Paper 224-29

Architectural Views of Building an Actuarial System using SAS/Warehouse Administrator®

Dominic Roy, DMR Conseil / Fujitsu Consulting, Sainte-Foy, Quebec, Canada
Alain Baillargeon, DMR Conseil / Fujitsu Consulting, Sainte-Foy, Quebec, Canada

ABSTRACT

SAS/Warehouse Administrator is an advanced code generation tool used in building data warehouses. In our project, the tool was used in order to support and document the development of a large actuarial system. This paper discusses the architecture of the system, the ways and means to solve the problems and the traps and pitfalls of such an approach.

INTRODUCTION

Roy & Genois (2004) explains the business context of an actuarial liability valuation system built at the Société de l'assurance automobile du Québec. The following presentation deals with architectural aspects of the project. However a short introduction to the context is necessary to understand the architecture.

The "Société de l'assurance automobile du Québec" (SAAQ) is responsible for the Public Automobile Insurance Plan in the province of Quebec. This plan is a no-fault insurance regime that protects all victims of motor vehicle accidents. It is a mandatory plan for all drivers and all vehicles. It covers Quebec citizens circulating on provincial roads within and out of the province. It also protects non-Quebec citizens while using provincial roads. The plan is financed with a part of the driver's license and auto registration fees.

The plan's actuaries are responsible for accurately predicting the amount of money that must be set aside (the liability) to cover the future needs of all new accident victims during the current year. They must also check whether the money set aside in previous years is sufficient to cover victims from those years. The liabilities are calculated and justified separately for each category of expenses.

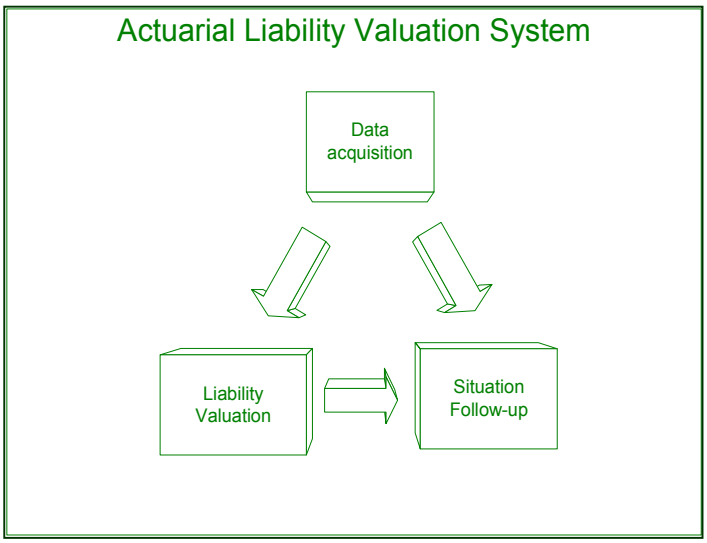
A new actuarial liability valuation system was necessary due to certain technological problems, but mainly because of the growing complexity of the valuation. At the same time, the upper management wanted a more secure system to sustain a corporate part of the mission.

