Paper 389-2008

SAS® for Real-Time Applications

Paul Kent, SAS Institute Inc., Cary, NC
Bryan Wolfe, SAS Institute Inc., Cary, NC
Dan Jahn, SAS Institute Inc., Cary, NC
James Carroll, SAS Institute Inc., Cary, NC
Dan Dotson, SAS Institute Inc., Cary, NC

ABSTRACT

The SAS® language is well suited for expressing program logic; people have been using it for over 30 years. There are many applications for building predictive models and then deploying these models to provide scores "on demand" as integrated components in a larger system. The paper describes and explores several patterns for using SAS® in a real-time or near-real-time context, and describes the performance characteristics of each pattern. 1. Stored Processes; 2. Web Services; 3. Raw Sockets; 4. Message Queues.

REAL-TIME BUSINESS INTELLIGENCE

Real-time business intelligence enables people to make better decisions. Decision makers could be company executives or anyone in a company; trading partners; and customers. Decisions based on current information are more reliable than decisions based on information that is stale.

Business intelligence should be integrated into the processes of an organization before it can be considered real time:

> There is a continuous flow of information throughout the organization. Processes do not wait for humans. Humans handle the exceptional cases. The amount of information flowing continually increases. Processes consume and create data. Information is in the data; business intelligence extracts information from data and applies information to the decision-making process.
>
> Processes are easily changed. Change is good! Processes can be replaced by more efficient processes. Changing the implementation of a process is easy. Changing the interfaces between processes is not easy. It is difficult to predict how changes to the input of a process will affect the output of the process.
>
> Processes are interrelated; changes in one process, or the data the process consumes, can result in changes in other processes. One process frequently uses another process. Processes might be internal and controlled by your organization; or they might be external and controlled by a trading partner or customer. Reliable processes can't fail if any of the processes they consume fail.
>
> The effect of a change in a process is observed by the process via feedback. Event delivery provides instant feedback. Feedback can be either positive or negative.

"ON DEMAND" AND "REAL TIME" AS IT RELATES TO SAS APPLICATIONS

Wikipedia has a useful definition of real time (http://en.wikipedia.org/wiki/Real_time). The definition includes the distinction between *hard* and *soft* real time. Hard real-time systems, like the computer control system embedded in your car, are not the topic of this paper – the focus here is on *soft* real-time applications that people have built using SAS® Software.

This paper describes systems that have these real-time and on-demand attributes:

> The activities that complete the process are automated. Decision making is completed with no human intervention required or where humans handle the exceptions only.
>
> The system is reliable.

The response is provided soon enough to be useful.

There is usually a provision to upgrade the system while it is running. Upgrades can be applied without disrupting the users of the system.

There is usually a degree of abstraction between the client (user) of the system and the program that implements (services) the request.

There are usually cooperating programs and processes in the on-demand scenario. The following are common scenarios:

A user is in a Microsoft Office program and wants to embed the results produced by another program.

An SOA-enabled client wants to call a Web service endpoint that is implemented by SAS 4GL

A credit decision-making engine that wants to *score* the current applicant against a predictive model.

There are many scenarios that call for exe*cutin*g a body of SAS language (syntax) "on demand." *SAS 4GL* refers to the fourth-generation programming language implemented in Base SAS® software, and extended by the other SAS products like the FORECAST procedure in SAS/ETS® Software.

When designing a system of cooperating programs like this, you'll be well served to consider the typical volume (number) of requests the server side will be expected to service, as well as the *amount of work to be done* in each service request. One useful distinction is whether the program fragments can be expressed purely in the DATA step or not – from an efficiency perspective, you should consider whether the intended application can afford to compile the program "on demand," or perhaps the volume requires that you find a way to compile the program once and have it service a queue of requests.

RELIABILITY AND SERVICE-ABILITY AND THE OTHER IT-MANDATED "-ILITIES"

How will your design handle changes? Volume increases and you need more servers; programs need updating and you can't take a protracted outage to do this; delivery guarantees have been made and you need to monitor the system to make sure you are meeting the requirements.

Successful real-time applications are deployed into production. This usually means that the ongoing care and feeding of the live system will be handled by a separate group of operations-oriented folks – the IT staff. They will want standardized ways of managing and monitoring the system.

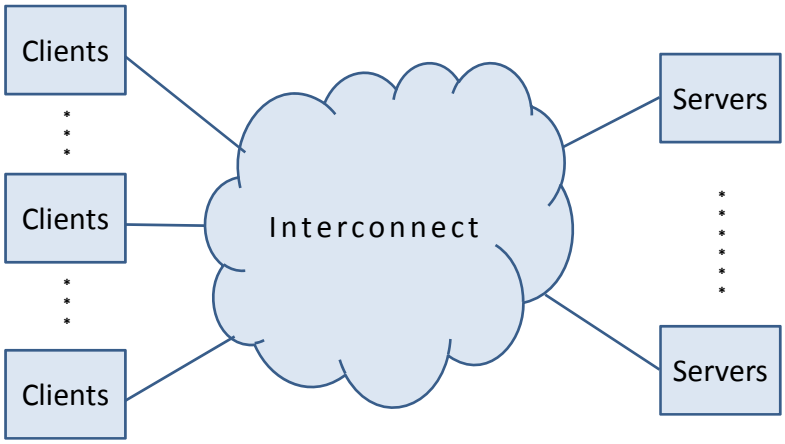

Figure 1: An abstract architecture diagram for a soft real-time system

There are one or more clients of the real-time application. There are one or more servers that provide the service. There is a switch, or interconnect fabric, that routes a client request to a server and returns the response when it is ready. The details are up to you. At some level of detail you have to concern yourself with the strategy you'll use for the *main loop* in the server to do the following:

wait for a request
extract the parameters that came along with the request
invoke the required logic
return the results
repeat

