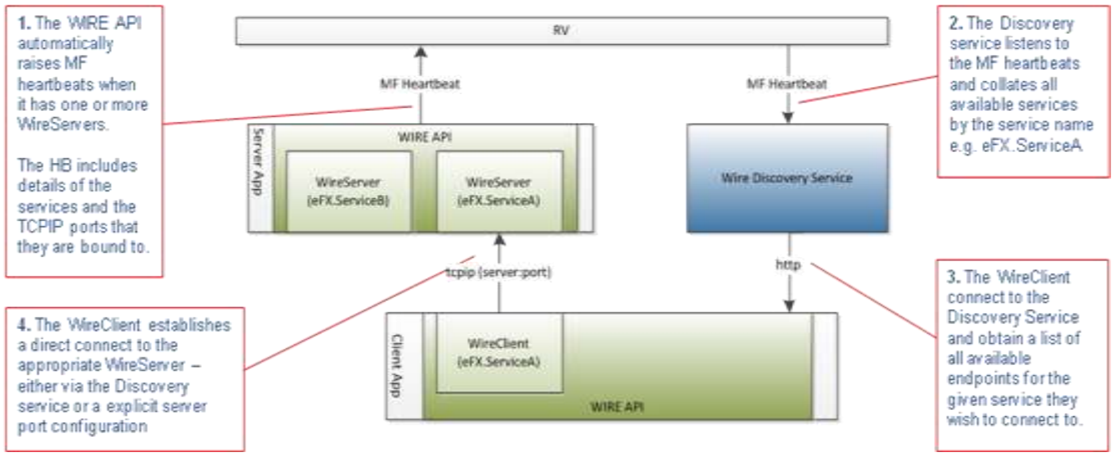

1. The WIRE API automatically raises MF heartbeats when it has one or more WireServers.
The HB includes details of the services and the TCPIP ports that they are bound to.
RV
MF Heartbeat
MF Heartbeat
2. The Discovery service listens to the MF heartbeats and collates all available services by the service name e.g. eFX.ServiceA
Server App
WIRE API
WireServer (eFX.ServiceB)
WireServer (eFX.ServiceA)
Wire Discovery Service
tcpip (server:port)
http
3. The WireClient connect to the Discovery Service and obtain a list of all available endpoints for the given service they wish to connect to.
4. The WireClient establishes a direct connect to the appropriate WireServer – either via the Discovery service or a explicit server port configuration
Client App
WireClient (eFX.ServiceA)
WIRE API

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.).

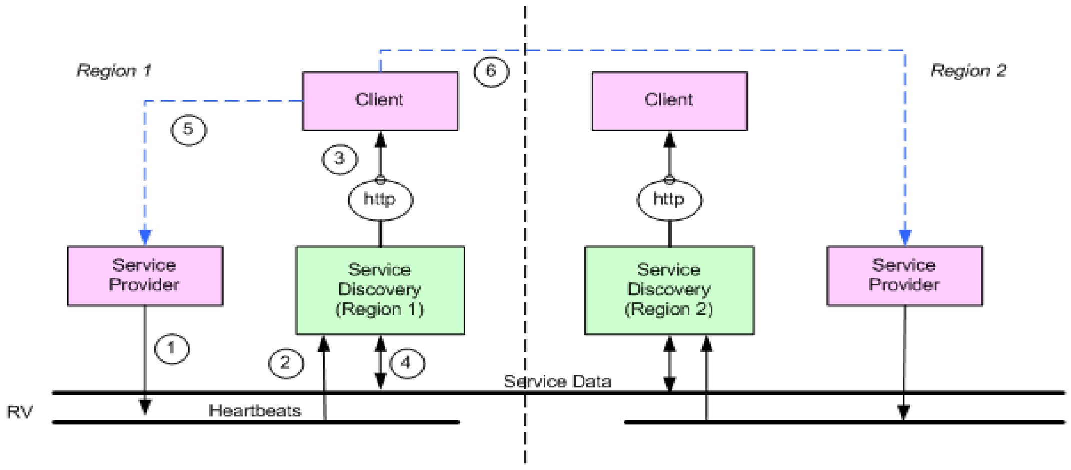


Figure 5: Discovery Illustration flow chart (The Wire.)

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)

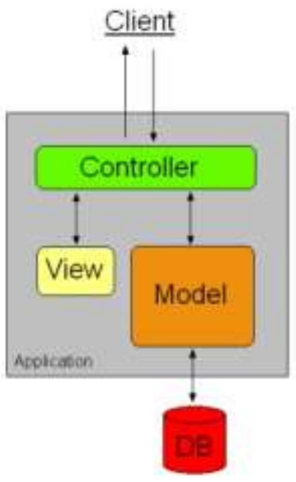


Figure 6: MVC diagram

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, TOCAdmin, 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 Queuepasa 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).

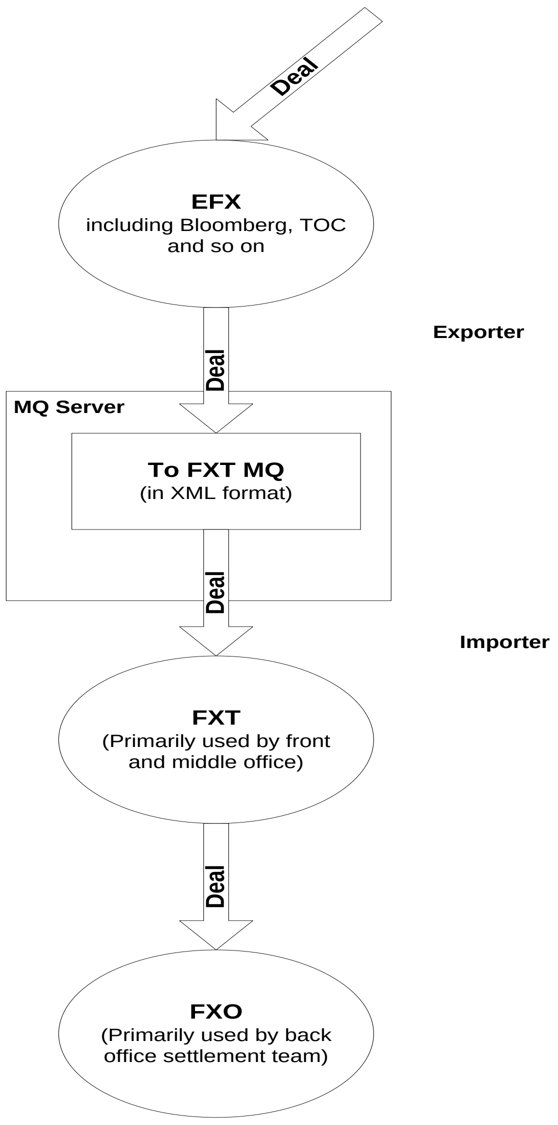

Deal
EFX
including Bloomberg, TOC and so on
Deal
Exporter
MQ Server
To FXT MQ
(in XML format)
Deal
Importer
FXT
(Primarily used by front and middle office)
Deal
FXO
(Primarily used by back office settlement team)

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).

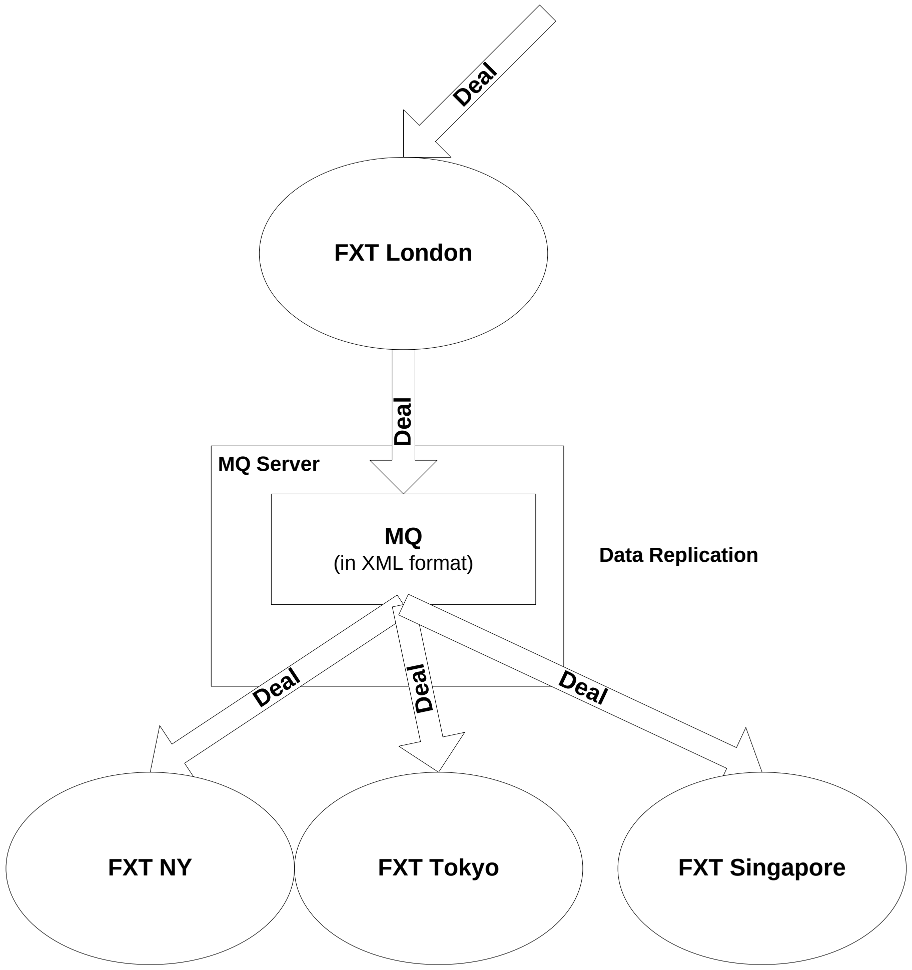


Figure 8: Deal information data flow (Storey)

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).

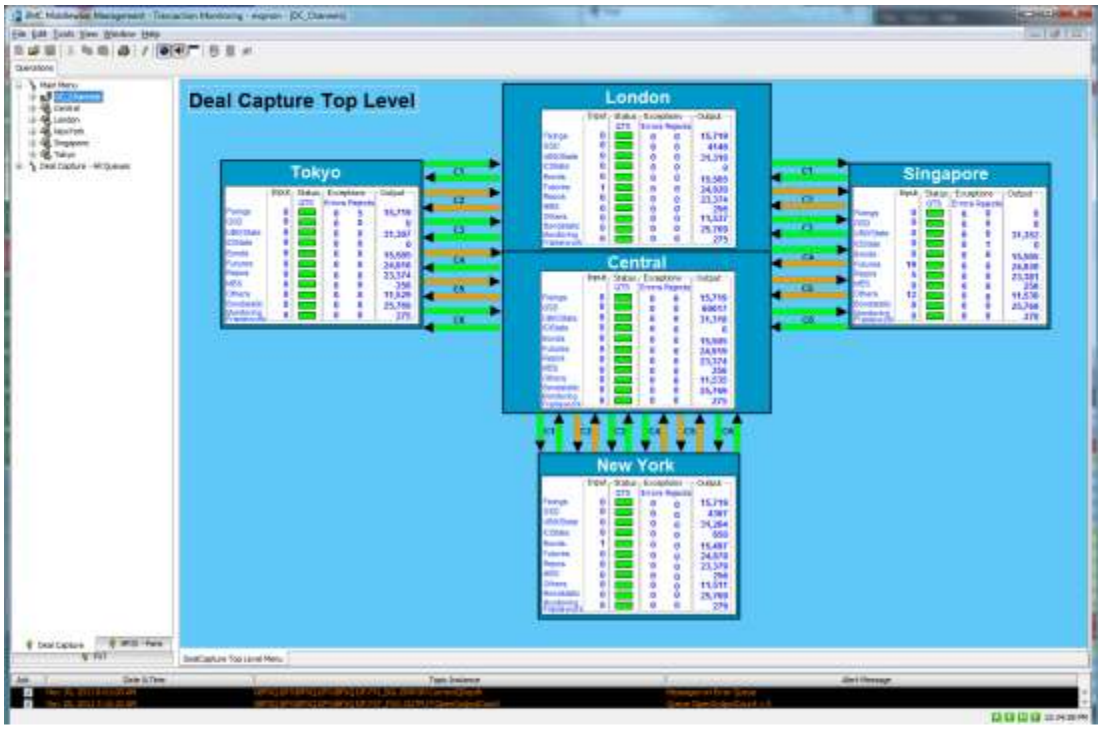


Figure 9: Snapshot of deal data transmitting in BMC I (Storey)