GLOBAL ARCHITECTURE

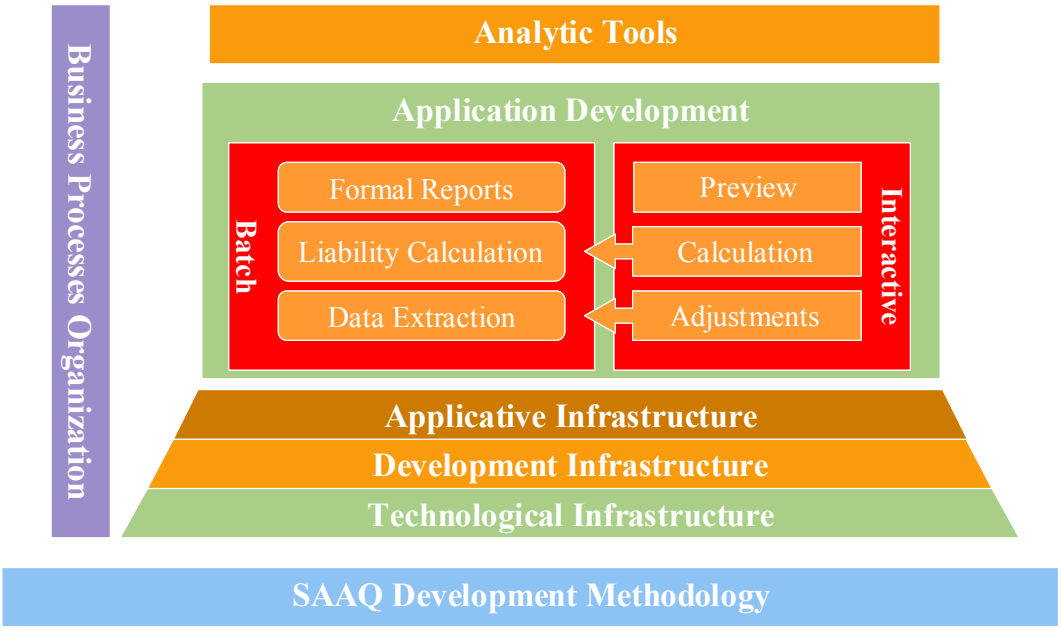
An actuarial liability valuation system is composed of three major processes: data acquisition, liability valuation and situation follow-up.

The data acquisition process is done by extracting the data, verifying the quality and making corrective actions on the data. The major proportion of the process can be taken over by SAS/Warehouse Administrator as a regular data warehouse activity.

The liability valuation process is the most complex segment and the less conventional business intelligence activity. Each expense category has its own evolution behavior, so each category must have a specific predictive model based on a more general predictive models. The actual system deals with 3 income replacement rents evaluated on an individual basis and more than fifty expense categories evaluated on a global basis. Those valuations must take care of interdependence between categories, taxes, economic hypothesis and reimbursements for other organizations.



The situation follow-up process is a more conventional business intelligence part in which users watch the current economic and cash flow situation. Reports can be scheduled with SAS/Warehouse Administrator, as well as multidimensional data preparation.



THE METHODOLOGY AND THE BUSINESS PROCESSES ORGANIZATION

Because the proposed system's predecessors were considered corporate level systems, it was essential that the new application would be a corporate level system as well. This meant that the highest standards of the SAAQ system development methodology had to be met, with a defined process, structured documentation and a clear definition of responsibilities.

This methodology requirement alone solved the major part of the documentation needs, the project steps and the communication between users and developers. It also brought many best practices to solve common problems. Moreover, it gave a common ground for the documentation in such way that at the same time the users, the developers and other IT staff members could understand what was being done in the system.