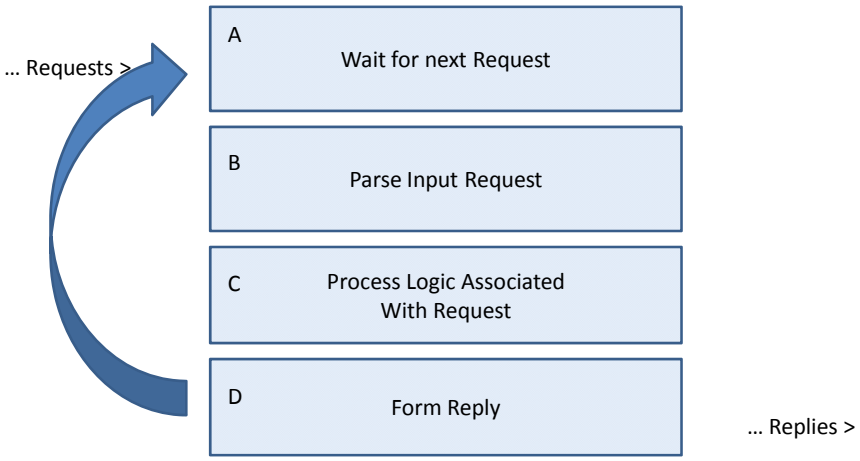

Figure 2: The main loop diagram for a soft real-time system

From a SAS perspective, the server can implement this loop outside of SAS (that is, invoke a fresh SAS session for each request); it might implement this loop inside one SAS session by waiting for a request and effectively %including the appropriate SAS 4GL code, or it might implement the loop in a self-contained DATA step.

FOUR SYSTEMS EXPLORED

This paper describes four modern systems built to handle returning the results of SAS code executed on demand.

1. Stored process architecture as implemented by the SAS Intelligence platform
2. Web service architecture as implemented by the SAS Intelligence platform
3. Web service architecture as implemented by the SAS® Real-Time Decision Manager component of SAS® Customer Intelligence suite
4. The Fraud Detection engine as implemented by the Credit Card Fraud detection solution in use at major commercial banks

Once these systems are described, the paper examines the choices the designers made, and the performance tolerances of these infrastructures. Contrast is drawn between the following:

1. How the client expresses the program name and any input arguments
2. How SAS transfers the request from the client to the server
3. How the server schedules the request for execution
4. How the server returns the result to the client
5. How the user updates the program fragments that service the requests (and the tolerance for no results while doing this)
6. The performance characteristics, such as how many requests per minute/hour are “reasonable,” on a given class of hardware

SAS® STORED PROCESSES ARCHITECTURE

SAS Stored Processes are traditionally SAS language programs that use SAS DATA steps, procedures, macros, and even SAS Screen Component Language (SCL), to deliver powerful SAS capabilities to client applications across an enterprise. SAS Stored Processes are stored and managed from a central server and must be described by metadata which includes the name of the process, information about where each process is stored, how it is executed, and what output it generates. This catalog of the stored processes and the description of the name and form of each input and output is guaranteed by the stored process architecture – an improvement over a library of SAS macros where details like this are optional and not available for programmatic inspection.

SAS Stored Processes open SAS capabilities to any client – from a simple Web browser to an application based on Java or Windows. Users can then interact with SAS in different environments according to their skills, needs, and personal preferences. SAS Stored Processes also introduce standard behavior and standard interfaces – thanks to the catalog of processes and the metadata that describes the inputs and outputs. The same stored process can be executed from a variety of clients. SAS® 9.1 includes interfaces that enable a variety of users to execute stored processes from Web browsers, Java clients, Microsoft Excel, Microsoft Word, SAS Web services and SAS® Enterprise Guide®.

Because stored process code is maintained and managed on a central server, a client application can execute the latest available version of the code and receive and process the results. This ensures that information distributed by the stored process throughout the organization is consistent. The process is updated when the code is updated on that central server.

There are various ways in SAS to generate stored processes and make them accessible to others. They can be generated programmatically in the SAS editor or interactively via SAS Enterprise Guide (the graphical user interface to SAS). Using SAS® Data Integration Studio, administrators can also generate a stored process for a job.

