CatRange 3.0 Architecture Options & Recommendations

April 20th 2009

Rajiv Rajak
Senior Solutions Architect
Aon Benfield
rjaiv_rajak@aon.com

Jason Lind
President / Chief Software Architect
Lind Innovation
jason@lind-i.com

Introduction

CatRange refers to the portion of the proposed CatView product which will aggregate and geocode projected loss data for presentation on the CatView Portal. The product will take intermediate exposure data, currently generated by CatExpress, which it will then map to Loss data, produced by RMS, AIR and IF modeling software, and aggregate the data by products, divisions, regions, zones and various geo levels. CatRange will then export the results to the CatView Portal for review and display.

CatRange's current production release is 2.2 with 2.3 under active development having a target release of early Q3 2009. The current architecture is a client driven model, implemented with .NET 3.5 in Windows Forms, supporting SQL Server 2000 SP4 and SQL Server 2005 SP3. The current model requires end users to attach the following databases: EIDM's and CPDM's (generated by CatExpress) , RDM's (generated by RMS RiskLink), Air Area Code, Loss and Exposure (generated by AIR Classic/2) and IF Results (generated by IF). The user is then required to export the results of the run from the CPDM to flat files which are then uploaded to CatPortal through CatInHat over SFTP.

The primary motivation behind CatRange 3 is to support the new data model under development for the CatView product. This presents us with the opportunity to make significant changes to the architecture that will decrease total cost of ownership and increase reliability, performance and usability.

Architecture Overview

Assumptions

The proposed architectures assumes that CatRange 3.0 will be a server centric application, unlike previous versions the bulk of the work will be performed on centralized servers and not client machines. This approach was decided upon for several reasons.

With the previous versions of CatRange a massive amount of effort is required diagnosing and solving environment specific related issues, including operating system and RDBMS service packs as well as differences in hardware and drivers. A controlled environment will result in a lower cost of ownership as well as increased performance. Management of decentralized data has proven to be a major hurdle with the current architecture with not only the network latency issues inherent to moving large data sets from one machine to another, but also obtaining and maintaining a complete picture of the physical locations of data has proven to be very difficult and human resource intensive. Monitoring the status and progress of client runs is another major challenge with the current architecture.

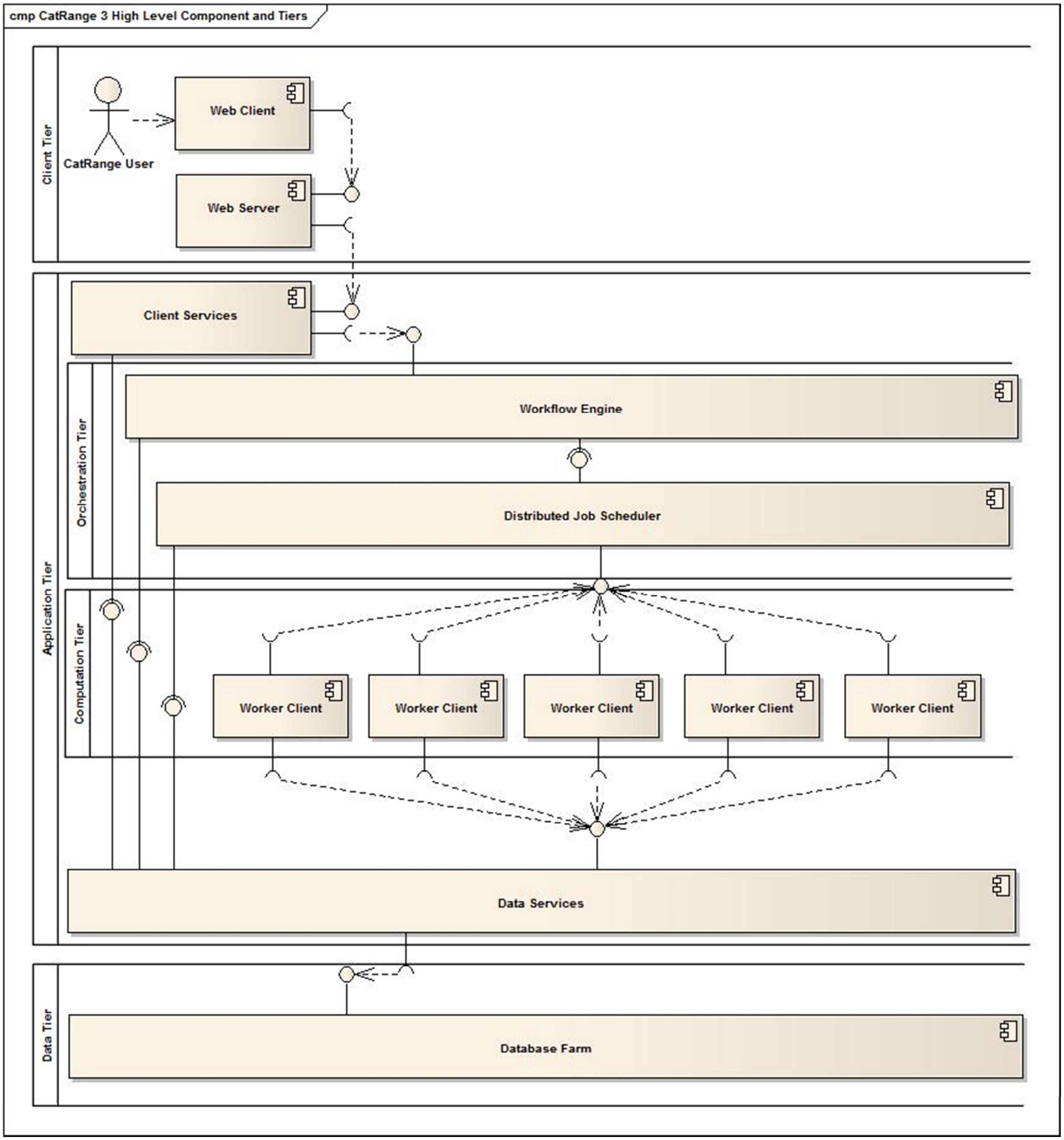
CatRange by nature is highly computational intensive product, and with the current version the team has made major strides in implementing multi-threaded parallelization of selected portions of the application which has significantly reduced overall run time of the product. In order to maintain this performance, and indeed improve it, a server centric architecture will need to implement portions of the application in a distributed environment. There are several options to making this reality, but a primary assumption of this architecture is the existence of a "computational cloud" which will allow the application to schedule worker processes without the knowledge of how and when those processes will be executed.