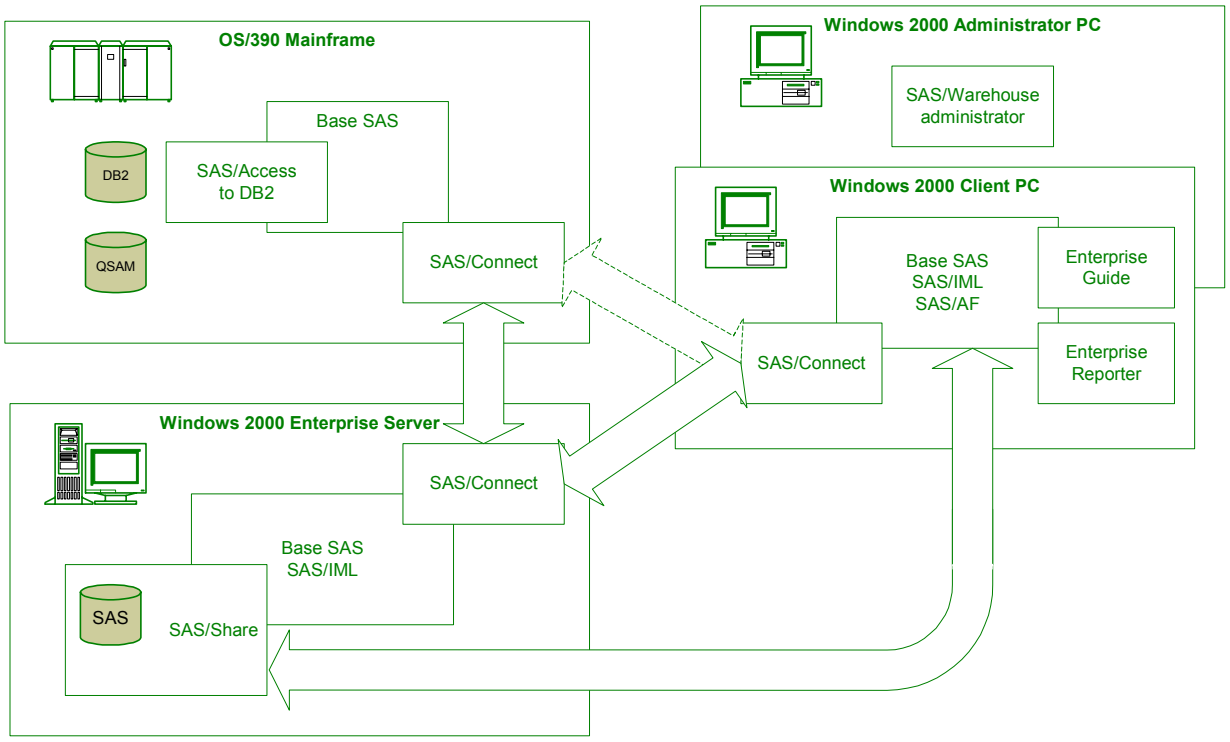
Managing the responsibilities of the members of the team of actuaries also solved some requirements. Incidentally, the security requirements were managed by defining a system administrator who had all the rights on the containers but very limited rights on the contents. However, software licensing issues and data integrity concerns forced us to decide that this administrator (and his replacement) was the only one could modify the metadata and run the jobs. Such a situation, even tough not ideal, was still manageable.

SAS/Warehouse Administrator imposed adaptation to the SAAQ methodology because it took in charge the data documentation, the job structure and the process structure. We overloaded the process structure to include non-computerized steps; this way, the process viewer became the administrator tasks checklist.

THREE INFRASTRUCTURE LAYERS

THE TECHNOLOGICAL INFRASTRUCTURE LAYER

In order to centralize both the data and the computing power needed to process it, a dedicated Windows 2000 server was acquired. Using SAS/Connect®, client PCs could initiate calculations on the server and avoid using local computing power. Two principles concurred at this moment: 1-centralize the data to facilitate the security issues and eliminate the risk of data duplication, and 2-use the processing power as close as possible to the data.



SAS/Connect was optimized and secured on each platform to give all the performance available. Even if there was a direct bridge between the client PCs and the mainframe, all the formal processes requiring an access from the PC to the mainframe had to cross the SAS/Connect server-mainframe bridge. This bridge had been optimized with the TCP/IP spawners and gave the maximum throughput. The biggest part of the data was extracted from mainframe flat files and finally stored on a Windows platform. It was proven that the most efficient way was to convert the data into compressed SAS tables on the mainframe and download them into compressed SAS tables on the Windows server. The compression rate was around 90%, which is typical with very long records filled with sparse data. The data compression had no significant negative impact on the computing performance afterwards.

SAS/Share® was installed on the server to reduce data contention among users. In addition, it served to imitate a true relational database manager in which the users didn't have to consider the physical locations of the tables. Three major objectives were targeted: 1) force the user to access the data through the SAS/Share server, 2) manage the access to data through the Admin Procedure, and 3) allow the administrators to change the physical location of the data without having to change any of the programs that access it.

SAS/Share was almost 100% successful in performing these tasks. On a Windows® 2000 system, we found that it was impossible to completely mask the physical address of data on the server. At the same time, user's access rights to a directory were defined primarily by the Windows security policies; SAS/Share could not grant rights greater than Windows would allow. This meant that a user could allocate a library to himself, overriding whatever rights were maintained by the SAS/Share server and denying other users access. That situation forced us to grant read access to production data to all users and write access to administrators only. To overcome this restriction, a new data lending utility application (with a journal) was developed that temporarily grants users the access needed to make corrections to the primary data and the hypotheses data sets.