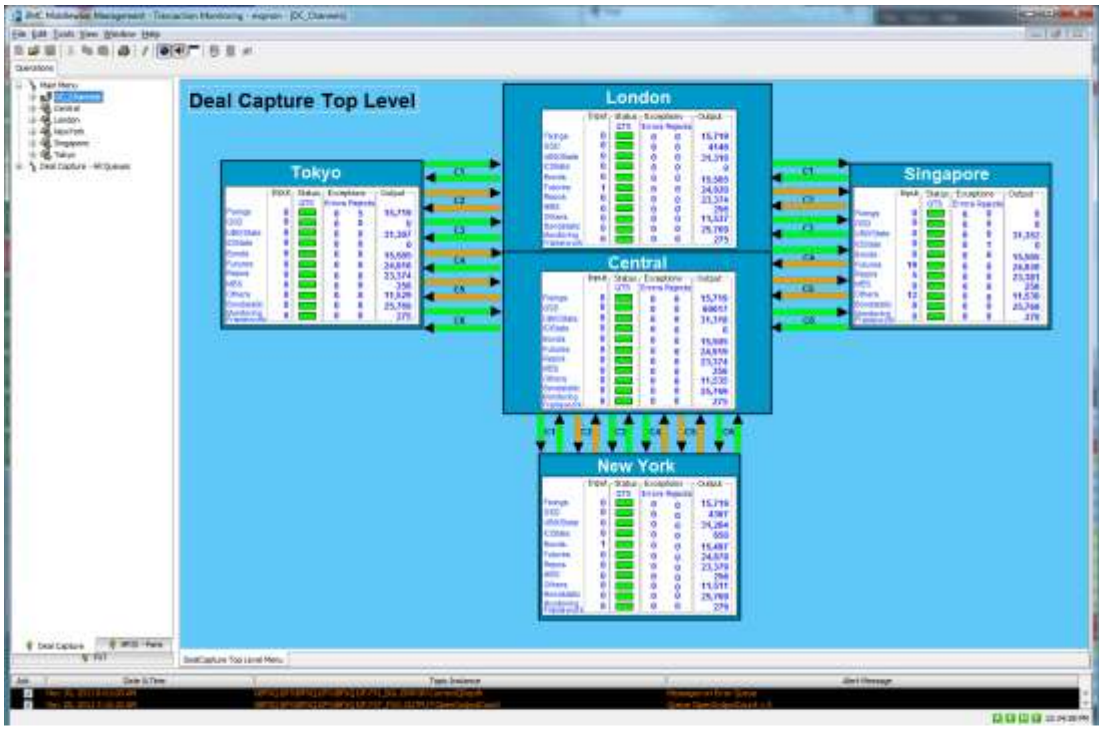


Figure 10: Snapshot of deal data transmitting in BMC II (Storey)

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.).

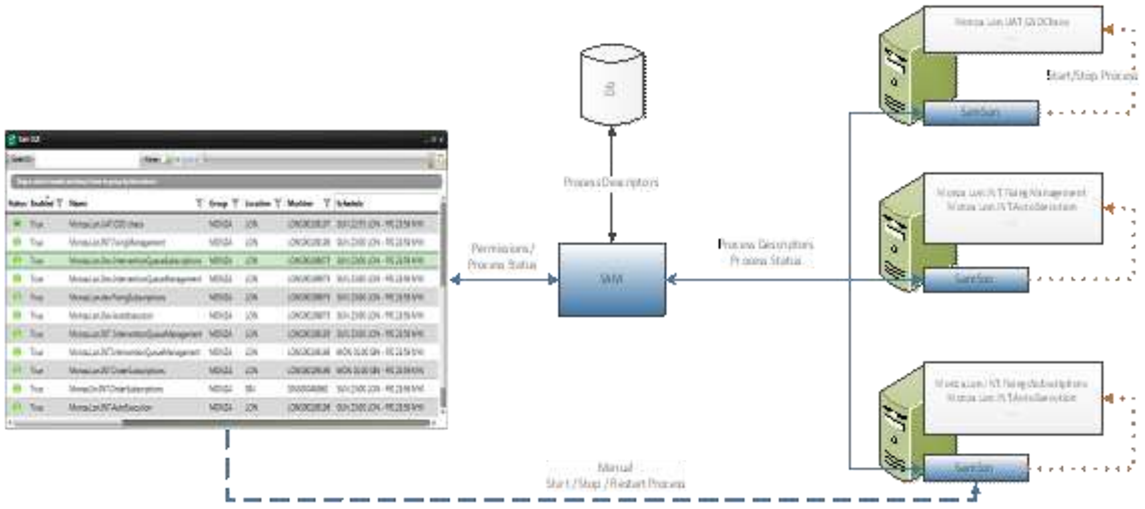

Db
Process Descriptors
Permissions / Process Status
SAM
Process Descriptors / Process Status
SamSon
Start/Stop Process
SamSon
SamSon
Manual Start / Stop / Restart Process

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, PoxEdge and PocGraph. In the PocVertex class, each item is assigned with an unqiue 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.

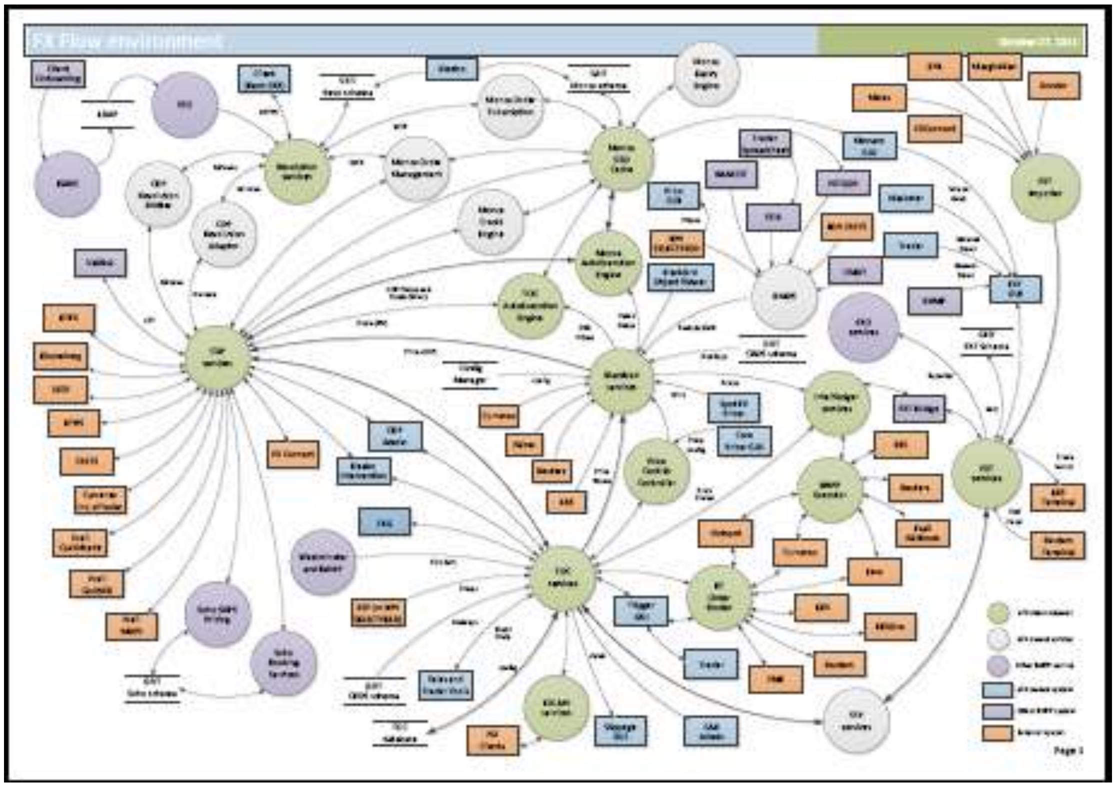


Figure 12: EFX flow environment display

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 diffult feat to display all this information clearly.

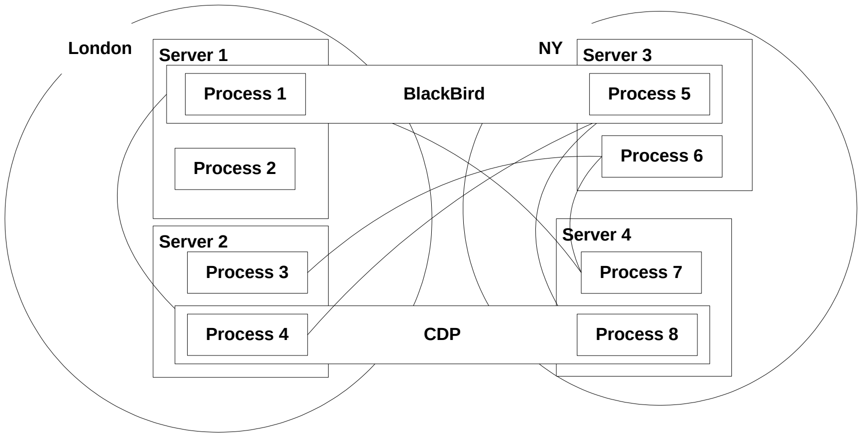


Figure 13 Logical view and physical view combined

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

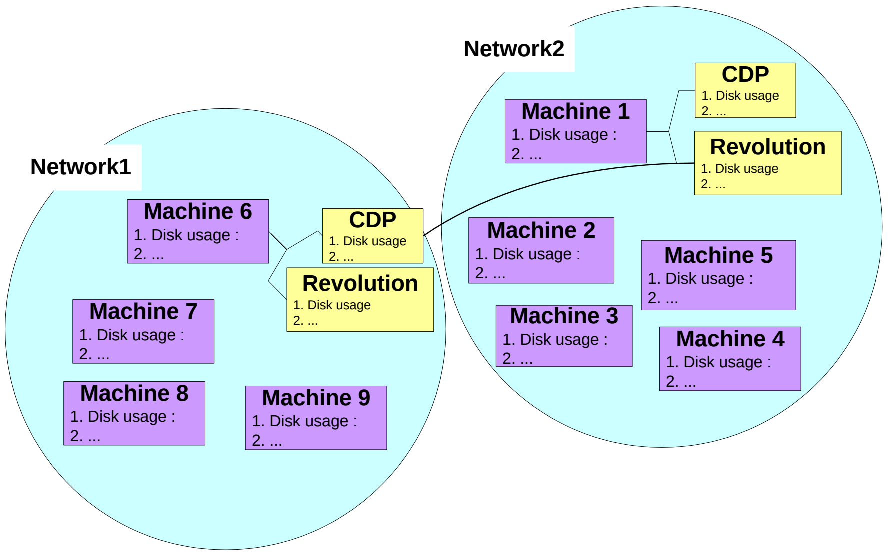


Figure 14 the Bubble View

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.

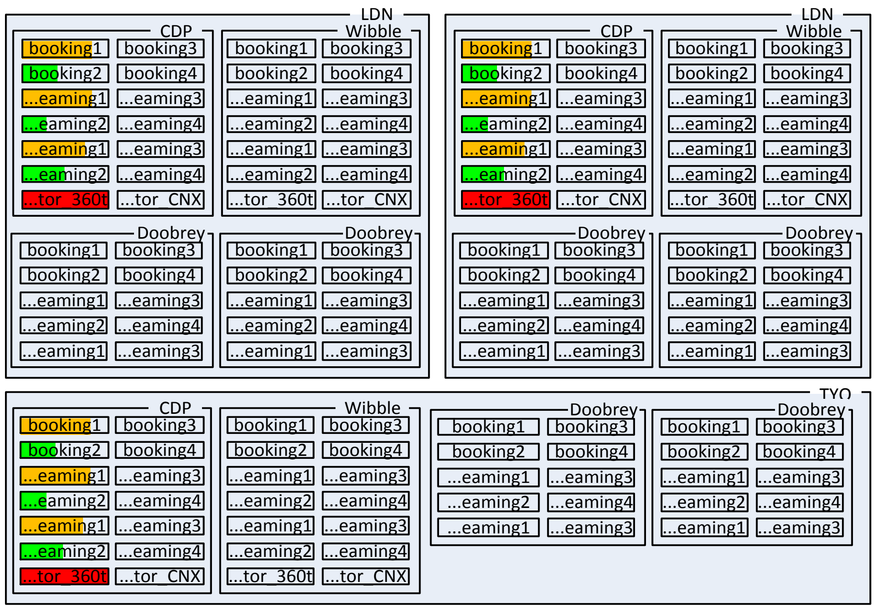


Figure 15 Grid View

Graphing tool comparison

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

Graphing Tools	Advantages	Disadvantages
Nshape	• 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	• only 10 basic shapes • very complicated to create shapes
Visio	• able to select templates • look very professional • easy to add graph	• not possible to develop a WPF or.Net based application
Ywork	• plenty of documentation and tutorial • super nice automatic layout • able to present overview and detailed view at the same time	• too expensive to buy the license

	• Convenient users control such as expanding or collapsing nodes and hovering over effects	
QuickGraph	• able to add shapes dynamically •simple to code with	• not enough documentation or tutorial
Graphviz	• open source • searching algorithms • using Dot text language to define a graph • flexible and fancy views • Reliable	• impossible to be directly used without a viewer
Graph#	• WPF based • relatively more documentation • completely free • able to customize features	• 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.