High Level Architecture

The following diagram is a high level component diagram of the proposed architecture. Regardless of implementation of any of the specific components the structure and connections of the components, as illustrated, would seem to be the best solution keeping in mind the aforementioned assumptions. The tiers represent a physical separation of farms of servers with those connections communicating via remote calls, such as web services or .NET Remoting.

The components can represent logical class libraries or entire systems depending on their context. A dashed line with an arrow describes remote communication between two components where as a circle surrounded by a semi circle describes an assembly reference. These components will need to be composed at a further granularity once the underlying implementation technologies have been identified and the full requirements defined.

HIGH LEVEL ARCHITECTURE COMPONENT DIAGRAM



HIGH LEVEL ARCHITECTURE COMPONENTS AND IMPLEMENTATION OPTIONS

CLIENT TIER / WEB CLIENT

DESCRIPTION

The Client Tier describes the system that the end user is using to interact with the application. The Web Client component is responsible for presenting content delivered by the Web Server component and enabling the end user to configure, manage and monitor CatRange client runs.

IMPLEMENTATION OPTIONS

There are two primary implementation options for the web client that have been identified, web browser and Silverlight 2 or 3. The web browser solution is inherently cross browser and OS as the web server would generate industry standard HTML which is more or less supported across the board. Silverlight is also cross browser OS although those OS choices are more limited in that Microsoft directly supports Silverlight on Internet Explorer 6+, Firefox 2+, Safari and Chrome on Windows and Firefox and Safari on Mac OS X.

Flex/Flash is an alternative technology to Silverlight, better supported with a much larger install base however limited in features to comparisons. Java Applets and Active X are other options however with even a more limited feature set. Another real option would be a more traditional smart client, such as XBAP or even Windows Forms, however that would not be cross browser or OS and the implementation features and effort would be similar to that of Silverlight.

BROWSER BASED

With browser based applications there are two primary architectures to consider: forms based and AJAX. In a forms based model the browser will perform a synchronous post back when requesting or interacting with the data, causing the page to reload. AJAX, Asynchronous Java and XML, on the other hand makes an asynchronous call to a web services through JavaScript and then renders the response into HTML and interjects it into the appropriate place on the page. AJAX models provide a richer overall experience as well as reduced overhead on the web server, however developing such applications can be very challenging especially ones that are cross browser.

SILVERLIGHT 2/3

Silverlight 2 is a Microsoft product released in Q3 of 2008 that enables developers to create rich cross browser and OS applications without the limitations of HTML markup. Silverlight UI's are defined in XAML, which is a highly featured XML based vector markup language. The UI logic of the application is written in any .NET language with a limited implementation of the .NET framework that will run on the clients machine, much of this logic would have to be in JavaScript in a browser based environment which has serious feature, performance and reliability issues when compared to .NET languages such as C# and VB.NET.

Silverlight 3 is currently in Beta, with the first version released earlier this month, and is scheduled to go into production sometime towards the end of the year. Silverlight 3 introduces many new features including 3D UI, improved render effects, support for the concept of themes, over 60 new and improved controls and improved data binding and performance.

IMPLEMENTATION RECOMMENDATION

A browser based application ensures maximum compatibility, especially in the mobile device space. However creating a rich interactive experience in a browser is much more difficult than in Silverlight. We recommend that CatRange 3 primarily target Silverlight 3 and also provide a browser based application with a limited feature set. The browser application would not leverage AJAX and would be specifically targeted for mobile devices. There is some risk in developing with a product that is still in beta, however Microsoft has committed to minimizing the changes between the Beta and RTW. The advantages of Silverlight 3 over 2 are significant and the target release date is well before the target release of CatRange 3.0.

In this implementation the browser component is optional, however strongly recommend as the use of smart phones is certainly on the rise. The dual technology implementation will result in the best of both worlds: a rich and highly interactive application for configuring and managing runs that can be easily developed and an interface for gaining mission critical monitoring on mobile devices. With the correct architecture the code duplication between the two implementations can be greatly minimized.

HIGH LEVEL ARCHITECTURE COMPONENTS AND IMPLEMENTATION OPTIONS

WEB SERVER TIER / WEB SERVER

DESCRIPTION

The Web Server tier is the server, or farm or servers, responsible for handling requests from the web client. The Web Server's role is slightly different depending on a browser based or Silverlight based application. In a browser based model the server will generate HTML for consumption by the browser. In a forms based model the bulk of the HTML will be generated on the server and in an AJAX application the servers role is to return HTML for injection and highly specialized XML for the JavaScript to interpret into HTML, depending on the implementation specifics. In a Silverlight model the web server will provide data for the Silverlight client to consume. In both models the web server has the role of providing authentication and authorization to the web client.

IMPLEMENTATION OPTIONS - BROWSER BASED

In the web browser model the web server is primary used for rendering data into HTML, as well as authentication and authorization. There are many technologies and platforms that perform this, Java J2EE/JSP, PHP and Ruby on Rails just to name a few, however Microsoft's ASP.NET offers one of the best in class solutions on the market. There are several flavors of ASP.NET currently available, ASP.NET Web Forms, ASP.NET MVC and ASP.NET AJAX.

The other option to consider is a stock portal application such as SharePoint, DotNetNuke or Oracle OBIE. The problem with stock portal applications is that, although there is relatively little effort needed to develop about 90% of the application, the last 10% involves heavy customization and workarounds to get the application to function to requirements. The other issue is that these kind of software tends to lead to applications that feel like a portal product, and not a product designed specifically for its purpose. These specific issues have been common problems with the current CatPortal product and we highly recommend that this route not be considered.

ASP.NET WEB FORMS

ASP.NET Web Forms is the most often used implementation of ASP.NET and has existed in some form since .NET 1.0. Web Forms provides not only a clean separation of UI and business logic, but also a separation of UI design and logic. The major concept in ASP.NET Web Forms is that each page is constructed from a UI file, which is a mix of HTML and custom control XML markup, and a code behind, written in a .NET language, that processes logical events and manipulates the objects represented in the UI markup. Web Forms has a defined page event lifecycle which raises events at defined stages of the page that generate and render the controls into HTML. In the web forms model the client uses a form post back to interact with the event life cycle, for example clicking a button raises its click event. The business and data tiers are called from the code behind and the code behinds sole purpose is to manipulate the UI and not business logic.

