Paper 178-28

A Successful Implementation of a Complicated Web-based Application Through webAF™ and SAS® Integration Technologies

Clare Nicklin, Amadeus Software Ltd., Oxfordshire, UK
Daniel Morris, Amadeus Software Ltd., Oxfordshire, UK

ABSTRACT

Amadeus Software Ltd. in the United Kingdom has been involved in analysing, designing, and implementing a complicated, web-based, tax modelling system. The system was built to be a browser based application, available internally through a company intranet. This paper, initially, details the applications software requirements where a combination of SAS® software (including webAF from AppDev Studio™, Integration Technologies, SAS/AF® software, and Base SAS Software Release 8.2) and additional software (such as Apache, Tomcat, and JDK) was used collectively. Secondly, the systems architecture is summarised to describe how the specified software and hardware components are organised. Thirdly, the applications design features and defining characteristics are explained. Finally, a number of difficulties that emerged during the process of creating the application using the specified software are described. This paper is intended to be of use to both developers and managers to aid in the conception, construction, and problem management of complicated, web-based applications which interact with The SAS System.

INTRODUCTION

The web-based tax modelling system explained in this paper allows a series of tax calculations to be defined, stored and run through a set of tax modeller calculation code. The system was initially built for a single tax model. However it was necessary to future proof it so that any number of different models could be integrated. In this way the design is such that much of the application is built using metadata and specified directory structures. It is therefore capable of handling modification and fine-tuning by the system administrators themselves.

APPLICATION IMPLEMENTATION

Software Used and Systems Architecture

The application was built to be housed on a Unix server and accessed from standard Windows desktop machines using Microsoft Internet Explorer version 5 and above. The browser based application is available internally to the company through its intranet. Both the web server and the application server are installed on the same physical Unix machine. Figure 1, shown in Appendix A of this paper, depicts the applications systems architecture. Its main features are:

- The **Web Server** hosts all the web-based files, including any HTML files, JavaServer Pages (JSPs), other Java classes, and JAR files necessary for the running of the application. These files are published to the Apache[1] web server, its Tomcat[2] servlet engine, or the JDK[3]. While Apache hosts the application, Tomcat is necessary to interpret the JSPs and provide full support for the necessary Servlet 2.2/JSP 1.1 standards.
- **JDK** is the short-cut name for the set of Java development tools needed to build and compile Java applications. The JDK environment includes the necessary class libraries and other files that support the execution of programs written in Java. Additional class libraries in the form of JAR files from AppDev Studio were placed here.
- The **SAS Server** consists of Version 8.2 with Integration Technologies. All data is stored and manipulated on this server.
- The **IOM Spawner (Integrated Object Model Spawner)** is part of SAS Integration Technologies. External applications communicate with SAS software through this object spawner, which listens constantly for calls to SAS on a specific port. Any conversation between the application (via the web server) and the SAS software is done through the spawner.
- **Base SAS** Software Release 8.2 is used for all data storage, manipulation and retrieval.
- The application is accessed from a browser, the default for this application being **Internet Explorer** version 5 or above.
- AppDev Studio's **webAF** was used for the applications development. The application comprises of many JavaServer Pages (JSPs). TransformationBeans were used in both Java scriptlets and XML tags to link to webAFs existing models and represent any results on web pages in HTML. InformationBeans were also used for providing the application with access to several SCL classes. Additional Java classes were also written to perform particular tasks throughout the application.
- **SAS/AF** was also used for the applications development to create various SCL classes. Parameters are passed from the Java into the SCL, where various tasks are performed. Information in the form of strings and lists are then sent back to the application.

Application Characteristics

Metadata Driven

The system has been designed to easily handle future modifications and changes in policy by using metadata, stored in SAS data sets, and specific directory structures to generate its screens. There are different levels of metadata:

- Environment level - the highest level of metadata allows the environment itself to be defined including all the system paths to the top level directory structures and user logon information.
- Application level – this metadata allows an administrator to define information to dynamically produce the applications screens. Metadata is generated from the stored data sets through SCL classes. It is then passed to the JSPs, where the Java builds the HTML for the screens content and appearance. Such screen elements built this way include:
    - Names, labels, groupings, ordering, valid value ranges, and screen positions for all parameters. This applications main function is the editing of large groups of parameters and their characteristics. This makes the parameter metadata the most complicated and comprehensive in the system. The metadata must be extensible to new sets of parameters and also to entirely new tax models.
    - Radio buttons for the selection of subsets. A radio button is generated for every subset that is defined in metadata.
    - Images and styles being used in the GUI. Company logos and colour schemes may change. Images and stylesheets can be amended without the need to re-develop entire screens.
    - Any displayed text such as screen labels. This means that language can be edited at any time during or after development.
    - Drop down menus and navigational button contents and destinations.
- User level - this holds various information such as user preferences and user defined formats.