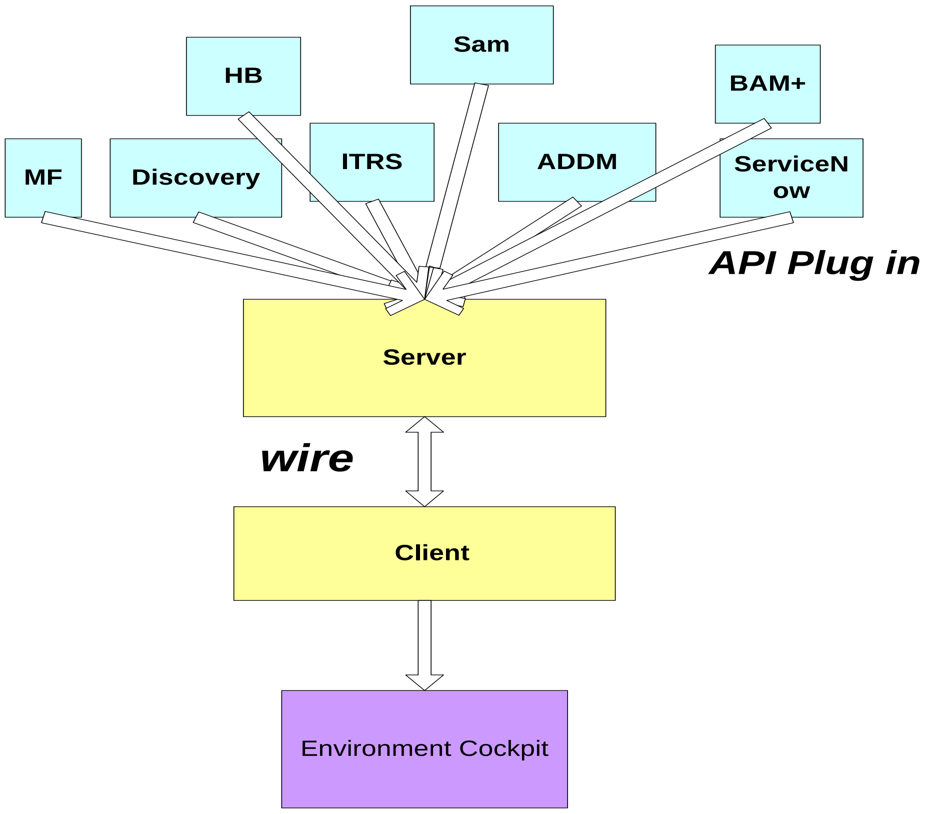
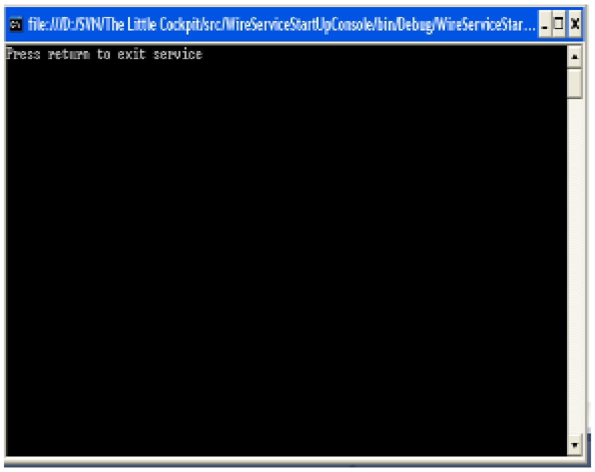


Figure 17 The Environment Cockpit architecture diagram

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 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.


file:///D:/SVN/The Little Cockpit/src/WireServiceStartUpConsole/bin/Debug/WireServiceStar...

```
Press return to exit service
```