ASP.NET MVC

MVC is a relatively new development in the ASP.NET family, released in Q1 2009. This framework leverages a model-view-controller pattern that helps map actions on the page to friendly URLs that process the request. A major advantage to this model is in unit testing where, unlike Web Forms, UI operations can be easily tested in an automated fashion. The big disadvantage is that there is no event life cycle and data binding is performed through code more reminiscent of the Classic ASP/JSP/PHP paradigm. MVC is the best option for creating highly interactive AJAX type applications as the model allows for much better componentization of parts of pages when rendered by JavaScript while maintaining strict control over markup.

ASP.NET AJAX

ASP.NET AJAX was released in Q1 2007 as a way to implement AJAX applications in a model more consistent with Web Forms. While development with this technology is generally less effort than on an MVC pattern there are major detractors. Customizing the AJAX behavior, especially for highly interactive options such as drag and drop, is not well support out of the box. In order to render any portion of the page the server side must complete the full event lifecycle greatly degrading performance. ASP.NET AJAX has proved very useful for adding small AJAX functionality to existing web sites, such as cascading dropdown lists and validation, however has not proved to be a solution for highly interactive applications.

HIGH LEVEL ARCHITECTURE COMPONENTS AND IMPLEMENTATION OPTIONS

WEB SERVER TIER / WEB SERVER

IMPLEMENTATION OPTIONS - SILVERLIGHT

Silverlight clients support two primary methods of interacting with the web server: asynchronous SOAP web services and sockets. Sockets provide the ability for the server to push information to the client in real time, with a significant cost in server side overhead in compassion to web services. Web services is the route taken by most Silverlight applications as the client requesting data is a more familiar and understood paradigm. There are many options for SOAP web service implementations including J2EE, BizTalk and even SQL Server 2008. The more common solutions in the Microsoft world are ASP.NET Web Services and Windows Communication Foundation. For sockets generally a custom service is needed to route events between the sockets open between the clients and server, although industry standard implementations such as TIBCO are also possibilities. The sockets model is not necessary with CatRange and will not be discussed further.

ASP.NET WEB SERVICES

ASP.NET Web Services has existed in some form since .NET 1.0 and has been widely used as a SOAP client in the Microsoft world. This product is hosted by IIS and abstracts XML Serialization and SOAP wrapper creation from the implementation. The product was a major advancement in web service development when released, however there has always been serious criticisms over the lack of control of the SOAP envelope markup. Web Services Enhancements was first released in Q4 2002 and provided primarily security enhancements to ASP.NET Web Services such as encryption. WSE, while addressing some serious limitations of ASP.NET Web Services, proved very difficult to implement and work with.

WINDOWS COMMUNICATION FOUNDATION (WCF)

WCF, released as a core part of .NET 3.0 in Q4 2006, represents a major innovation in the world of web service clients. The primary feature with WCF is that the authorization and markup are completely controllable through the configuration file, so the same logic could deliver SOAP, JSON in various encrypted capacity without modifying the underlying code base. Since these are configuration files these changes can actually be made at run time without recompilation. WCF service libraries can be hosted not only by IIS, but indeed by anything that can spawn a process with the relevant permissions. Most WCF services are actually deployed as Windows Services.

Another major change is that the operations contract, the methods exposed by the service, are implemented as interfaces and the same contract can be implemented in different ways and the configuration will control which implementation is called. For example there might be a read only service and a read/write service implementation, and based on the authorization one or the other is used. The high degree of configurability combined with the ease of implementation makes WCF the ideal candidate for most applications.

AUTHENTICATION AND AUTHORIZATION

All versions of ASP.NET have the same authentication and authorization architecture. ASP.NET supports a pluggable provider model of membership providers and role provider, each of which can be custom implemented to support the necessary identity and authorization systems such as Active Directory. The exact identity and authorization systems being used are out of scope for this document, however whichever ones are chosen the provider model will support. The authentication and authorization components will be componentized so that other tiers can leverage that logic.

IMPLEMENTATION RECOMMENDATION

For a highly interactive browser based application ASP.NET MVC is probably the best option on the market. However, as discussed in the web client section, we recommend that the browser solution only target a small subset of the requirements and leave the highly interactive portions to the Silverlight client. For this kind of application ASP.NET Web Forms is the best fit. For the Silverlight application WCF, while not the only option, will provide the best flexibility in terms of implementation. The ASP.NET Web Form and Silverlight client/server, while not able to use the same control logic, will be able to share much of the same underlying logic in common components which in turn communicate with the Application Tier.

HIGH LEVEL ARCHITECTURE COMPONENTS AND IMPLEMENTATION OPTIONS

APPLICATION TIER

The Application Tier is the physical tier where business logic resides. The Orchestration and Computational Tiers, while not residing physically on the same server farm as the Application Tier, are logically considered part of this tier.

APPLICATION TIER / CLIENT SERVICES

DESCRIPTION

Client Services provides a standard interface for the Client Tier components to communicate with. In addition to supporting the propose dual browser / Silverlight implementation proposed this will serve as a way for unanticipated future clients to interact with CatRange, which includes other projects in the proposed CatView product. This component has the responsibility of routing requests to the Workflow Engine or Data Services components or some combination thereof. This will abstract the low level implementation details from the implementing clients and maintain control over the business logic.

IMPLEMENTATION RECOMMENDATION

WCF, as described on page 6, is the perfect candidate for the contract portion of this logic. The business logic itself, defining validation and external communication, will be further componentized to support reuse without a making a costly service call. This will support the possibility of hosting this portion of the logic on the Web Server Tier in addition to the Application Tier, which may or may not improve performance. Further analysis will be required to make this determination. The Client Services component will have an assembly reference to the Data Services logic and a service reference to the Workflow Engine component.

APPLICATION TIER / DATA SERVICES

DESCRIPTION

Data Services provides a standard interface for data centric activates such as querying, transforming and transferring data. This component has the responsibility of forming and executing queries on the Data Tier as well as mapping data objects to those operations.

IMPLEMENTATION ANALYSIS

The specific implementation of this logic will heavily depend on the database platform and data model design. For a data model similar to the one currently in use, where logical data components are separated into different logical databases, no Object Relational Mapping tool on the market is sufficient for our purposes. The problem is that ORM's generally assume two requirements which are invalid in the current CatRange environment, the data objects reside in the same physical database and the table names are constant.

ORM's build their queries from some sort of mapping definition, these mapping definitions are then used to build the actual query syntax using object oriented concepts. The SQL, regardless of variant, has a short cut for identifying tables in the schema based on the context of the query, querying tables in different schemas requires a fully qualified syntax which most ORM's do not support. The table name invalidation comes from AIR, where analyses are placed in their own tables, which while having a constant schema, the names of these tables are dependent on the data. Again most ORM's do not allow for the names of the tables to be changed at run time.