SAS/Share was installed using the SAS Service Configuration Utility. As many as thirty services were installed. This way, each SAS/Share service owned all the libraries under a specific directory root. Generally speaking, all the services had the same set of libraries. The programs that would use these libraries would allocate them with the SLIBREF option like in the next statement.

```
libname saspgm slibref=saspgm server=&servname;
```

This indirect allocation process supports the major part of the development infrastructure, as well as the multiple periodic environments. The common working environment is composed of 34 libnames, many of them being level-2 libnames (libnames defined as the concatenation of two other libnames).

THE DEVELOPMENT INFRASTRUCTURE

SAS/AF stores its components into catalogs, while SAS/Warehouse Administrator looks for source programs into catalogs. Also in search of a source control management tool, we discovered that SAS/AF SCM tool was the most

interesting. It is capable of working within catalogs and it includes table management; but it is less appropriate for managing external files such as .SAS sequential files. We therefore decided to manage all the components within catalogs structures. Approximately 1500 components (entries) are managed within 275 catalogs. 266 components are classes, compiled IML modules, images, etc; 85 components are SCL code (15,000 lines with comments); and approximately 1150 components are base SAS, macro and SAS/IML code (200 000 lines with comments).

Five environments levels were necessary to support the test strategy. At the programmer's phase there was a private environment (level 0) and a shared environment (level 1). Compilers had been developed to take care of formats, compiled macros and SAS/IML modules. SAS/Warehouse Administrator metadata was maintained and tested only at the integrated level (level 2) to limit the number of metadata versions and the number of libraries to maintain per level. There was an acceptance level (level 3) and a production level (level 4). Most promoting activities were done using SCM.

Maintaining SAS/Warehouse Administrator metadata for each 3 upper levels (2, 3 and 4) was not easy because there was no intrinsic mechanism to promote a single metadata element; it's quite frequent on the market. However, when we delivered a new version of the system, all the metadata was promoted to the production level as a whole; before doing that, we checked that all minor changes done in upper levels were done in the lower levels (retrofit). The metadata is easy to copy from one environment to the other, provided some conditions: no physical information can be registered within (physical library names, for example); new metadata object must be created at the lowest level (level 2) to prevent the use of two distinct objects with the same unique identifier, etc. We made a program to compare metadata between levels, which was useful to ensure that levels were identical. We gave up to build a tool to identify the specific differences between two metadata versions; it was too complex but more importantly it was an approach comparing tables instead of metadata objects.

THE APPLICATION INFRASTRUCTURE

While operating the system, SAS/Warehouse Administrator was not used only as a simple menu feature for starting the jobs. Rather, it encapsulates each job using prolog and epilog. The prolog prepares the environment, sets the objectives of the results, and checks the up-stream task dependencies. The automated code generator rebuilds the resulting data table allowing keeping a complete control on the data structure. The epilog is used to check the success of the job and the validity of its results versus the objectives, to make a control report, to write a log and to generate the audit trail report. We also used it to scan the SAS log to interpret the potential uncontrolled problems.

To support those prolog and epilog features, two add-in tools have been developed to extract metadata into specific SAS data tables, because normal SAS code don't have access to metadata. One add-in tool extracted the job information. It allowed the prolog to check which jobs are the precursors of the current job. The other add-in extracted the data information into a SAS data table to help the prolog check the presence of the data tables that are in the input of the current job. This extracted information was also the core of the audit trail report on input data.

We also used these tables to create a new HTML job report, on the model of to the one provided with SAS/Warehouse administrator that generates a HTML data report. While this one reports the structure of the data table and the job that generates it, ours reports the structure of the current job and the structure of the data table going out of the current job. This add-in tool provides a report that is attached to the requirement document written with Microsoft® Word®.