Figure 18 Wire service Window

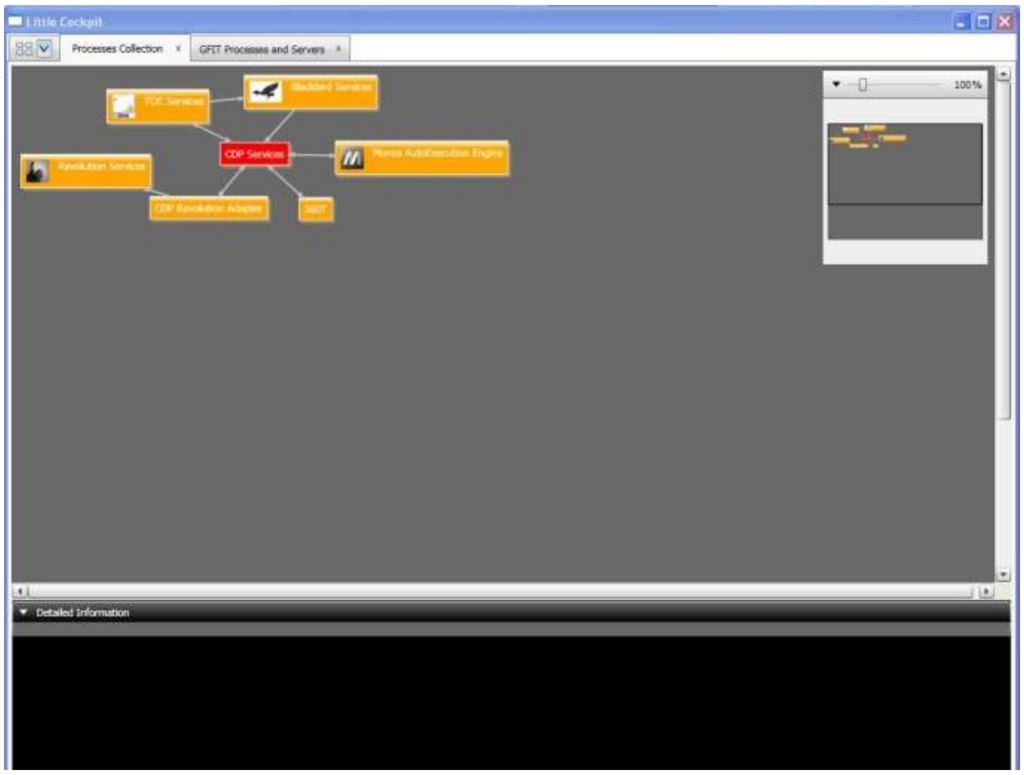


Figure 19 Cockpit Prototype snapshot I

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

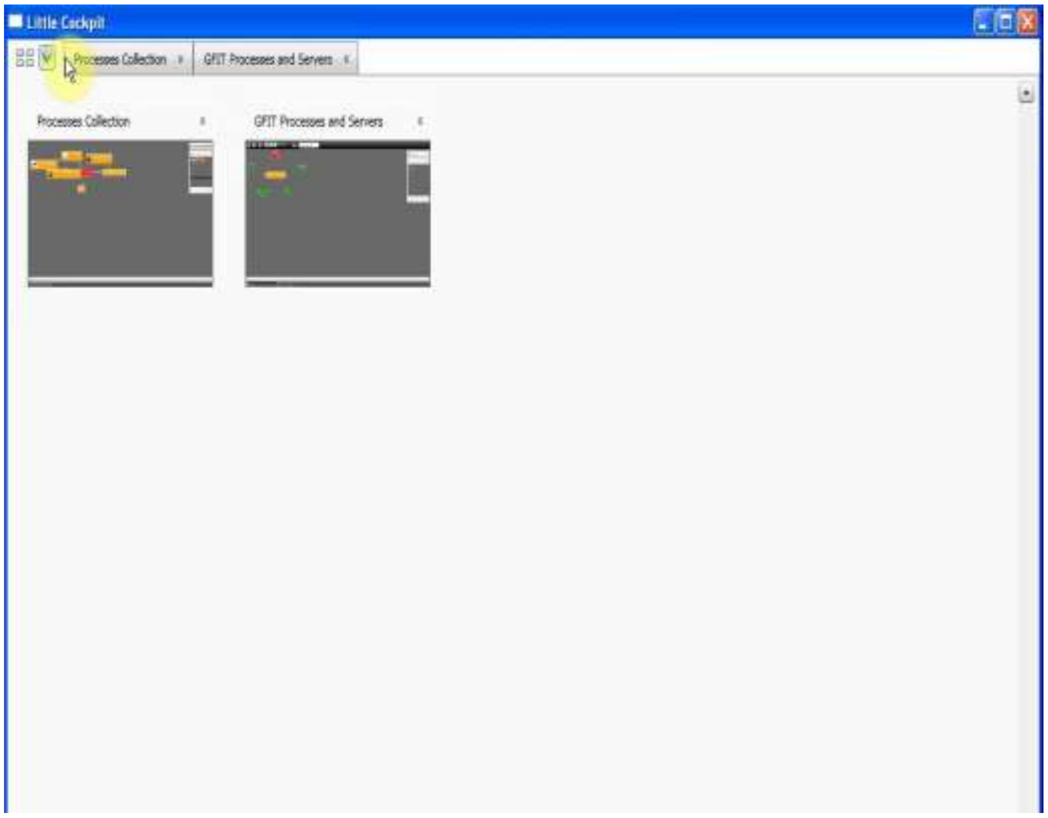


Figure 20 Tab view demonstration

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

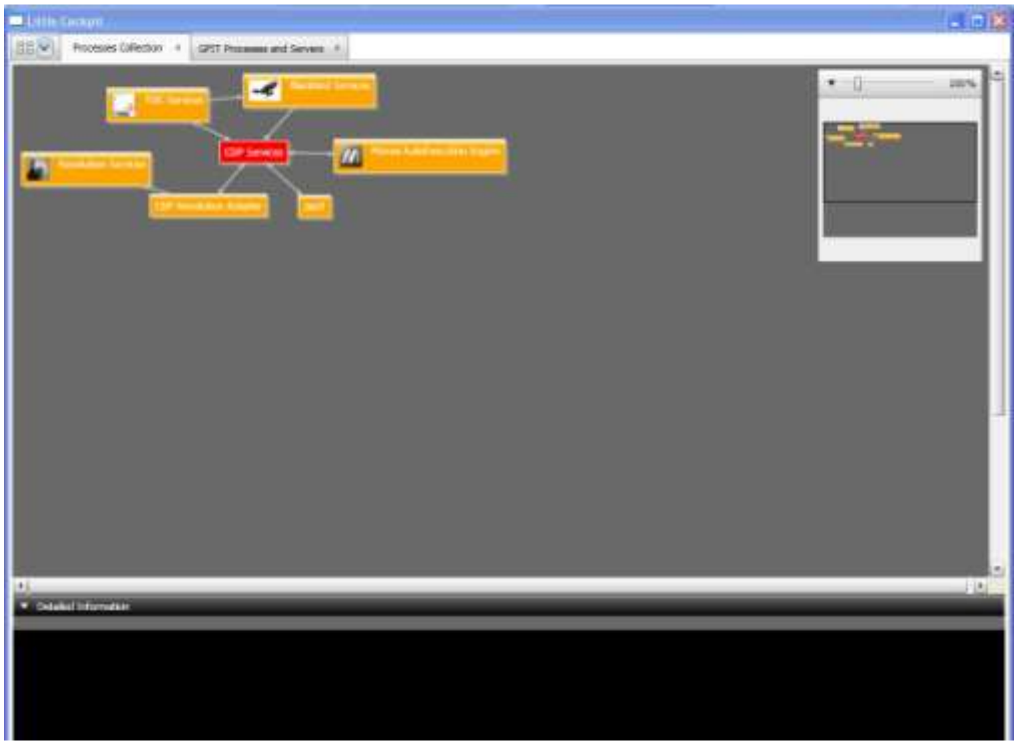


Figure 21 The cockpit prototype snapshot II

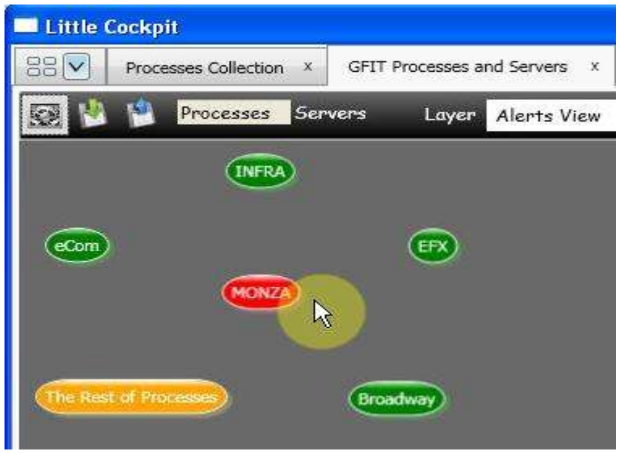


Figure 22The cockpit prototype snapshot III

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

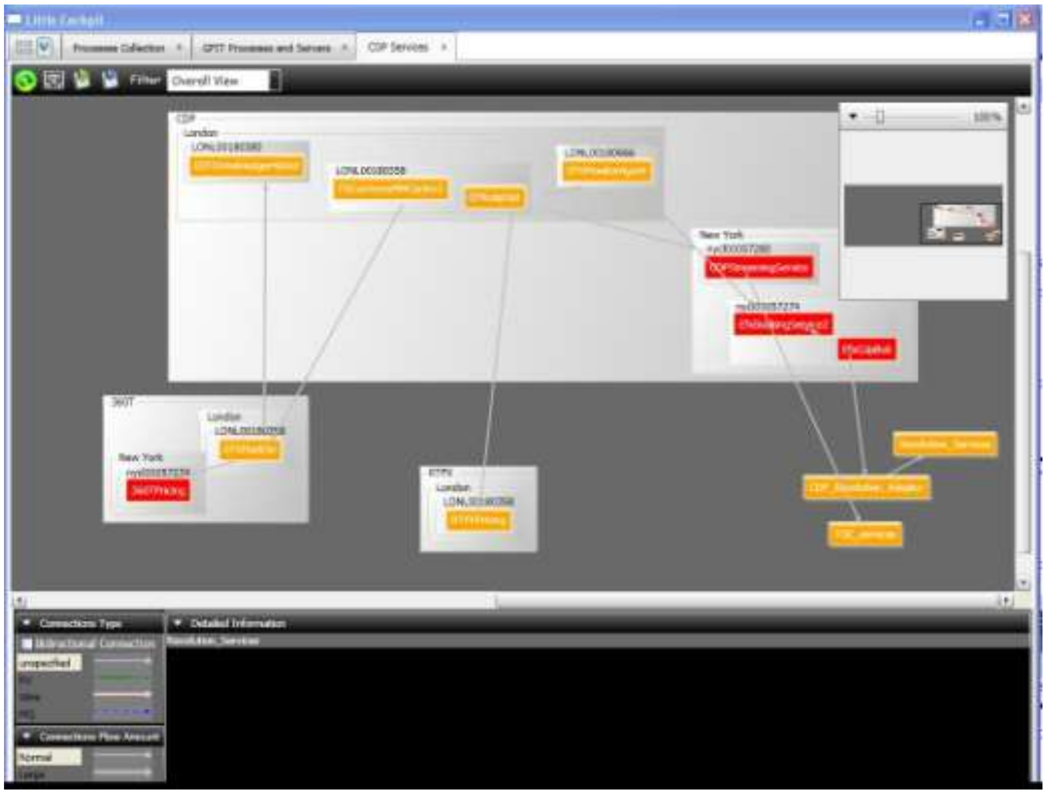


Figure 23 Cockpit Prototype snapshot IV

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.

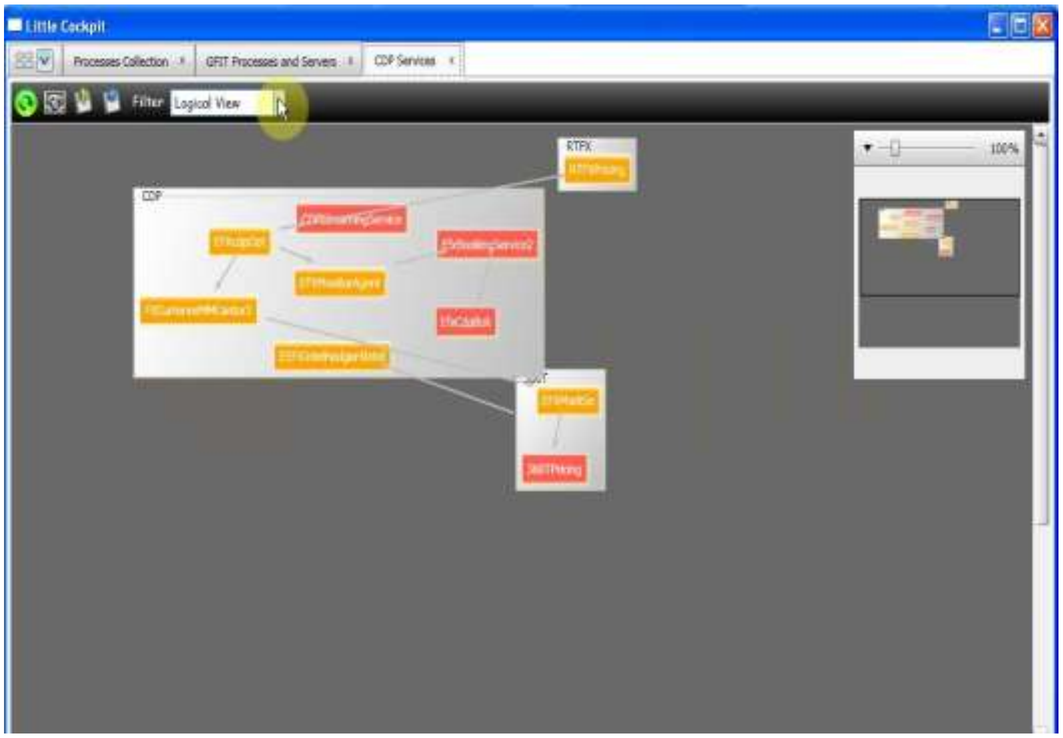


Figure 24 Logical view filter

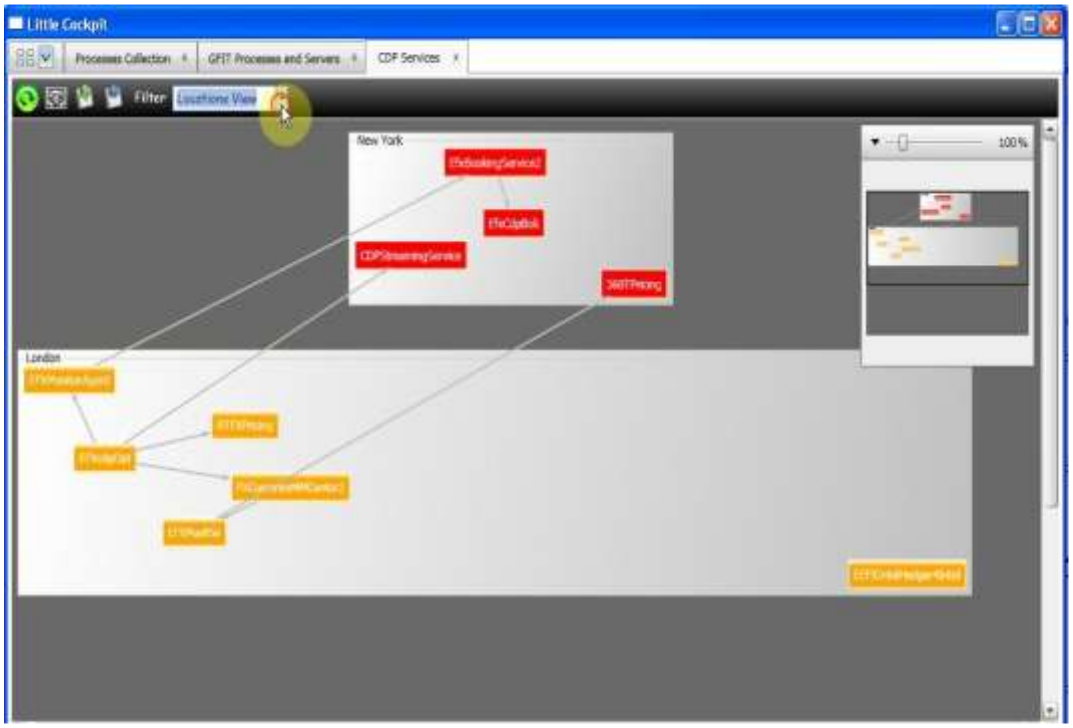


Figure 25 Physical view filter—location view

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 relationships from the system, especially for database connections. Thus, for a better management of the dataflow connections, Environment Cockpit 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 FXCurrenexMMCantor2 process to EFXMonitorAgent process, the unidirectional and large amount RV dataflow from FXCurrenexMMCantor2 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.

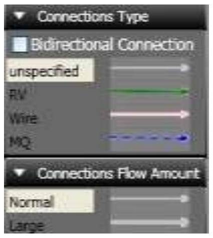


Figure 26 Connection Type

Figure 27 Connectors

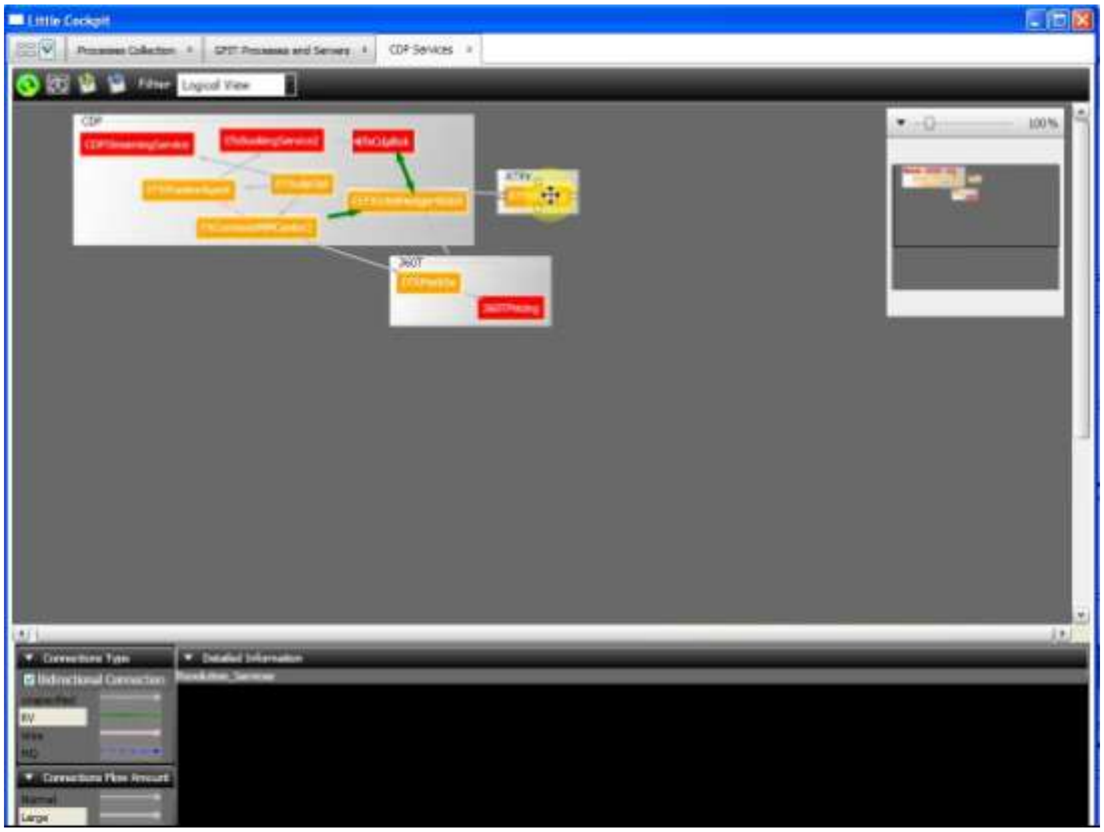
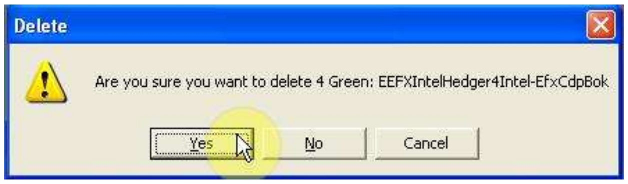
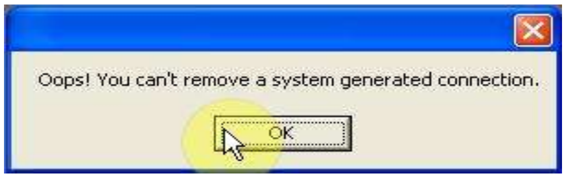


Figure 28 Cockpit Prototype Snapshot V



Are you sure you want to delete 4 Green: EEFXIntelHedger4Intel-EfxCdpBok



Oops! You can't remove a system generated connection.

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

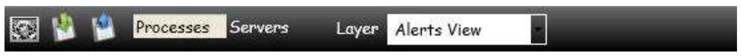
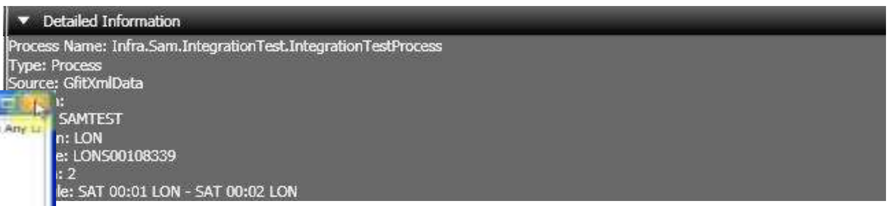


Figure 31 Load and Save button

The changed graph can be 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.