The current version of CatRange makes heavy use of on the fly dynamic SQL, which has proved very difficult to not only develop and maintain but also debug. The problem is dynamically generating SQL, either in stored procedures or in code, is not compile time checked and very difficult to test. What we propose is a generalized dynamic Query Engine framework, the foundation of which already has been implemented in CatRange 2.3, that will dramatically increase reliability, performance and development effort of the data centric portions. With no known third party solution this seems to be the only viable option. The alternative of manually developing and maintaining dynamic SQL logic does not seem to be a realistic solution that will lead to a stable and reliable product.

Although this component should be developed with the future possibility of supporting multiple RDMS in mind, for now development should focus on the RDMS solution chosen by the CatView team.

HIGH LEVEL ARCHITECTURE COMPONENTS AND IMPLEMENTATION OPTIONS

APPLICATION TIER / DATA SERVICES

IMPLEMENTATION RECOMMENDATION

Like the Client Services component, as described on 7, it is recommended that the Data Services contract is implemented in WCF, as described on page 6, and then further componentized to support reuse without the overhead of web services. This is key so that the Client Services and Workflow Engine components will have an assembly connection to this component, while the Worker Clients will have a web service connection. The Data Services component must have direct access to the Data Tier where it is unwise for the Worker Clients to have the same as depending on where those are hosted it could pose a serious security risk. The Query Engine will be componentized in such a way for reuse in unrelated projects and the CatView solution in general.

ORCHESTRATION TIER

The Orchestration Tier is responsible for managing the logic flow of the core long running processes CatRange requires. The tier should be responsible for managing the overall state of a run, providing automated validation of data, distributing status and error message and provide the ability to suspend and roll back individual runs.

ORCHESTRATION TIER / WORKFLOW ENGINE

DESCRIPTION

A Workflow Engine is a product that abstracts the flow of logic from its implementation. This is very useful in many long running processes as the logic defects are generally not related to the flow of logic, so the logic itself can be changed and componentized separately from the flow of that logic. There are many workflow orchestration engines on the market, several implement industry standard XML languages such as XPDL and several proprietary implementations such as Windows Workflow Foundation and BizTalk.

WHY A WORKFLOW ENGINE?

A major problem faced in the current version of CatRange is there are an enormous amount of possible combinations of model configurations, and while it attempts to support many of these combinations all of the combinations have not been and tested and result in invalid output due to invalid assumptions about the data model. Since Aon does not control the output of RiskLink and AIR Classic, and a different team is responsible for Impact Forecasting as well as the unavailability of data at many of these combinations, this is not a problem that can be addressed through anything but informed trial and error. The iterative process of running new combinations, discovering output issues and addressing the issues through modifications to the business logic and queries will probably never go away.

What can be addressed is the process through which the changes are made, currently a change to any portion of CatRange's logic at the very least means the run needs to be restarted from scratch and likely means a recompile requiring a release process. This has resulted in an unacceptable delay in providing loss data on CatPortal. A proper implementation of an Workflow Engine will allow fundamental changes to be made to the mission critical business logic and restart a run from the necessary point. For example if there is nothing wrong with the initial data transformations there is no reason to rerun that portion of the work flow if the problem is in the computational portion or output transformations. It can even be granularized to level where if only a specific combination was effected by the problem, the other combinations will not have to be rerun. This can potentially save days of time lost under the current model.

IMPLEMENTATION OPTIONS

WINDOWS WORKFLOW FOUNDATION

WF, released as a core part of .NET 3.0 in Q4 2006, is a very new concept for most developers and has seen limited implementation thus far. It is arguable that very few applications warrant this separation between logic flow and implementation, as discussed about CatRange certainly justifies this kind of separation. WF can be thought of as flow charting with a code behind, very much like ASP.NET applications only the markup is controlling the events, not an underlying page event lifecycle or user interaction. Workflows in WF are written in the same XAML syntax which Silverlight and WPF use. One limitation of WF in .NET 3.0 addressed in 3.5 is that web services, both for communicating to the workflow and the workflow communication to other components, is now fully supported out of the box. WF has support for workflow persistence which enables transactional activities that can be rolled back and restarted.

HIGH LEVEL ARCHITECTURE COMPONENTS AND IMPLEMENTATION OPTIONS

ORCHESTRATION TIER / WORKFLOW ENGINE

IMPLEMENTATION OPTIONS

BIZTALK

BizTalk is a business process management server which has a modular adapter architecture providing standardized integration between a large variety of products. The primary interest in BizTalk from CatRange's perspective are the built in and third party workflow automation components available in the product. However conceptually BizTalk could be leveraged to perform integration between the various components of the CatView product and could also be used to manage the communication between the various tiers within CatRange. BizTalk adapters also exist for various distributed computing schedulers and could be useful there as well.

The problem is that although adapters exist that would provide much of the functionality out of the box that would need to be custom developed in WF, configuring and architecting these components is not trivial. In order to implement BizTalk a seasoned architect with extensive BizTalk experience would be required, and those resources are very expensive. Training Aon Benfield employees to support and manage a BizTalk installation is again not a trivial matter. Another downside is that in its current release, version 2006 R2, only .NET 3.0 is supported and Visual Studio 2005 is required, meaning that our core components could not leverage the 3.5 framework and language enhancements, let alone .NET 4.0. BizTalk Server 2009 is slated for Q4 2009 and will support .NET 3.5 and Visual Studio 2008, however many existing third party adapters will not be immediately available and therefore is unlikely to be a viable option for this solution.

IMPLEMENTATION RECOMMENDATION

Further analysis is required before making an implementation recommendation, BizTalk may indeed be the best solution for this product however without a BizTalk architect's analysis it is not possible to perform an adequate comparison to WF. We recommend that a BizTalk architect be brought in for a high level analysis of BizTalk's capabilities with respect to CatRange and CatView in general.

ORCHESTRATION TIER / DISTRIBUTED JOB SCHEDULER & WORKER CLIENT

DESCRIPTION

According to SearchDataCenter.com: "A job scheduler is a program that enables an enterprise to schedule and, in some cases, monitor computer "batch" jobs (units of work, such as the running of a payroll program). A job scheduler can initiate and manage jobs automatically by processing prepared job control language statements or through equivalent interaction with a human operator. Today's job schedulers typically provide a single point of control for all the work in a distributed network of computers."