Other tools were developed to maintain the multiple simulation and periodic environments; they are base SAS program run outside of SAS/Warehouse Administrator. These administrative tools have to create, copy, move and archive the environments. While it is relatively easy to copy the directories, many dependencies are to consider. For example, SAS/Share services must be initiated or shut down. A set of automated administration tables helped to simplify this process. Autoexec programs became complex to support these movements. Many functions have to be supported while creating new simulation and periodic environments: some of them are created virgin, some are partially emptied and others are almost pure copies.

THE APPLICATION DEVELOPMENT PLATFORM

A COMMON BASE FOR MACRO COMPONENTS

Two very different platforms were used to support the applications: the batch platform under the regime of SAS/Warehouse Administrator and the interactive platform that is the kingdom of SAS/AF. They share data tables and many common components together. Those common components are base SAS and SAS/IML code entries that are most often encapsulated into parameterized macroprograms. They are also built as complex chains of callers and callees. This way, the same programs can be called from the interactive platform (as a SAS/Connect remote submit) or with SAS/Warehouse Administrator encapsulated into a job (also a SAS/Connect remote submit).

We chose SAS/IML® to replace our spreadsheet calculations. This simple language includes many of the functions previously used in our spreadsheets. Because it is less redundant than spreadsheet code and does not need to be updated when matrices change their dimensions, its use rather reduced the complexity of our calculations. We further simplified our logic by using parameter-driven custom code to load data into matrices and to insert matrix

data into data sets. All newly written calculations deal only with matrices, allowing the development team to partition the work required by a complex calculation.

THE BATCH PLATFORM

The batch platform is the most controlled one where absolutely all the official results are obtained. The batch platform used, at its core, SAS/Warehouse Administrator. These programs are documented and run under the process viewer. All the programs had the responsibility to deliver their messages into a standard table and their control information into another table.

Because we needed periodic and simulation environments, we conceived that each of them would be an independent data warehouse (in WA-speak). It derived into two types of data warehouses: the extraction warehouse and the actuarial liability calculation warehouse. The extraction warehouse was mainly a batch process and could be run on a scheduled basis. Very little normalization was done in the structure of the SAS tables. Normalization could have been beneficial for storage management and SAS/Warehouse Administrator performance. As actuaries used to work with very long vectors on which they loop, it was therefore better to maintain tables with repeating data elements. SAS/Warehouse Administrator suffered from this choice because some tables climbed to a summit of 500 columns. When it deals with so many columns the response time is longer. It was partially corrected by indexing the metadata tables. This situation may ask for the most powerful personal computer to install the software.

The normal data warehouse operations are mapping a data from a source table to a destination table. SAS/Warehouse Administrator has the tool to do that efficiently. However, an actuarial system needs more complex processes that that. This is why we almost exclusively used the user written 'mapping' programs that were actuarial calculations. We used the normal 'load step' code generation in the tool to create and load the data into new tables. We had to adjust the way to stop a program before the load step when a so-called mapping step hit an exception.

Two options were used to generate the reports. SAS Enterprise reporter® was used to generate sleek and stable reports, while ODS was used for common or highly unstable reports (unknown number of columns, for example). Enterprise reporter was not easy to manage in a system with multiple-level environments, while it added new components to promote. We also gave up to put this computation on the server and finally we came to the conclusion that it could be run only on the administrator PC. ODS for Acrobat® PDF® was used extensively on the server with no problem at all; but we customized templates very little. All the results are stored permanently in their own periodic or simulation environment.

Each set of monthly extracted data is stored in a different a warehouse, and becomes accessible through a unique SAS/Share service name. The system administrator only needs to keep the monthly data that is of use to the actuaries, normally just the current month and the last month of the previous year. The other months are archived and restored on demand. Each 'monthly' environment contains its own data structure, parameters, programs and metadata.

Liability calculation exercises are performed regularly. At the end of each year, a statutory valuation is made to serve as the basis for the plan's annual report. During the year, intermediate valuation exercises are done to monitor the plan's performance.