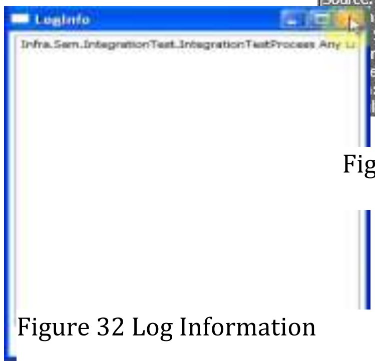
Detailed information and log files display

When users left click



Detailed Information
Process Name: Infra.Sam.IntegrationTest.IntegrationTestProcess
Type: Process
Source: GfitXmlData
:
SAMTEST
n: LON
e: LONS00108339
: 2
le: SAT 00:01 LON - SAT 00:02 LON

Figure 33 Detailed Information



LogInfo
Infra.Sam.IntegrationTest.IntegrationTestProcess Any

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.

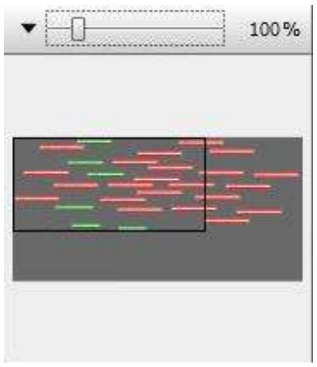


Figure 34 Zoom Box

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

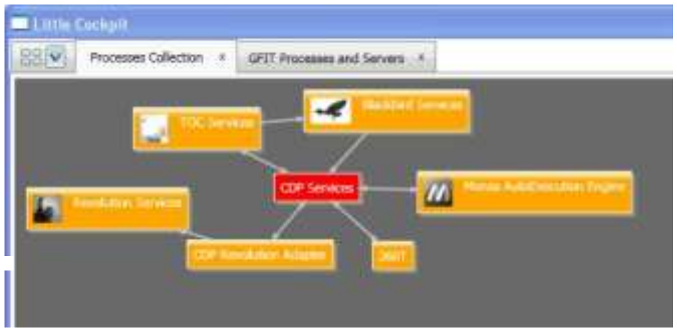
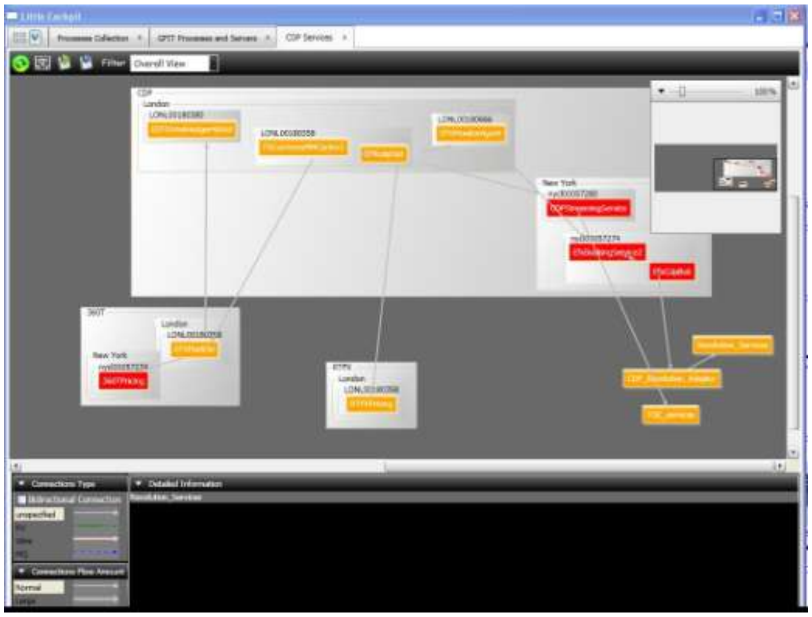


Figure 35

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

users can observe the environment in both logical and physical perspectives. In logical

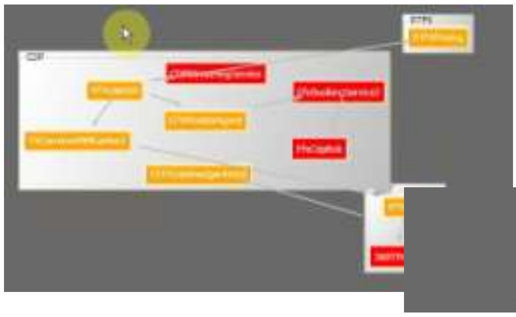


Figure 36 Logical View

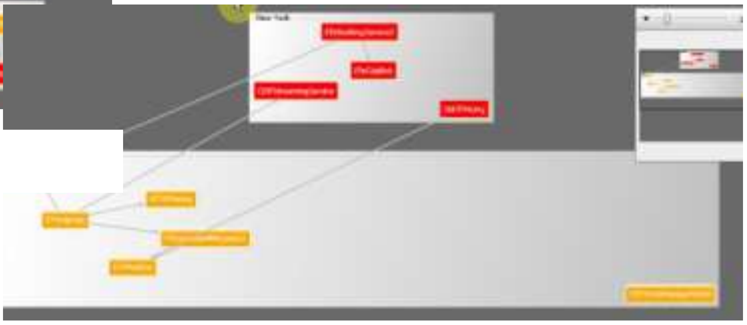


Figure 38 Physical View

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

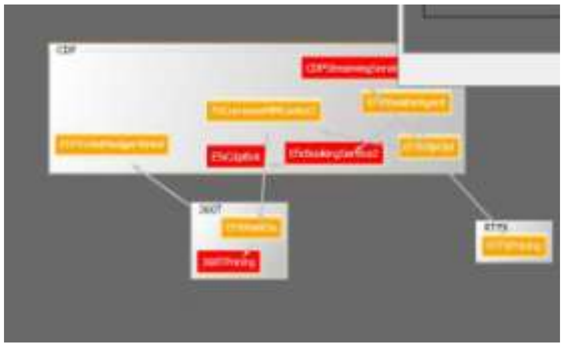


Figure 39 Adding Connection

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 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

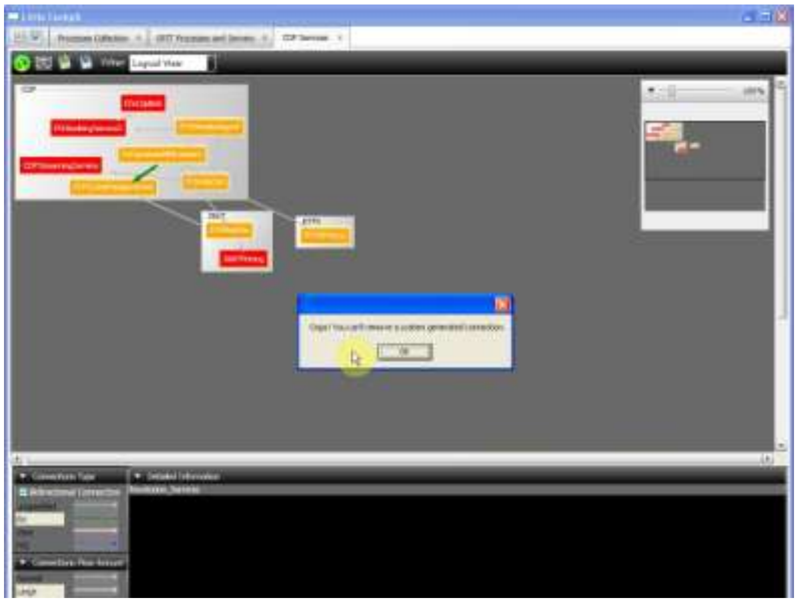


Figure 40 Delete Connection

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 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

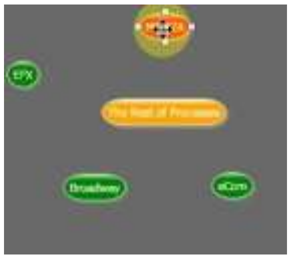


Figure 42 Gfit Processes and Servers

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 "Monza" domain, and enter the two groups, "CORE" and "MONZA" within the "Monza" domain. As Figure 40 indicates, Figure 36 both "CORE" and "MONZA" groups are red too. If the supporters double click on the "MONZA" group, all the processes within the group (Figure 41) will show up in red, meaning every item within "MONZA" group is generating alerts. This indicates the whole Monza system is down.

Figure 41 Group View of Monza

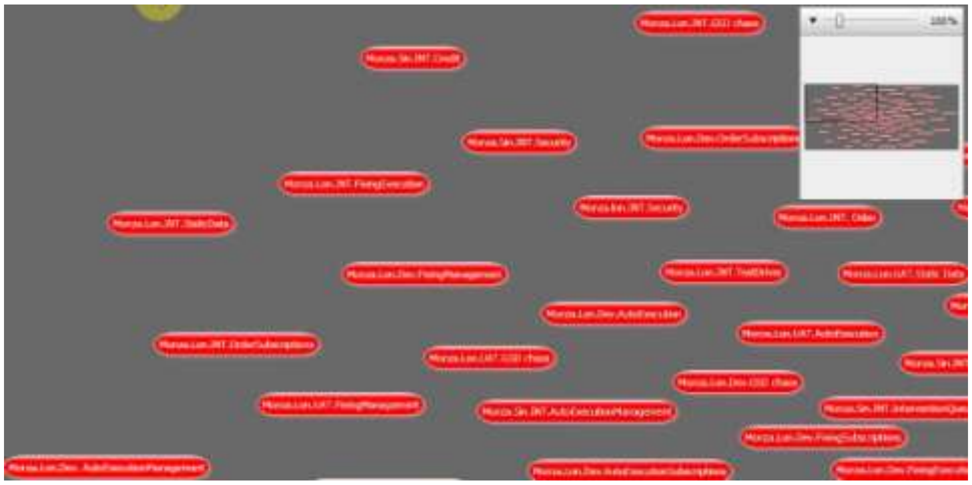


Figure 41: Double click on “Monza”

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.

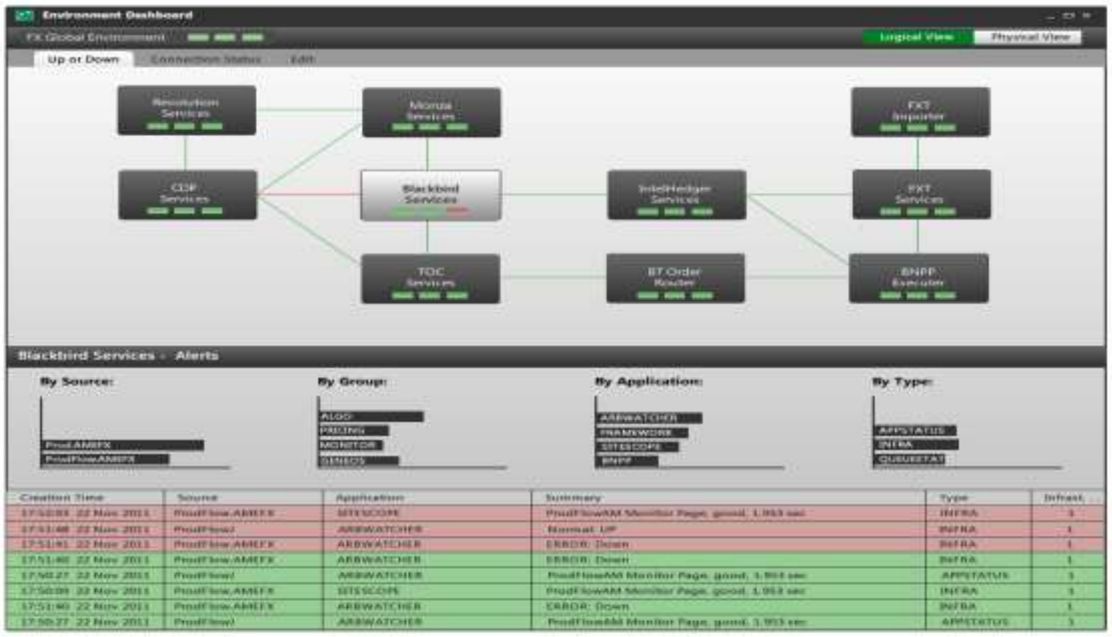
Five windows include CDPMainWindow (437 lines of code), DummyProcessCollectionMainWindow (111 lines of code), GFitWindow (494 lines of code), InfoLog (60 lines of code) and MainWindow (240 lines of code). The MainWindow window utilizes FabTab library to set up a basic tab view and then adds DummyProcessCollectionMainWindow 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. PocSerialzeHelper 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