In the context of CatRange, the Distributed Job Scheduler is responsible for instructing the Worker Clients to perform the computational intensive calculations that will be displayed in CatView. The scheduling mechanism must abstract resource allocation and availability, so that from the Workflow Engine's perspective the jobs it schedules are merely asynchronous operations. The Workflow Engine will use the Distributed Job Scheduler to schedule various combinations of events that need to be processed and the Worker Client will use the definition of the combination it is processing to select the events for that combination which it will use to perform calculations on those events.

For CatRange purposes there are primarily two scheduling architectures that would work. One architecture would be a straight forward queuing scheduler, in which available workers poll the scheduling service for their next job. The other would be a scheduler that uses complex resource allocation algorithms to determine the best worker available based upon any number of definable variables. For example, the events of any given CatRange job may have a strong correlation with another job, therefore the scheduler could recognize that correlation, instruct the Worker Client to perform caching and then send similar jobs to that client, which would reduce network usage and greatly increase performance.

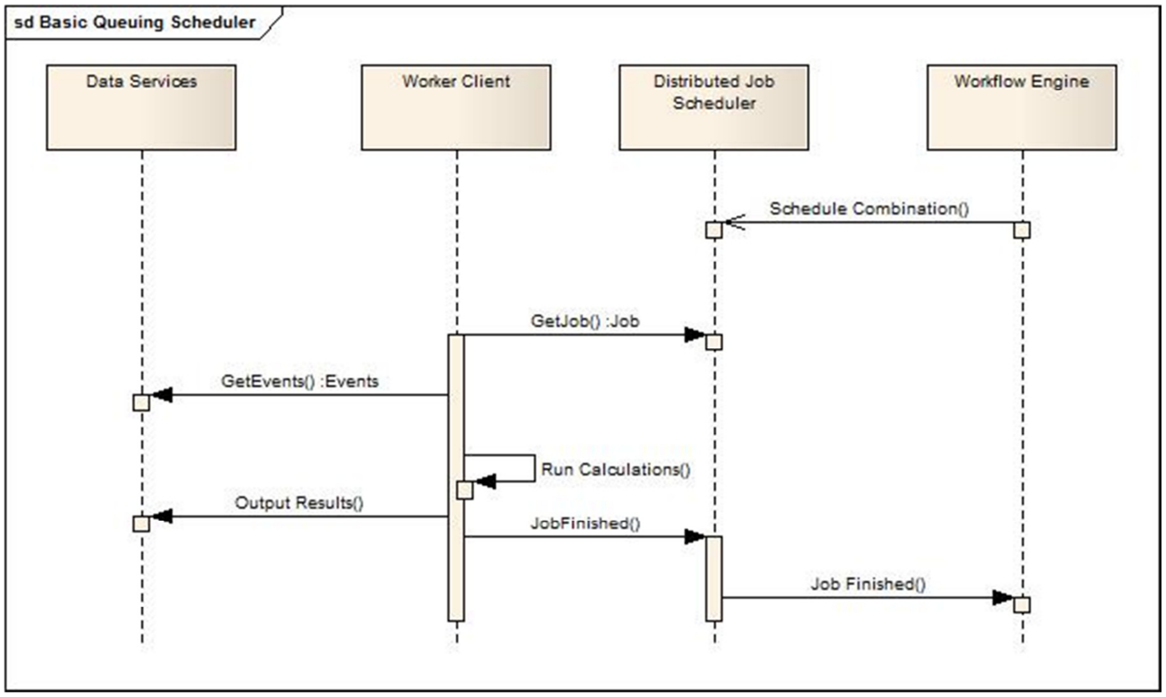
While it is feasible to develop a custom scheduler for CatRange, there are many existing fully featured enterprise grade solutions on the market. For the custom route a queuing scheduler would be the best option as resource allocation scheduling algorithms are very complex and can take millions of man hours to design and implement properly.

HIGH LEVEL ARCHITECTURE COMPONENTS AND IMPLEMENTATION OPTIONS

ORCHESTRATION TIER / DISTRIBUTED JOB SCHEDULER & WORKER CLIENT

IMPLEMENTATION OPTIONS

CUSTOM QUEUING BASED SCHEDULER



The architecture shown here would be sufficient, although not ideal, for the distributed portion of CatRange. The Workflow Engine will schedule a job by submitting the parameters the job would execute under. Currently a calculation combination is defined by where the data resides, what analysis is being processed, and a specific geo level, product, division, region and zone combination. The Distributed Job Scheduler would then add that job to the queue of jobs which are to be scheduled. When a Worker Client is not running a job it will poll the Distributed Job Scheduler for work, the Distributed Job Scheduler will then assign the client a job and ensure that job is not scheduled for another client.

The Distributed Job Scheduler would have to deal with prioritizations, it is recommended this functionality is very limited in scope as prioritization scheduling can be very difficult to implement. At the very least it will need to ensure that if multiple runs are in progressed the jobs for all of those runs are being worked on in such a way that there is not a large back log due to any one of the jobs. From the Worker Client's perspective any prioritization, or other resource allocation optimizations for that matter, will not be important as the client's only concern is processing the jobs it has fetched.

Under this model the scheduler would also not be aware of what clients are available nor any information about those clients that could be used for resource allocation prioritization, all clients would be treated equally. Once started these jobs would not be able to be stopped, since the average job run time will be around one minute this is not a major concern. Jobs would not be able to be partially executed and then restarted, all jobs must therefore be roll back-able.

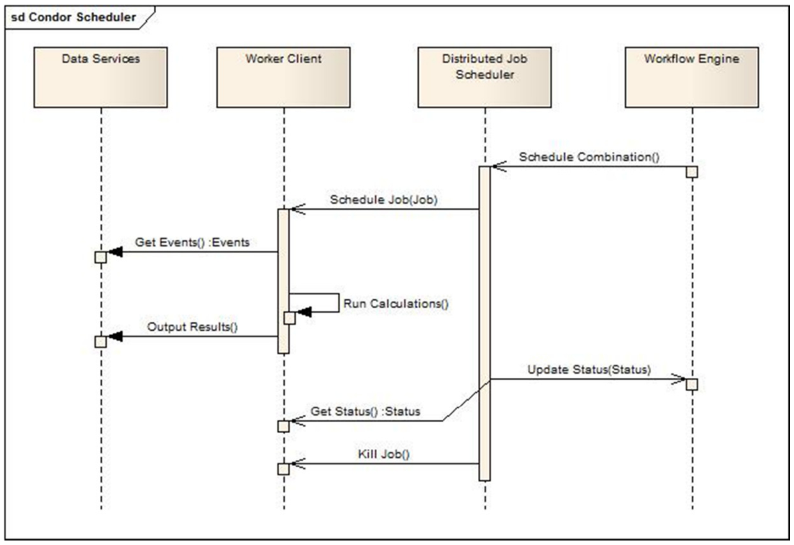
Another major concern is keeping the Worker Client's binaries up to date. Any updates to the orchestration component of the Worker Client, that is the component responsible for polling the scheduling service and executing the jobs it receives, will need to be updated through an update service mechanism similar to windows update. The component that executes the job, which communicates with the Data Services component and runs the calculations, would be an external binary that can be executed by running the process and passing it parameters. This executable component could be distributed through the same update service, however it would be better if the Distributed Job Scheduler could manage these as it is possible different jobs could be running different versions of the executable.