This liability calculation warehouse also has its own data structure, parameters, programs and metadata. Unlike the extraction warehouse though, this one can have many instances, each of which supports a particular progression environment or a simulation environment. Each instance also requires its own set of data, parameters, programs and metadata. These progression environments are incremental in their changes and are ordered. The first one is qualified as the plain exercise environment because it is similar to the reference environment – the last environment of the previous exercise. The final exercise environment of the current year is the one that integrates all the changes and improvements to the valuation methods.

Each liability calculation warehouse has a life of its own. A part of the calculations must be run using the process editor for each version of metadata inside SAS/Warehouse Administrator. Each environment keeps track of the status of its jobs and maintains its own set of results.

During a valuation exercise, all the progression environments for the current period must be compared. This part is the most complex program with the dynamic allocation of data libraries and even of program libraries. It becomes impossible to verify the validity of the results, make minute changes in the parameters and the programs, and always have the confidence that the results are on a solid ground.
To do it, the answer was quite simple but rather complex to implement: 1-separate the valuation process into manageable chunks that we will call the calculation, 2-run the same calculation in two different context, and 3-compare the results. Doing that actuaries can even create any number of calculations and choose which they want to pliability. This is where the actuaries can 'play' with the data, developing their own models in an effort to improve forecast costs and more accurately calculate the required liability. This part of the system was built on the interactive platform.

THE INTERACTIVE PLATFORM

SAS/AF® was used to develop our graphical user interfaces. Version 8 System Component Language (SCL) was used exclusively, with its object orientation, dot notation and classes. SAS/AF proved to be both stable and powerful but required a great deal of training and subsequent experimentation. It's object-oriented paradigm proved a challenge for our more conventionally trained SAS programmers; Java® or Visual Basic® programmers might have come up to speed more quickly. While the newer classes (the Table Viewer, Form Viewer and SAS Data Set Model) are not well documented, we were able to develop several needed composite classes with them. We also found the object-oriented approach used by the Unified Modeling Language (UML) to be both powerful and stable. Roy and Milliard (2004) explains the work done to articulate UML© standard with SAS/AF.

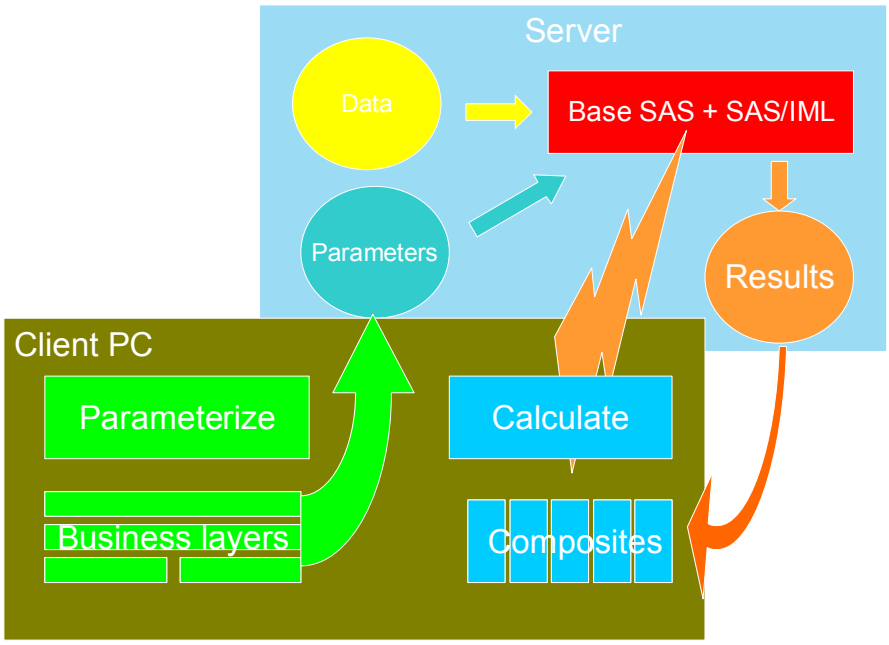
The major SAS/AF applications were built on a concept separating the user interface from the business rules layers. Base (non widget) classes were used to support the business rules layers. The principal Frame application is responsible for all interactions with the user. All business logic, including the interactions with SAS data sets are found in the business rules layer. In the INIT section of Frame, all the classes required by this processing are instantiated. In the remainder of the application, graphical controls send messages to the business classes, which maintain the relationships between the instantiated objects. All data sets are updated automatically and are temporarily stored on the client PCs both to support undo operations and the refreshing of our forms.