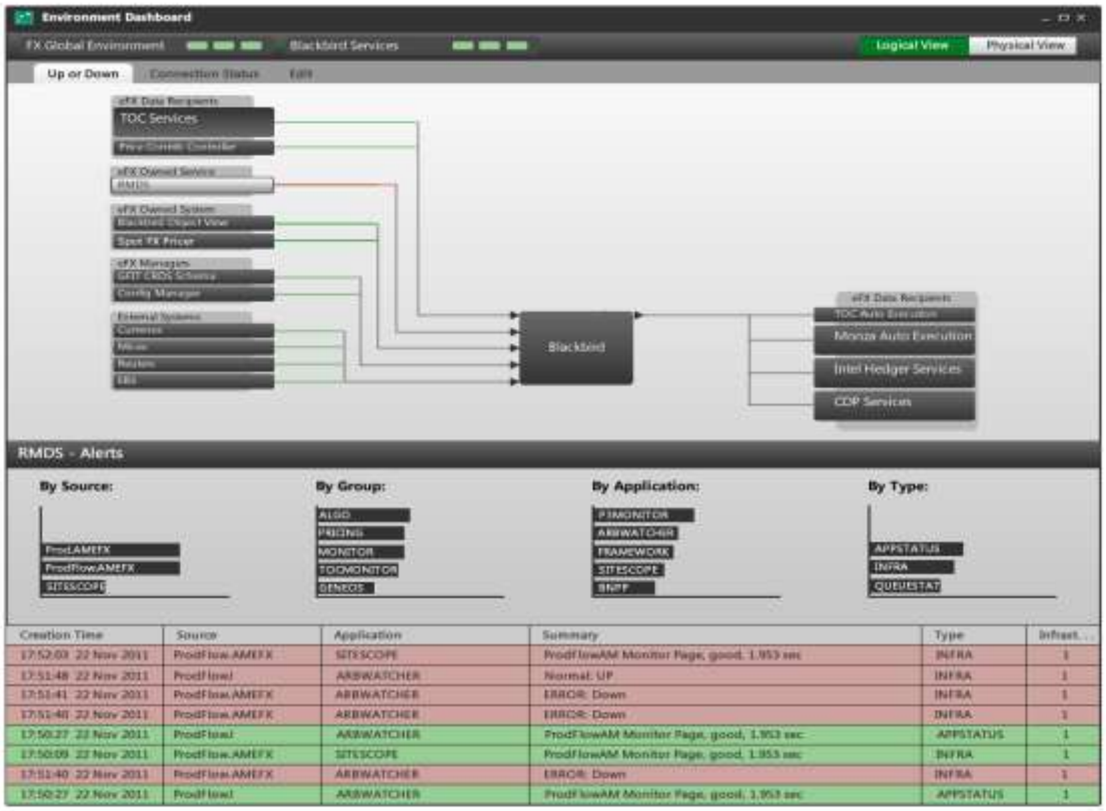
In order to define a clearer Environment Cockpit GUI design for future implementation of the real Environment Cockpit, the architecture group and our team actively interacted with UX team and the APS team supporters. The UX team and our team came up with the following GUI design.



Creation Time	Source	Application	Summary	Type	Infrast...
17:52:03 22 Nov 2011	ProdFlow.AMEFX	SITESCOPE	ProdFlowAM Monitor Page, good, 1.953 sec	INFRA	1
17:51:48 22 Nov 2011	ProdFlowJ	ARBWATCHER	Normal: UP	INFRA	1
17:51:41 22 Nov 2011	ProdFlow.AMEFX	ARBWATCHER	ERROR: Down	INFRA	1
17:51:40 22 Nov 2011	ProdFlow.AMEFX	ARBWATCHER	ERROR: Down	INFRA	1
17:50:27 22 Nov 2011	ProdFlowJ	ARBWATCHER	ProdFlowAM Monitor Page, good, 1.953 sec	APPSTATUS	1
17:50:09 22 Nov 2011	ProdFlow.AMEFX	SITESCOPE	ProdFlowAM Monitor Page, good, 1.953 sec	INFRA	1
17:51:40 22 Nov 2011	ProdFlow.AMEFX	ARBWATCHER	ERROR: Down	INFRA	1
17:50:27 22 Nov 2011	ProdFlowJ	ARBWATCHER	ProdFlowAM Monitor Page, good, 1.953 sec	APPSTATUS	1

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.



Creation Time	Source	Application	Summary	Type	Infrast...
17:52:03 22 Nov 2011	ProdFlow.AMEFX	SITESCOPE	ProdFlowAM Monitor Page, good, 1.953 sec	INFRA	1
17:51:48 22 Nov 2011	ProdFlowJ	ARBWATCHER	Normal: UP	INFRA	1
17:51:41 22 Nov 2011	ProdFlow.AMEFX	ARBWATCHER	ERROR: Down	INFRA	1
17:51:40 22 Nov 2011	ProdFlow.AMEFX	ARBWATCHER	ERROR: Down	INFRA	1
17:50:27 22 Nov 2011	ProdFlowJ	ARBWATCHER	ProdFlowAM Monitor Page, good, 1.953 sec	APPSTATUS	1
17:50:09 22 Nov 2011	ProdFlow.AMEFX	SITESCOPE	ProdFlowAM Monitor Page, good, 1.953 sec	INFRA	1
17:51:40 22 Nov 2011	ProdFlow.AMEFX	ARBWATCHER	ERROR: Down	INFRA	1
17:50:27 22 Nov 2011	ProdFlowJ	ARBWATCHER	ProdFlowAM Monitor Page, good, 1.953 sec	APPSTATUS	1

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.

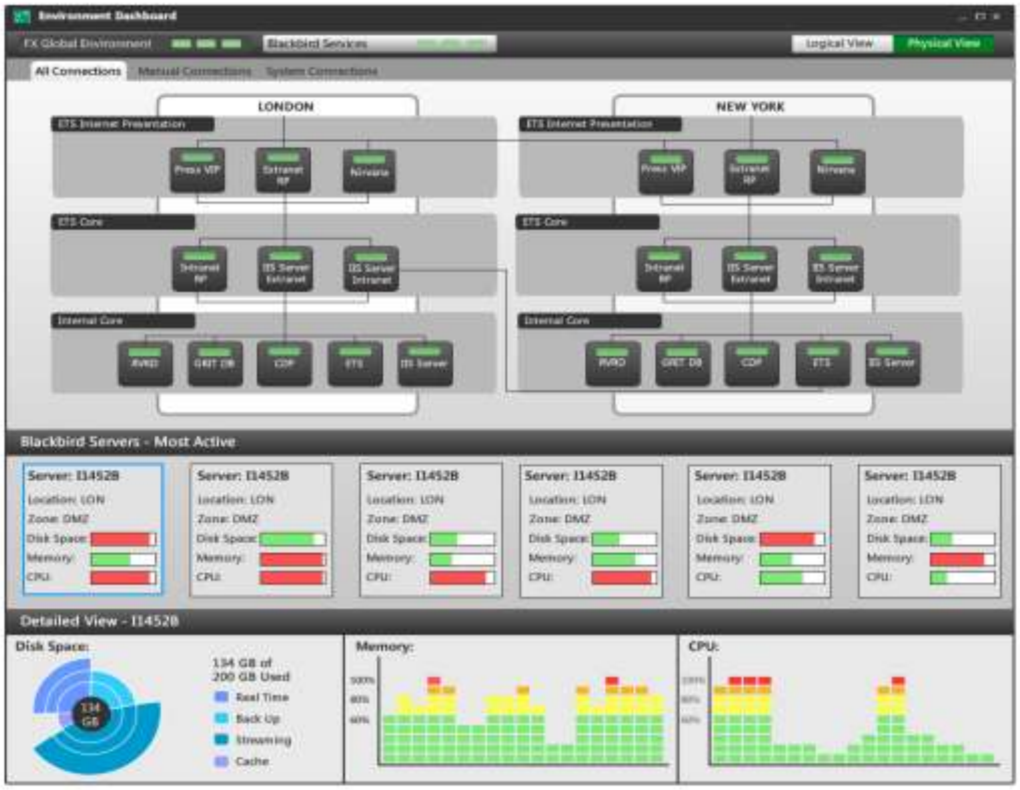


Figure 45 UX team Gui Design III

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.

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}">
```

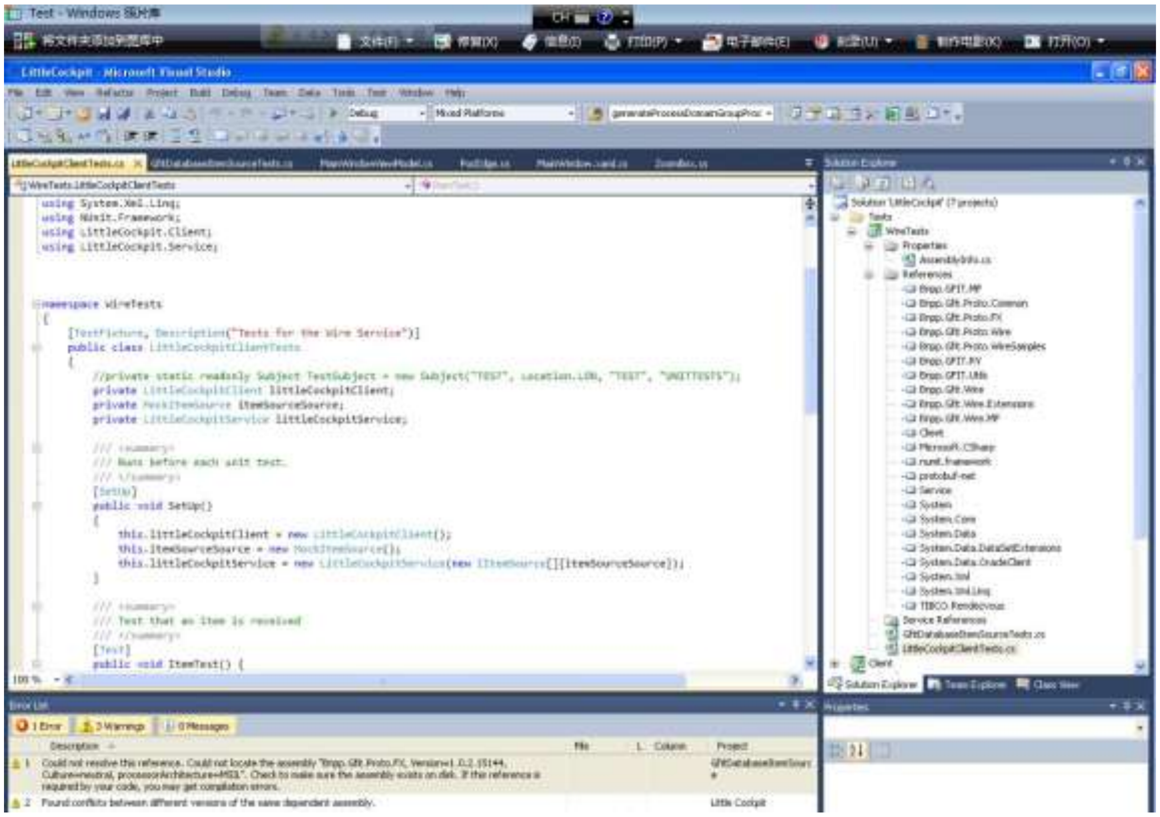
```
                <Binding RelativeSource="{RelativeSource TemplatedParent}"
                    Path="Source.(graphsharp:GraphCanvas.X)" />
                <Binding RelativeSource="{RelativeSource TemplatedParent}"
                    Path="Source.(graphsharp:GraphCanvas.Y)" />
                <Binding 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>
```

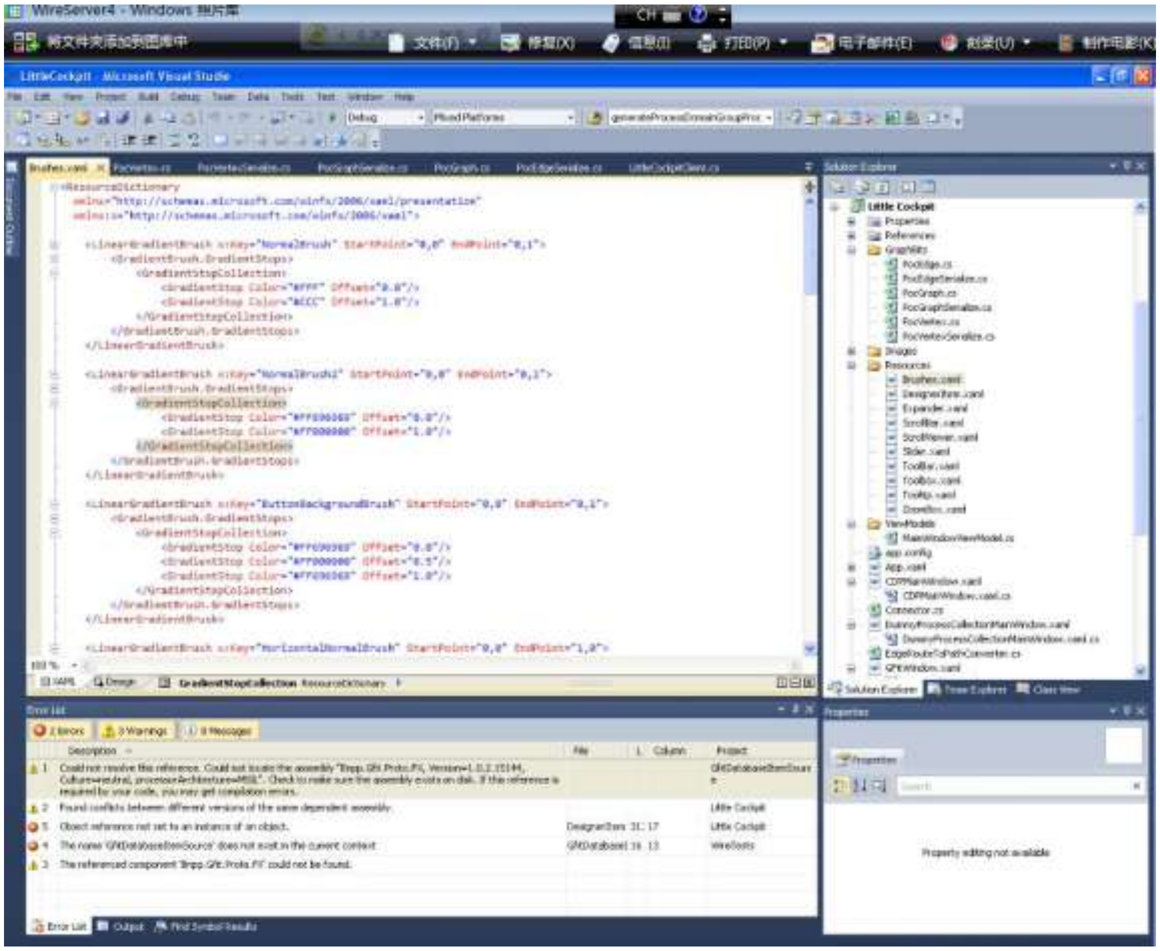
ClientTest.cs