HIGH LEVEL ARCHITECTURE COMPONENTS AND IMPLEMENTATION OPTIONS

ORCHESTRATION TIER / DISTRIBUTED JOB SCHEDULER & WORKER CLIENT

IMPLEMENTATION OPTIONS

CONDOR HIGH THROUGHPUT COMPUTING SYSTEM



Condor is one of the most mature and widely used distributed computing schedulers on the market. Form the Condor website: "Condor is a specialized workload management system for compute-intensive jobs. Like other full-featured batch systems, Condor provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their serial or parallel jobs to Condor, Condor places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion." Condor is a free and open source project founded and run by the University of Wisconsin in 1986 as primarily a cycle scavenging product, designed to utilize the resources of desktop systems when not in used. Since then considerable effort has been spent on developing Condor not only into the premier general purpose cycle scavenging product on the market, but also as a major distributed computing scheduler used by many of the world's most powerful supercomputers, including IBM Blue Gene and NASA's Advanced Supercomputing facility.

Condor would not only provide a heavily used and well supported and featured scheduler for use in a dedicated cluster, but also possibly be able to leverage every desktop system on Aon's network when they are not in use. There is built in functionality for scheduling jobs on configurable prioritization and resource allocation policies as well as providing enterprise grade monitoring of jobs in progress and automated handling of failures of both execution logic and client hardware. For instance if a job was running on a desktop and that desktop became unavailable for any reason, Condor would be able to reschedule that job to another client. In fact Condor supports a concept of application check points where it would know the progress of the executable and be able to then restart the job from that point. A dedicated cluster for CatRange could still exist and jobs would be prioritized onto that cluster first and then schedule jobs to the desktop cluster, since the specialized cluster would likely have better overall performance than the desktop clients.

Condor's scheduler, monitoring services and clients can be implemented on a wide variety of operating systems, including Windows and Linux. The executables themselves can be developed in any technology supported by the clients.

The biggest difference between the Condor and the Queuing architecture described on page 10 is that the Worker Clients do not poll the Distributed Job Scheduler in Condor and instead the scheduler instructs the client. One of the big benefits of having the scheduler be able to contract the clients directly is better overall monitoring support.

One interesting option is the possibility that the client executables could be written in compiled MATLAB. CatRange 1.0 was actually developed entirely in MATLAB and ported to .NET in version 2, although .NET has proven to be a better solution in most areas of the application the calculation portion of the application was both faster in execution and less code in the MATLAB version. However parallel, let alone distributed, development in MATLAB is very difficult and since version 2 was able to leverage multithreaded parallel iteration of the various geo combinations 2.0 has achieved a significant reduction in run time when compared to 2.0.

While Condor is freely available, initial configuration and the development of the prioritization and resource allocation policies is not trivial, however once implemented overall system administration is not overly difficult. UW Madison offers priority support contracts starting at $2,323 a year. For more information: http://www.cs.wisc.edu/condor/

HIGH LEVEL ARCHITECTURE COMPONENTS AND IMPLEMENTATION OPTIONS

ORCHESTRATION TIER / DISTRIBUTED JOB SCHEDULER & WORKER CLIENT

IMPLEMENTATION RECOMMENDATION

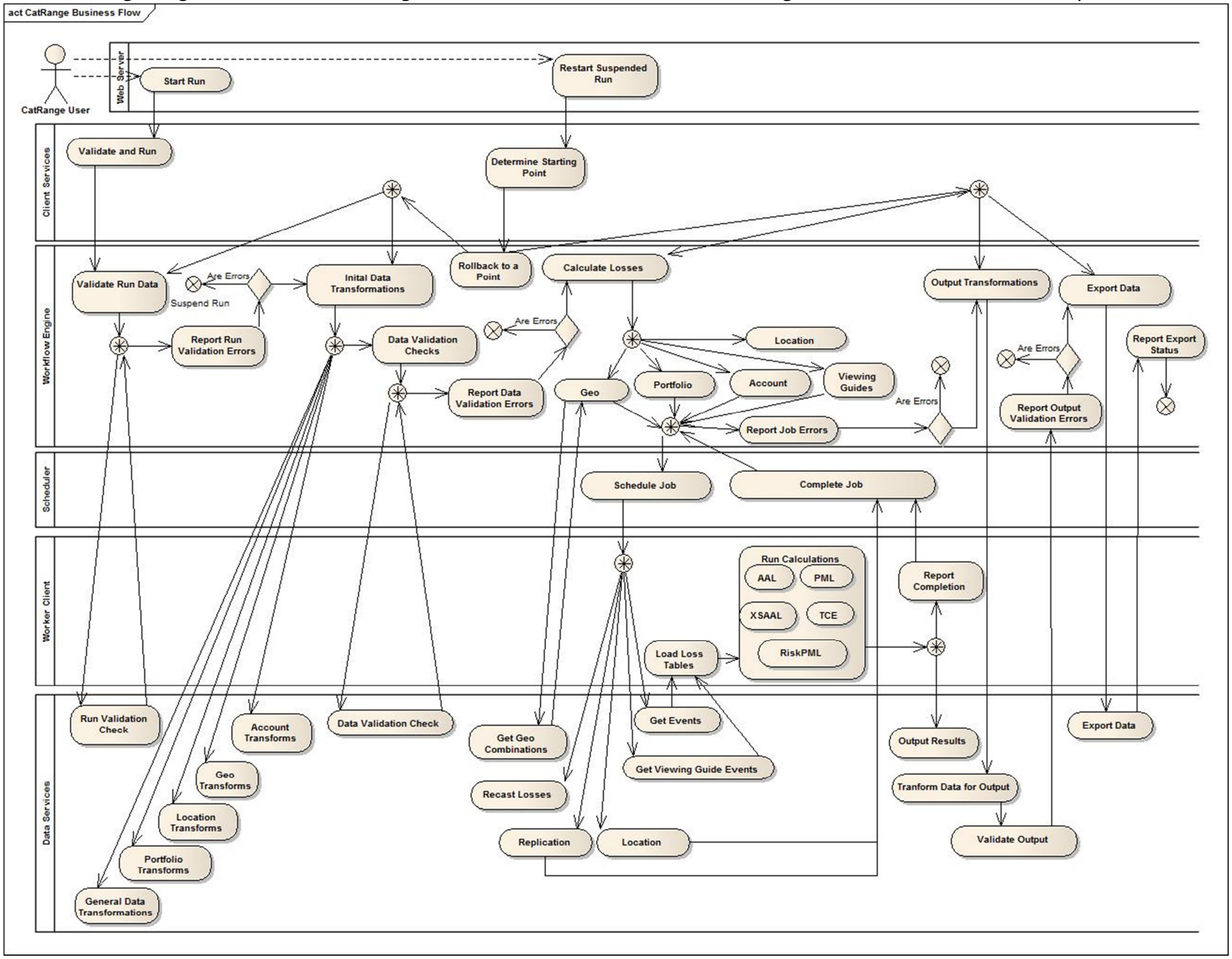
Both custom developing a Queuing Scheduler and using Condor are viable options, and while Condor certainly offers more out the box functionality than would be feasible for Aon to develop in a custom scheduler, a consultant would likely need to be brought in for at least the initial configuration as well as a support contract for the duration of CatRange's use of Condor. Schedulers are a very complex problem with serious implementations requiring a large cross section of areas of expertise including: information theory, statistics, computer science, networking and combinatorics. Schedulers like Condor have taken hundreds of millions of dollars and decades to refine and there is no feasible way for Aon to develop a custom scheduler with a feature set comparable to that of Condor or other off the shelf distributed computing schedulers. However a very scaled down scheduler using the queuing architecture, while not ideal, would be sufficient for CatRange's purposes and could be developed and supported without outside resources. We recommend that an expert in Condor to be brought in and help perform a cost benefit analysis of the options presented.