Metadata Managers

Provision had to be made for non SAS programmers to edit the various datasets involved in defining the applications metadata[4]. Screens were therefore built to allow the systems administrators to make changes (ordinary users do not have access to such screens). The managers consist of:

- Parameter Metadata Manager – creating new parameters and new parameter sets, editing existing parameters, and moving parameters between different groupings are necessary tasks for this application. This metadata defines how the parameters are displayed on the screen within the application that allows for the editing of the parameters actual values.
- Subset Metadata Manager – subsets can be used on data within the tax modelling code. These are set up by a SAS knowledgeable administrator who defines a meaningful label for a subset and then includes the actual SAS code to be run. This metadata defines the number of subsets available in the system and allows users who are not SAS conversant to simply select a subset description.
- Stylesheet Manager - the look and feel of the application is based around the clients web site using specified company standards and images. Due to the future proof nature of the application, the design has to be flexible for when standards change. Stylesheets are used heavily to define such standards and to provide a single access point for editing. The application comes equipped with a stylesheet editor to assist changes. The stylesheet to be used is stored in metadata.
- Image Manager - it is also possible that images and logos change in the future. An image manager inside the application is used to select new GIF or JPG files for defined areas of the application screens. The images selected to be used are stored in metadata.
- User Manager – an administrator can define new users of the application along with their logon information and system privileges.

Directory Structure Driven

As well as various types of metadata, the application uses specific directory structures to store system data and save various definitions. The specified directory structure exists at three levels (the paths to which are held in metadata):

- The system area - contains all code templates, input data and parameter templates. If any new inputs are necessary, for example the inputs for a new year, they can simply be placed in the correct directory structure.
- The user area - contains all the users defined jobs (and the pre-specified directory structure for those jobs) and the user level metadata
- The utility area - contains a dustbin for deleted jobs to be stored, a public area for users to share their work, and a directory for images to be dropped so that different pictures can be selected in the image manager (described above) to change the look and feel of the application.

Development and Technical Difficulties

The application proved to be a complicated system for a web-based application. There were therefore several difficulties during the development process:

- There are long periods of time where nothing is returned to the browser, for example when tax modelling code is being run on a large number of records. The browser would receive connection timeout errors making it unable to forward to the next screen. This problem is due to the use of proxy servers in the clients network architecture. The proxy server forwarding the browsers connection to the application server would timeout and therefore lose the browsers connection to the program being run. This is currently being solved by by-passing the proxy servers in the advanced connection features of the browsers internet options.

- Rogue SAS processes were being left on the Unix server which would continue to run and use up memory. Processes are left behind when an error occurs in the application or when users exit the browser itself instead of using the applications own Exit function (which contains the Java code to stop the SAS connection). This was a major problem especially as large numbers of processes were left on the process lists which had to be killed manually. Error trapping was attempted which would stop the SAS connection but not all circumstances could be identified. Eventually a way of combating this problem was found by tying up the SAS connection with the HTTP session itself. No matter how the application is exited, the HTTP session used by the browser times out after a preset time period. A Java class was written which implements the HttpSessionBindingListener. The valueUnbound method of this listener traps the event of the session closing and closes the SAS connection at the same time.
- There are several screens which have a large number of input fields to be edited by the user. Dynamically displaying these fields on the screen using the metadata, while also making the screens efficient and appealing to the user was challenging. Many more metadata fields had to be added in order to build these screens and allow the administrator more control over the display of the input fields. Further training was then required to instruct the administrator about the impact of this extra functionality.
- Speed issues were initially a concern as information has to be constantly transferred between the client and the server to fill screen objects and save definitions. The application initially tried to avoid using SCL classes in favour of the newer Java technologies. However, longer delays were experienced using the Java equivalents to the SCL methods. For example, retrieving a value from a data set using a data set interface class directly in Java took about 4 times longer than performing the same task by calling an SCL method and passing the retrieved value back to the application as a parameter. The SCL classes were therefore used more prolifically than anticipated.
- When it became time to publish the application to the server machine, the Integration Technologies installation on the Unix server seemed incomplete. SAS could not find a series of necessary SCL classes. The catalogues that hold these classes were located on a Windows machine and had to be ported across to the Unix server.
- AppDev Studio had various difficulties generating the Java proxies necessary for interfacing with SCL classes through its InformationBean Wizard. Memory problems were also encountered while running AppDev Studio even though the development machines exceeded the recommended specification requirements.

CONCLUSION

In summary, this project has pushed the limits of what we feel is appropriate for a web based application using AppDev Studio and Integration Technologies. The metadata driven approach, required for the future proofing and extensibility of the system, is demanding in terms of functionality, performance, and maintenance. At the time of writing, the application is in the final stages of user and acceptance testing and is expected to go live in early 2003. Although the project has been a challenge, the application is proving to have both met and exceeded the clients original business requirements.