```
using System.Xml.Linq;
using NUnit.Framework;
using LittleCockpit.Client;
using LittleCockpit.Service;


namespace WireTests
{
    [TestFixture, Description("Tests for the Wire Service")]
    public class LittleCockpitClientTests
    {
        //private static readonly Subject TestSubject = new Subject("TEST", Location.LON, "TEST", "UNITTESTS");
        private LittleCockpitClient littleCockpitClient;
        private MockItemSource itemSourceSource;
        private LittleCockpitService littleCockpitService;

        /// <summary>
        /// Runs before each unit test.
        /// </summary>
        [SetUp]
        public void SetUp()
        {
            this.littleCockpitClient = new LittleCockpitClient();
            this.itemSourceSource = new MockItemSource();
            this.littleCockpitService = new LittleCockpitService(new IItemSource[]{itemSourceSource});
        }

        /// <summary>
        /// Test that an Item is received
        /// </summary>
        [Test]
        public void ItemTest() {
```

```
1 Could not resolve this reference. Could not locate the assembly "Emgp.Glt.Proto.FX, Version=1.0.2.15144,
  Culture=neutral, processorArchitecture=MSIL". Check to make sure the assembly exists on disk. If this reference is
  required by your code, you may get compilation errors.                                        GltDatabaseItemSource
2 Found conflicts between different versions of the same dependent assembly.                    Little Cockpit
```

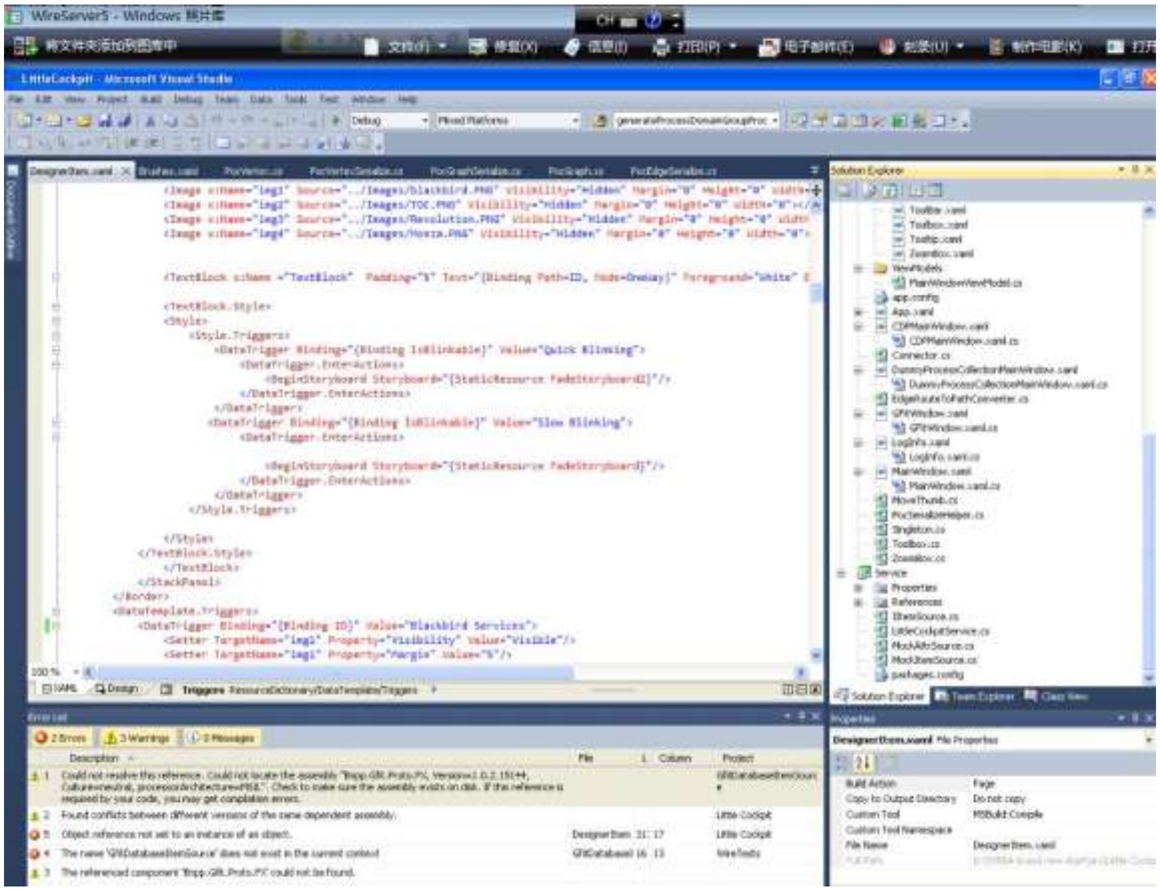
Brushes.xaml



```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

  <LinearGradientBrush x:Key="NormalBrush" StartPoint="0,0" EndPoint="0,1">
      <GradientBrush.GradientStops>
          <GradientStopCollection>
              <GradientStop Color="#FFF" Offset="0.0"/>
              <GradientStop Color="#CCC" Offset="1.0"/>
          </GradientStopCollection>
      </GradientBrush.GradientStops>
  </LinearGradientBrush>

  <LinearGradientBrush x:Key="NormalBrush1" StartPoint="0,0" EndPoint="0,1">
      <GradientBrush.GradientStops>
          <GradientStopCollection>
              <GradientStop Color="#FF090363" Offset="0.0"/>
              <GradientStop Color="#FF000000" Offset="1.0"/>
          </GradientStopCollection>
      </GradientBrush.GradientStops>
  </LinearGradientBrush>

  <LinearGradientBrush x:Key="ButtonBackgroundBrush" StartPoint="0,0" EndPoint="0,1">
      <GradientBrush.GradientStops>
          <GradientStopCollection>
              <GradientStop Color="#FF090363" Offset="0.0"/>
              <GradientStop Color="#FF000000" Offset="0.5"/>
              <GradientStop Color="#FF090363" Offset="1.0"/>
          </GradientStopCollection>
      </GradientBrush.GradientStops>
  </LinearGradientBrush>

  <LinearGradientBrush x:Key="HorizontalNormalBrush" StartPoint="0,0" EndPoint="1,0">
```

```
Error List
2 Errors | 3 Warnings | 0 Messages

   Description                                                                     File            L   Column  Project
⚠ 1 Could not resolve this reference. Could not locate the assembly "Bmpp.GN.Proto.FX, Version=1.0.2.15144,      GNDatabaseItemSource
    Culture=neutral, processorArchitecture=MSIL". Check to make sure the assembly exists on disk. If this reference is
    required by your code, you may get compilation errors.
⚠ 2 Found conflicts between different versions of the same dependent assembly.                                    Little Cockpit
● 5 Object reference not set to an instance of an object.                           DesignerItem    31  17      Little Cockpit
● 4 The name 'GNDatabaseItemSource' does not exist in the current context           GNDatabaseI     16  13      WireTools
⚠ 3 The referenced component 'Bmpp.GN.Proto.FX' could not be found.
```

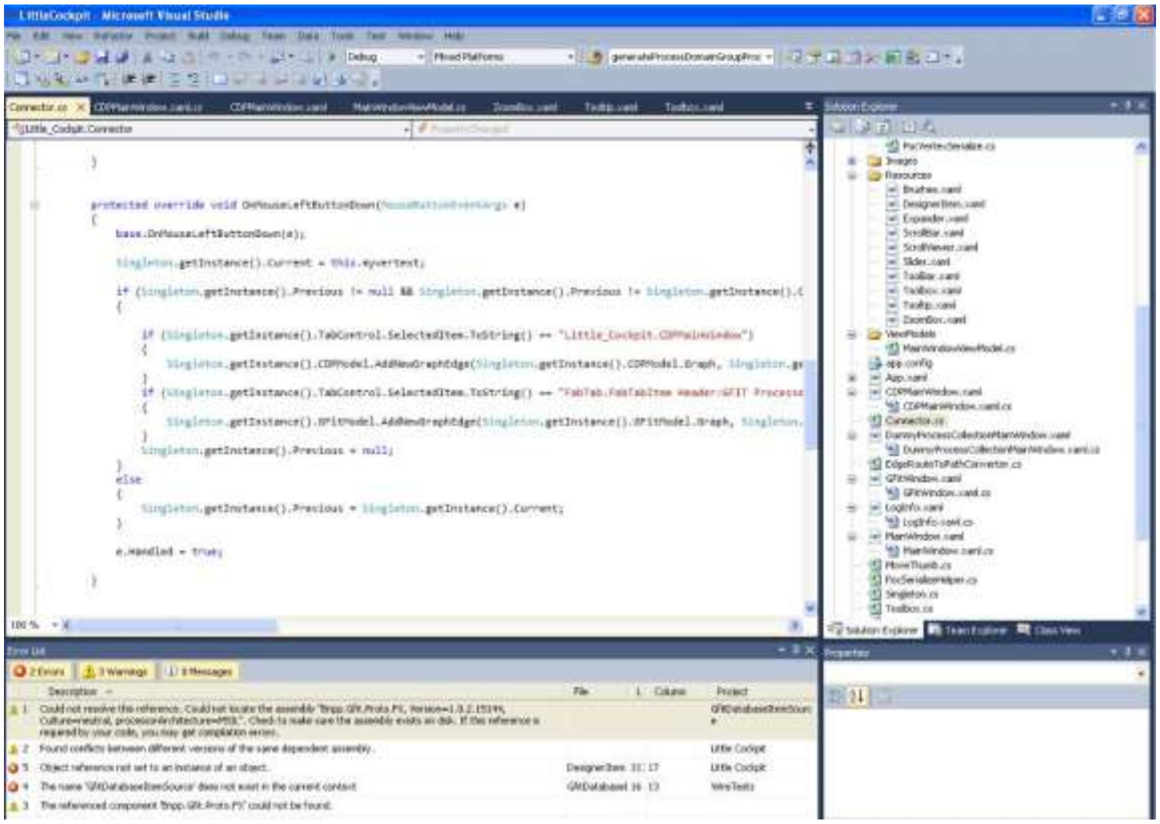
DesignerItem.xaml



```
                <Image x:Name="img1" Source="../Images/blackbird.PNG" Visibility="Hidden" Margin="0" Height="0" Width
                <Image x:Name="img2" Source="../Images/TOC.PNG" Visibility="Hidden" Margin="0" Height="0" Width="0"></
                <Image x:Name="img3" Source="../Images/Resolution.PNG" Visibility="Hidden" Margin="0" Height="0" Width
                <Image x:Name="img4" Source="../Images/Hosra.PNG" Visibility="Hidden" Margin="0" Height="0" Width="0">


                <TextBlock x:Name ="TextBlock"  Padding="5" Text="{Binding Path=ID, Mode=OneWay}" Foreground="White" 

                <TextBlock.Style>
                <Style>
                    <Style.Triggers>
                        <DataTrigger Binding="{Binding IsBlinkable}" Value="Quick Blinking">
                            <DataTrigger.EnterActions>
                                <BeginStoryboard Storyboard="{StaticResource FadeStoryboard2}"/>
                            </DataTrigger.EnterActions>
                        </DataTrigger>
                       <DataTrigger Binding="{Binding IsBlinkable}" Value="Slow Blinking">
                            <DataTrigger.EnterActions>

                                <BeginStoryboard Storyboard="{StaticResource FadeStoryboard}"/>
                            </DataTrigger.EnterActions>
                        </DataTrigger>
                    </Style.Triggers>