A composite widget approach was also used in SAS/AF to support a viewer concept. The table viewer widget was used to expose all the data, the parameters, the intermediate results and the final results in a calculation. It was demonstrated that all the possible views of these tables were hierarchical alterations on a common ground (reminding the class inheritance concept). All those variants were coded as separated composite widgets that are dynamically inserted into a navigator screen. This approach paid off when we decided to implement the protection masks in which some cells were allowed to be modified and the other not in different region patterns.

Contention between users and between graphical and business components is a major concern during the opening and updating of the data sets. Running SQL with "'UNDO POLICY='NONE'" allowed the needed flexibility for these components. User contention was managed with SAS/SHARE services' row level control and quick SCL updates. Including the business rules layer, the completed system grew to 15,000 lines and required the talents of many skilled programmers. The SAS/AF Source Control Manager (SCM) allowed us to keep track of the forty components (SCL, CLASS and FRAME) in the application. Business classes were easily developed and tested outside the graphical application and each class had its own test component. In this way a single programmer could develop very complex logics outside the graphical interface. When ready, another programmer responsible for the user interface was able to quickly integrate that code into the application.

This business rules layer model proved to be powerful enough to implement complex and CPU intensive logic on a remote computer. While we built our user interface under Windows using SAS/AF, another possibility is a Java interface that would be accessible from the Web. Multi-threading and multitasking could benefit this application as well.

The actuaries use two main applications within the interactive platform: the justified data modification application and the interactive liability calculation application. The justified data modification application is a very secured screen where only the actuary in charge can bring change to the data. For every cell that is changed, a comment explaining the justification must be entered. This comment can be reused for many cell changes. Behind the scene, an audit trail runs. It captures every single move, recording the change itself, when it took place and by whom. Repeated changes are recorded independently. This information will be processed in the audit trail report made by the epilog.



The interactive liability calculation application is made of two main screens (Parameterize and Calculate) combined with business layers and composite widget classes. The first screen manages the parameters and the calculations they control. It allows the user to import calculations from a previous exercise, duplicate calculations from the current exercise or delete calculations altogether. It also works as a workflow monitor to sustain the incremental nature of the calculations. Finally, it allows the user to change the values of the parameters used in the calculations to be run.

The second screen initiates the calculation and offers the user a detailed view of the results. Generally, thirty different views are available for each calculation to sustain the validation of intermediate calculations. In addition, some corrections to the data can be made from this screen, with the same approach of the justified modification application. This screen allows printing the PDF report associated with the calculation, and this PDF can be viewed with the default browser.

Only parts of the liability calculations are done using this application; those liability calculations are called 'global liabilities'. All these calculations are done with SAS/IML programs. The SAS/AF interface is used to capture the parameters, create new calculations and load these calculations on the server using SAS/Connect. Once the actuaries have finished and approved these calculations, they are run another time on the server using SAS/Warehouse Administrator. This bulk processing guarantees that all the calculations are made in the correct order and in a coherent environment. This bulk processing can also be used to make global changes in both the data and the parameters and allows the actuaries to observe the impact of these changes on the results produced.

THE ANALYTIC TOOLS PLATFORM

The data extraction phase and the liability valuation phase generate a great amount of data. With the multiple data warehouses approach, each warehouse sits on a different SAS/Share service ensuring that every single data warehouse is integral, complete, coherent and well documented.

The analytic tools platform is made of two approaches complemented by SAS/Warehouse Administrator. The first approach is to automate the access to coherent warehouse using base SAS Display Manager System (DMS). This is done by the main menu application that lists all the available warehouses in a structured manner and, when a choice is made, it allocates all the libraries and the resources necessary to play on a safe ground. This DMS SAS session is very well tooled with SAS/EIS®, SAS/STAT®, SAS/ETS®, SAS/OR®, SAS/Insight® and SAS/Assist®.