DATA TIER / DATABASE FARM

The exact design of the database farm is outside of the scope of this document as another CatView team is responsible for this component. From a CatRange perspective the main importance is that the Data Services layer should be able to query the farm without knowledge of the physical location of the data and that there be one point of entry to the databases needed.

HIGH LEVEL BUSINESS LOGIC OF CATRANGE

The following diagram details at a high level the workflow of a CatRange based on the current system and the proposed architecture.



CatRange Run Workflow

CatRange User → Start Run

The CatRange User will define a run configuration, through other activities on the Web Server which are not yet defined, which the Web Server will then use to launch the run.

Start Run → Validate and Run

The Web Server will call the Client Services component with the run configuration and wait for the configuration to be validated. This validation will only validate that the configuration is correct and not that the underlying data is. The data validations will happen asynchronously in other activities. If the configuration is valid the Client Services component will launch the run with the defined configuration asynchronously.

Validate and Run → Validate Run Data

The Client Services component will launch the run configuration on the Workflow Engine, the Workflow Engine will then validate that the underlying data is valid for analysis. This would include checks such as ensuring data is available at the levels and perspective selected and that constraints such as region, zone and line of business schemes are correct.

Validate Run Data ⟳ Run Validation Check

The Workflow Engine will call the required Data Service component's Run Validation Checks and capture the results for further processing.

Validate Run Data → Report Run Validation Errors

After processing the necessary Run Validation Checks the Workflow Engine will notify the CatRange User of the validation status of the run. If there are any validation failures the Workflow Engine will suspend the run until the CatRange user indicates it is ready proceed. The CatRange user can either choose to ignore the validation failures and continue with the run or make changes to the configuration or data and restart the run.

Report Validation Errors → Initial Data Transformations

After notifying the CatRange User of successful validation checks, or after being instructed to ignore the failed checks, the Workflow Engine will transform the data for user by later activities to both improve performance and simplify logic.

Initial Data Transformations ⟳ Generate Data / Portfolio / Location / Geo / Account Transformations

The Workflow Engine will instruct the Data Services component to perform the necessary General, Portfolio, Location, Geo and Account transforms based on the configuration. Depending on the size of the data these transformations currently take between several minutes and several days to occur. The performance will be heavily impacted by the number of concurrent runs and the concurrent query capacity of the Data Farm. This is currently referred to as the "Setup Portion".

Initial Data Transformations → Data Validation Checks

After the Workflow Engine completes the necessary transformations it will then run a series of Data Validation Checks to ensure that the data was transformed properly. In the current version of CatRange many runs pass the initial validation checks but still produce incorrect output, many of these scenarios could be avoided through straight forward validation checks of the data.

Data Validation Checks ⟳ Data Validation Check

The Workflow Engine will call the required Data Service component's Data Validation Checks and capture the results for further processing.

Data Validation Checks → Report Data Validation Errors

After processing the necessary Data Validation Checks the Workflow Engine will notify the CatRange User of the validation status of the run. If there are any validation failures the Workflow Engine will suspend the run until the CatRange user indicates it is ready proceed. The CatRange user can either choose to ignore the validation failures and continue with the run or make changes to the configuration or data and restart the run. It is also possible to make manual changes to the intermediate data so that the Initial Data Transforms would not have to be run, while this is something that should probably be avoided there are certainly scenarios that could justify this.

CATRANGE RUN WORKFLOW

REPORT DATA VALIDATION ERRORS → CALCULATE LOSSES

After notifying the CatRange User of successful validation checks, or after being instructed to ignore the failed checks, the Workflow Engine will begin the process of calculating losses for the configured run. Depending on the size of the data and other data factors, such as the number of geo combinations, calculating losses in the current version can take anywhere from minutes to many days to complete. The performance of the new version will be heavily impacted by the number of concurrent runs, the number of worker clients and their processing capacity, and the concurrent query capacity of the Data Farm.

CALCULATE LOSSES ↻ PORTFOLIO / ACCOUNT / GEO / LOCATION / VIEWING GUIDES

Based on the configuration, the Workflow Engine will calculate losses for Portfolio, Account, Location, Hurricane Viewing Guide and Geo level data.