                </Style>
            </TextBlock.Style>
                </TextBlock>
            </StackPanel>
        </Border>
        <DataTemplate.Triggers>
            <DataTrigger Binding="{Binding ID}" Value="Blackbird Services">
                <Setter TargetName="img1" Property="Visibility" Value="Visible"/>
                <Setter TargetName="img1" Property="Margin" Value="5"/>
```

	Description	File	Line	Column	Project
1	Could not resolve this reference. Could not locate the assembly "Bapp.GR.Proto.PX, Version=1.0.2.15144, Culture=neutral, processorArchitecture=MSIL". Check to make sure the assembly exists on disk. If this reference is required by your code, you may get compilation errors.				GRDatabaseItemSource
2	Found conflicts between different versions of the same dependent assembly.				Little Cockpit
5	Object reference not set to an instance of an object.	DesignerItem	31	17	Little Cockpit
4	The name 'GRDatabaseItemSource' does not exist in the current context	GRDatabase	16	13	WireTests
3	The referenced component 'Bapp.GR.Proto.PX' could not be found.				

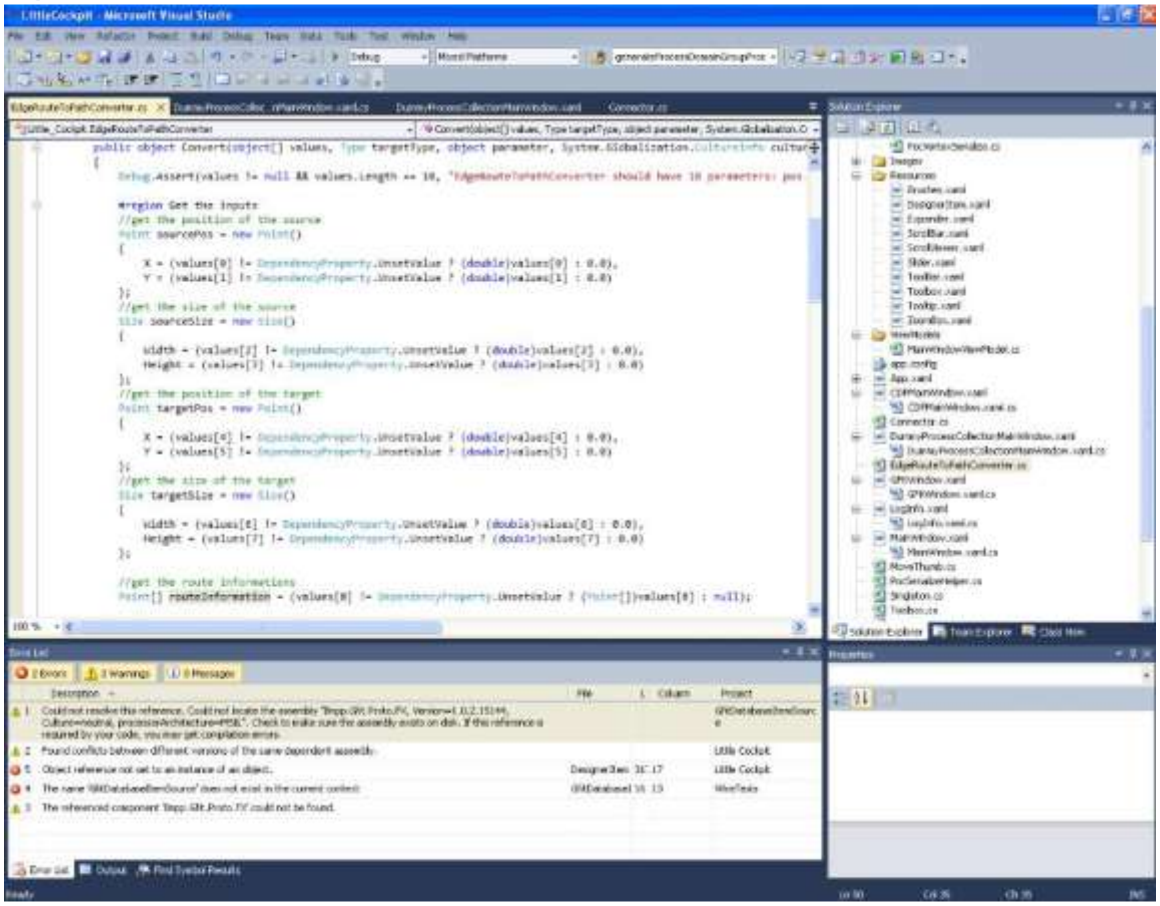
Connector.cs



```
        }

        protected override void OnMouseLeftButtonDown(MouseButtonEventArgs e)
        {
            base.OnMouseLeftButtonDown(e);

            Singleton.getInstance().Current = this.myvertext;

            if (Singleton.getInstance().Previous != null && Singleton.getInstance().Previous != Singleton.getInstance().C
            {

                if (Singleton.getInstance().TabControl.SelectedItem.ToString() == "Little_Cockpit.CDPMainWindow")
                {
                    Singleton.getInstance().CDPModel.AddNewGraphEdge(Singleton.getInstance().CDPModel.Graph, Singleton.ge
                }
                if (Singleton.getInstance().TabControl.SelectedItem.ToString() == "FabTab.FabTabItem Header:GFIT Process
                {
                    Singleton.getInstance().GFitModel.AddNewGraphEdge(Singleton.getInstance().GFitModel.Graph, Singleton.
                }
                Singleton.getInstance().Previous = null;
            }
            else
            {
                Singleton.getInstance().Previous = Singleton.getInstance().Current;
            }

            e.Handled = true;

        }
```

```
2 Errors | 3 Warnings | 0 Messages

   Description                                                                    File           L   Column   Project
⚠ 1 Could not resolve this reference. Could not locate the assembly "Enpp.GfIt.Proto.FX, Version=1.0.2.15194,    GfitDatabaseItemSource
    Culture=neutral, processorArchitecture=MSIL". Check to make sure the assembly exists on disk. If this reference is
    required by your code, you may get compilation errors.
⚠ 2 Found conflicts between different versions of the same dependent assembly.                                  Little Cockpit
● 5 Object reference not set to an instance of an object.                       DesignerItem  31  17      Little Cockpit
● 4 The name 'GfitDatabaseItemSource' does not exist in the current context     GfitDatabaseI 16  13      WireTests
⚠ 3 The referenced component 'Enpp.GfIt.Proto.FX' could not be found.
```

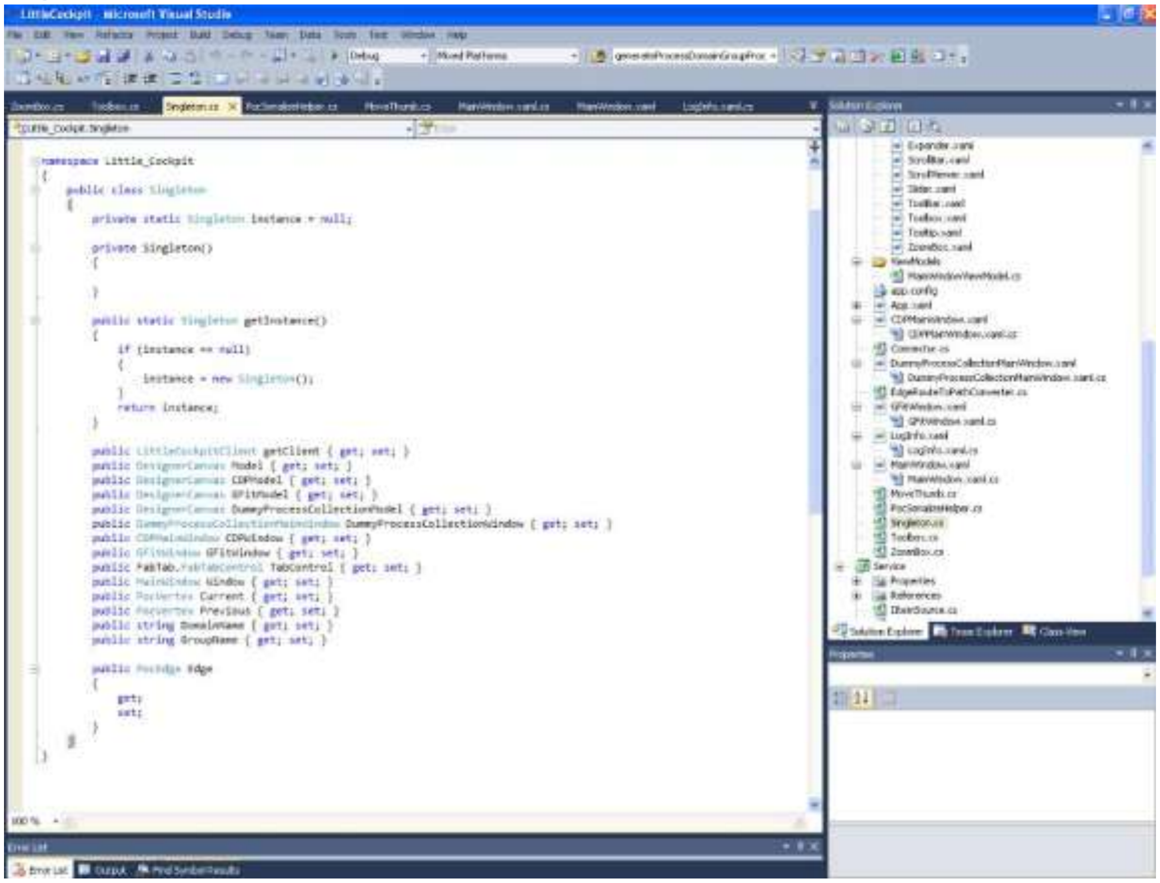
EdgeRouteToPathConverter.cs



```
public object Convert(object[] values, Type targetType, object parameter, System.Globalization.CultureInfo cultur
{
    Debug.Assert(values != null && values.Length == 10, "EdgeRouteToPathConverter should have 10 parameters: pos

    #region Get the inputs
    //get the position of the source
    Point sourcePos = new Point()
    {
        X = (values[0] != DependencyProperty.UnsetValue ? (double)values[0] : 0.0),
        Y = (values[1] != DependencyProperty.UnsetValue ? (double)values[1] : 0.0)
    };
    //get the size of the source
    Size sourceSize = new Size()
    {
        Width = (values[2] != DependencyProperty.UnsetValue ? (double)values[2] : 0.0),
        Height = (values[3] != DependencyProperty.UnsetValue ? (double)values[3] : 0.0)
    };
    //get the position of the target
    Point targetPos = new Point()
    {
        X = (values[4] != DependencyProperty.UnsetValue ? (double)values[4] : 0.0),
        Y = (values[5] != DependencyProperty.UnsetValue ? (double)values[5] : 0.0)
    };
    //get the size of the target
    Size targetSize = new Size()
    {
        Width = (values[6] != DependencyProperty.UnsetValue ? (double)values[6] : 0.0),
        Height = (values[7] != DependencyProperty.UnsetValue ? (double)values[7] : 0.0)
    };

    //get the route informations
    Point[] routeInformation = (values[8] != DependencyProperty.UnsetValue ? (Point[])values[8] : null);
```

```
2 Errors   3 Warnings   0 Messages

    Description                                                              File          L   Column   Project
1   Could not resolve this reference. Could not locate the assembly                                     GRIDatabaseItemSourc
    "Tripp.GRI.Proto.FX, Version=1.0.2.15144, Culture=neutral,                                          e
    processorArchitecture=MSIL". Check to make sure the assembly exists
    on disk. If this reference is required by your code, you may get
    compilation errors.
2   Found conflicts between different versions of the same dependent assembly.                          Little Cockpit
5   Object reference not set to an instance of an object.                    DesignerItem  31  17       Little Cockpit
4   The name 'GRIDatabaseItemSource' does not exist in the current context   GRIDatabasel  16  13       WireTests
3   The referenced component 'Tripp.GRI.Proto.FX' could not be found.
```

Singleton.cs



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

        private Singleton()
        {

        }

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

        public LittleCockpitClient getClient { get; set; }
        public DesignerCanvas Model { get; set; }
        public DesignerCanvas CDPModel { get; set; }
        public DesignerCanvas GFitModel { get; set; }
        public DesignerCanvas DummyProcessCollectionModel { get; set; }
        public DummyProcessCollectionMainWindow DummyProcessCollectionWindow { get; set; }
        public CDPMainWindow CDPWindow { get; set; }
        public GFitWindow GFitWindow { get; set; }
        public FabTab.FabTabControl TabControl { get; set; }
        public MainWindow Window { get; set; }
        public PocVertex Current { get; set; }
        public PocVertex Previous { get; set; }
        public string DomainName { get; set; }
        public string GroupName { get; set; }

        public PocEdge 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	• Name • Function/Owner • Hardware Location • Name in Config • Share • App server • DNS • Cube1/Cube2
Server	• Usage • OS • Location • Server • Cores • Memory
Process	• Status • Status Change Time • Enabled • Version • modifiedBy

	• TimeModified • Monitored? • Process Descriptor History • Invocation History • Status History • Log • Service history
Process descriptor	• 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	• 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	• Subject • Server Name • Server id • Process name • Service Type • Expected HBs • Received HBs • Last HB

	• Status

Appendix C – Timeline

10/24/11(Monday)-10/28/11(Friday)	**Get familiar with the exisiting 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