TRADEMARK CITATION

CONTACT INFORMATION

We very much welcome your feedback, comments, and questions on this paper. Contact the authors at:

Amadeus Software Ltd.
Orchard Farm
Witney Lane
Leafield
Oxfordshire
OX29 9PG
United Kingdom
www.amadeus.co.uk

clare.nicklin@amadeus.co.uk
daniel.morris@amadeus.co.uk

APPENDIX A – SYSTEMS ARCHITECTURE

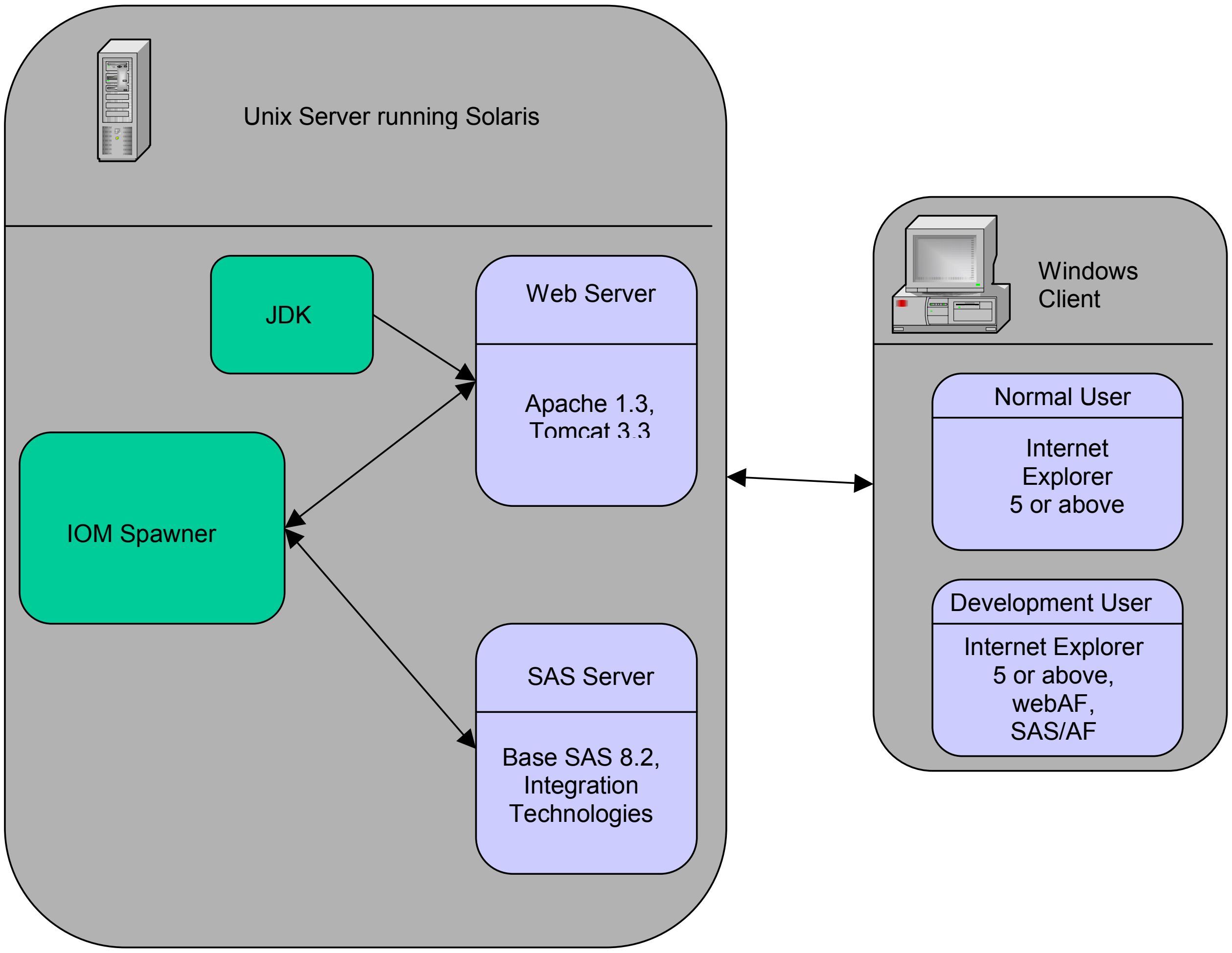
Unix Server running Solaris
JDK
Web Server
Apache 1.3, Tomcat 3.3
IOM Spawner
SAS Server
Base SAS 8.2, Integration Technologies
Windows Client
Normal User
Internet Explorer 5 or above
Development User
Internet Explorer 5 or above, webAF, SAS/AF

Figure 1 - Systems Architecture

[1] http://httpd.apache.org
[2] http://jakarta.apache.org
[3] http://java.sun.com
[4] Some metadata predominantly affects the development team, for example environment level metadata or menu contents. As such it is changed by directly editing the SAS data sets storing the metadata rather than building any further managers within the application itself.