PORTFOLIO CALCULATIONS → SCHEDULE JOB

For portfolio data there is only one combination of calculation that needs to be processed. Currently XSAAL calculations for Account and Geo are dependent on this job. Ideally in CatRange 3 this dependency will be removed, or at least separate it from the rest of the Account and Geo calculations. Another dependency on portfolio values is a performance optimization in the Geo calculations called country replication, the idea being if there is only one country in the data set (as is mostly the case) the values in the portfolio will be the same for the country level combination for all products, divisions, regions and zones. It is possible to remove this dependency by waiting to schedule the replication logic until after the Portfolio and Geo calculations have completed.

LOCATION CALCULATIONS → SCHEDULE JOB

The current implementation of the calculation logic requires a single query to handle the calculations for all locations since only an aggregatable metric is currently required. If non-aggregatable metrics are required in the future more than one job will need to be scheduled.

ACCOUNT CALCULATIONS → GET ACCOUNT → ACCOUNT CALCULATIONS ↻ SCHEDULE JOB

The Workflow Engine will contact the Data Services component for the specific account ids in the underlying data set. Account calculations can be broken down and scheduled by account, the event data for each account are completely independent and impossible to correlate without analyzing the data directly.

GEO CALCULATIONS ↻ GET GEO COMBINATIONS → GEO CALCULATIONS ↻ SCHEDULE JOB

This process is normally by far the most time and resource consuming portion of CatRange. The Workflow Engine will contact the Data Services component for the specific Geo Combinations in the underlying data set. Currently a Geo Combination is defined as a Geo Level (country, state, county, etc) and a combination of a Product, Division, Region and Zone with the possibility of any of these combinations being marked as 'ALL'. Since these combinations are related in various ways there are performance enhancements in place and the possibility for further enhancements to be made.

One current performance enhancement is the concept of replication as defined above in the Portfolio Calculations, in addition to country level replication data can be replicated at other various levels as well in the case that there is only say one state in a country. The replication jobs will be delayed until after the vanilla jobs have completed, so that the availability of necessary data is ensured.

VIEWING GUIDE CALCULATIONS → GET SEVERITY GATE COMBINATIONS → VIEWING GUIDE CALCULATIONS ↻ SCHEDULE JOB

Hurricane Viewing Guides define event sets according to a predefined combination of severities and gates and which is then used to run various calculations similar to those discussed on page 16. Specifically Hurricane Viewing Guides are used to project losses based on historical hurricane data. The severity and gate combinations are fetched from the Data Services component and the Viewing Guide Calculation jobs are scheduled for those combinations.

RECAST LOSSES CALCULATION → SCHEDULE JOB

Recast Losses calculate the losses for a single specific historical event and is performed entirely in SQL.

REPLICATION → SCHEDULE JOB

As described above under certain circumstances both aggregatable and non-aggregatable data can be replicated from previously calculated results. These jobs are performed entirely in SQL and scheduled after the results are available.

CatRange Run Workflow

Schedule Job ↻ Portfolio / Account / Location / Geo Calculation / Replication

The Scheduler will assign the various jobs to the Worker Client which will then execute the jobs. The calculation jobs are very CPU intensive whereas the transformation queries in the Location and Replication jobs are more database heavy. These database heavy jobs do not necessarily need to use the same execution mechanism as the calculation jobs.

Calculation Job → Get Events → Load Loss Tables → Run Calculations

A calculation job will fetch the specific event combination for the scheduled combination and load those events into a Loss Table which will then be used to run the calculations.

Viewing Guide Calculation Job → Get Viewing Guide Events → Load Loss Tables → Run Calculations

A viewing guide calculation job will fetch the specific event combination for the scheduled severity gate combination and load those events into a Loss Table that is used to run the specific calculations. These calculations are slightly different from those described in this document.

Run Calculations → AAL / XSAAL

AAL and XSAAL are aggregatable calculations used in projecting losses, AAL without any event thresholds and XSAAL with a defined threshold. These calculations can be written in SQL and may not need to be run on the worker client.

Run Calculations → PML / TCE / RiskPML

PML, TCE and RiskPML are non-aggregatable calculations which are difficult or impossible to implement in SQL. These calculations are very CPU and memory intensive and are prefect candidates for distributed computing. Probable Maximum Loss (PML) is a beta distribution calculation which describes the cumulative probability and cost of losses in an event set whereas RiskPML is the same for events in the portfolio but not part of the combination. Tail Conditional Expectation (TCE) is a calculation that measures the probability and cost of tail-events which are events that are not expected to happen. TCE uses the same event set as the PML calculation for a given combination. These events are only run at Account and Geo Levels

Run Calculations ↻ Output Results → Report Completion → Complete Job → Calculate Losses → Report Job Errors

After each calculation is run the results will be outputted in an intermediate format that is better suited for internal purposes than the target data model. While this is certainly true with the current architecture this may not be the case in the final architecture, however it will still be beneficial to separate in progress and completed results. Examples of this benefit include staging purposes as well as the ability to group previously executed runs together into the same logical analyses for purposes of display in CatView.

Upon completion of the result outputs the Worker Client will inform the scheduler that the Job is complete which will in turn inform the Workflow Engine. The Workflow Engine will then notify the CatRange user of the status of the job, if there are any exceptions or validation errors the run will be suspended, otherwise the run will proceed to the output transformations. In the event of an error the CatRange user will be able to modify the configuration, underlying data or instruct the development team to modify logic and then optionally be able to restart all or any part of the run.

Report Job Errors→Output Transformations→Transform Data→Validate Output→Report Output Validation Errors

After the intermediate data has been successfully validated that data will be transformed into a format defined by the CatView team. Currently this portion of the logic is called CPDM population. The output will then be validated to ensure all data constraints are properly met and any violations will be reported to the CatRange user by the Workflow Engine. In the event of invalid output the CatRange user will be able to modify the configuration, underlying data or instruct the development team to modify logic and then optionally be able to restart all or any part of the run. After the successful completion of this process storage of the intermediate data is no longer necessary, although would be extremely valuable if additional calculations were necessary on the same analysis in the future.

Report Output Validation Errors →Export Data → Export → Report Export Status ⊘

After the intermediate data has been successfully transformed for output the data will be exported to CatView. Currently this process involves exporting the CPDM data to flat files and then transferred using SFTP via CatInHat. Ideally the new version will allow for better automation through a mechanism such as SQL Server Integration Services. CatView will verify the validated of the data and the CatRange user will be notified of any validation errors preventing transfer.