The second approach is with Enterprise Guide®. This software product was installed and configured on the actuaries' PCs, allowing them to access and analyze both their primary data as well as their results. This approach was a mixed success. We installed the product as an entry-level application and it was very well accepted as such. However, we installed it using the local server option (instead of IOM). As this implementation requires that an administrator prepares the EG binders and shares it with the users, it was decided to let the time flow until the next version. What was expected is a stronger integration of the metadata from SAS/Warehouse Administrator to Enterprise Guide.

At the time of writing these lines, the teams had very little time to implement a more complete business intelligence platform using summary tables and cubes within the data in SAS/Warehouse Administrator. This is one of the major avenues in future development tracks on the system.

CONCLUSION

SAS/Warehouse Administrator incorporates a specific data warehouse paradigm that must be adopted by its users. Getting up to speed with this tool can be a long and sometimes tedious process, but we found that it pays off in the long run. In our opinion, this software has the flexibility to support development of any type of large-scale SAS system. Our intention was not to push this product to its limits; we knew the power of frameworks and the Metadata API. We developed a custom add-in to link our specifications document to the job icon in the process view. We also developed an HTML report for documenting the job instead of the data. In addition, we developed an extractor to produce two tables: a job dependencies table and a data dependencies table.

SAS/Warehouse Administrator can be adapted to support a multiple simulation environment with relatively few modifications. It allowed us to document a complex system and helped us to keep that documentation up-to-date. The software has both greatly simplified our system development processes and made our documentation available to anyone who needs access to this information.

This project was a success because it utilized the right mix of technologies and included personnel familiar with these tools as well as with the client's need. Good communication between the clients, the system architects and its implementers provided a sound foundation. Object-oriented modeling techniques provided a workable vision. System development standards encouraged common expectations and ensured on-time deliverables. Prototyping proved essential to creating and revising the system during its development.

REFERENCES

CROSS-REFERENCES

Roy, D. and Genois, J. (2004), "Business Views of Building an Actuarial System using SAS/Warehouse Administrator®," *Proceedings of the Twenty-Ninth Annual SAS Users Group International Conference, 113-29*

Roy, D. and Milliard, A. (2004), "Modeling Object-Oriented SAS/AF® Applications Using UML®," *Proceedings of the Twenty-Ninth Annual SAS Users Group International Conference, 161-29*

WEB SITES

People interested in insurance and particularly in automobile insurance in Canada can consult the following sites:

The Canadian Institute for Actuaries: http://www.actuaries.ca
Société de l'assurance automobile du Québec (SAAQ): http://www.saaq.gouv.qc.ca/

ACKNOWLEDGMENTS

We want to thank our employer, the Quebec City office of DMR Conseil, a subsidiary of Fujitsu Consulting for their support, and particularly to Mr. Pierre Hamel who encouraged us to go ahead with this paper.

We thank the SAAQ (Société de l'assurance automobile du Québec) for their confidence and their encouragement, especially Mrs. Linda BellWare, Mr. Robert Ferland and Mr. Denis Doyon.

The authors would like to express their appreciation to the following people for their assistance in this paper:
Susan Assad, DMR Conseil
Josée Genois, Société de l'assurance-automobile du Québec

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Dominic Roy
DMR Conseil / Fujitsu Consulting
2960 boulevard Laurier
Sainte-Foy, QUEBEC
CANADA, G1V 4S1
Work Phone: 418-653-6881
Fax: 418-653-4428
Email: dominic.roy@consulting.fujitsu.com
Web: http://globalservices.fujitsu.com/services/

Alain Baillargeon
DMR Conseil / Fujitsu Consulting
2960 boulevard Laurier
Sainte-Foy, QUEBEC
CANADA, G1V 4S1
Work Phone: 418-653-6881
Fax: 418-653-4428
Email: Alain.Baillargeon@consulting.fujitsu.com
Web: http://globalservices.fujitsu.com/services/