SAS Stored Processes are slightly different from traditional SAS programs. The SAS® Integration Technologies server implements the *main loop* described earlier and waits for requests (Section A in Figure 2). The server parses the request and extracts the name and input parameters. The name is translated to a filename containing `SAS 4GL` and the server expects the SAS programs to begin and end with required SAS macro invocations. These macros implement sections B and D of the main loop diagram shown in Figure 2, and the body of the process can be any SAS code that implements the desired function – A DATA step, a single procedure, or many procedures in a row.

The SAS stored process architecture and the techniques for calling a stored process are unique to SAS. Different components provided by SAS can interoperate smoothly, but interoperability with a wide range of other systems is not practical.

PERFORMANCE CHARACTERISTICS

A modest 4-CPU-class server machine can easily handle 1000 stored process invocations per hour. Of course, that is exclusive of the actual SAS work done in section C of the main loop diagram in Figure 2; an expensive optimization or regression procedure, or a massive DATA summarization will typically take the same amount of time as it would as a batch job. The SAS® Stored Process Server supports a pooled configuration such that several servers (possibly on different computers) can service a single queue for increased throughput and enhanced reliability.

WEB SERVICE ARCHITECTURE

One of the challenges of real-time business intelligence is getting services that need to be used in a real-time process to work together. Sometimes these services are running on different operating systems and provided by different software systems. Sometimes these services are all running on the same machine; sometimes they are running on machines throughout the world. Services also need to span company boundaries, where a process might include calls to services provided by your trading partners or your research colleagues. The most common technology used to solve this interoperability challenge is Web services.

Web services provide a standard way to exchange documents between a service provider and a service consumer. These standards stack on top of each other, with each layer in the stack building on the layer under it and adding some feature. The stack starts with XML. XML describes the format of the documents that get exchanged between

clients and servers. The actual content of the document (the schema includes the input and output "parameters" as well as the method name) is described by a Web Services Description Language (WSDL) document. The Simple Object Access Protocol (SOAP) standard describes how a document actually looks when it is sent across the wire. These three standards (XML, WSDL, and SOAP) describe the minimum requirements necessary to interoperate using Web services. There are many more standards that build on these (all those WS-* standards); the most notable are the security-related standards.

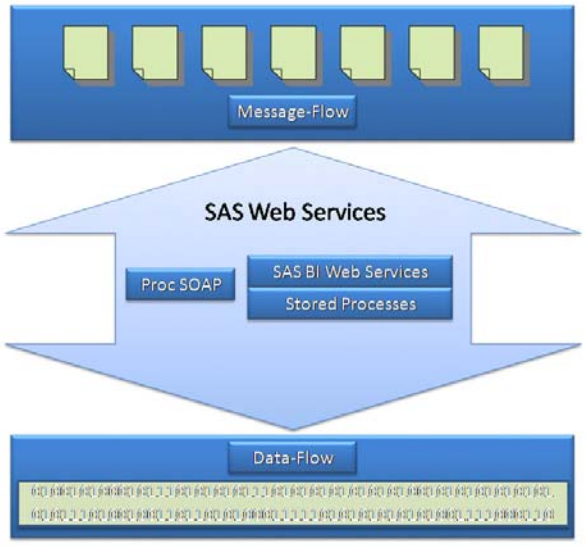

Figure 3: SAS Web services

SAS provides a way to call Web services (using the SOAP procedure) and a way to be called as a Web service (SAS BI Web services). These two features of SAS combine to bridge the gap between the documentflow-based Web services world and the data-flow SAS world. In message-flow systems, relatively small amounts of data are contained in discrete messages. In SAS, data flows have no limit on how much data can be analyzed by SAS. These Web service features make it easy to incorporate SAS into your real-time systems, enabling SAS to call into other systems and enabling SAS to be called by your real-time systems.

DEFINING THE SERVICE

Think of SAS BI Web services as a technology that provides a standard international electrical adapter useful in any country for the "unique" electrical plug/socket required by the SAS Stored Processes. Before the service (adapter) can be defined, a SAS developer must define a stored process. Once the stored process has been defined, the developer can go into the **`Folders`** view of SAS® Management Console, right-click on the stored process, and select **`Deploy as Web Service…`**. The Deploy as Web Service wizard runs, and it enables the developer to specify where the service is to be deployed and what it should be named, as well as a namespace for the service and any keywords. Deployment of the service causes a new WSDL to be generated. To continue the electrical plug analogy, WSDL describes the specific adapter – its shape and the number as well as the orientation of prongs. The local side of the adapter is the "native"l plug of the land from where you hail, and the foreign side of the adapter is the unfamiliar prongs, posts, and blades found on electrical plugs in the country you are visiting.

```
……
<wsdl:types>
<s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/dan" >
 <s:import namespace="http://support.sas.com/xml/namespace/attachments-9.2" />
 <s:element name="MeanVar">
   <s:complexType>
     <s:sequence>
       <s:element minOccurs="1" maxOccurs="1" name="datasetname" type="s:string" />
       <s:element minOccurs="1" maxOccurs="1" name="colname" type="s:string" />
     </s:sequence>
   </s:complexType>
 </s:element>
<s:element name="MeanVarResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="MeanVarResult" type="s:double" />
     </s:sequence>
   </s:complexType>
 </s:element>
……
<wsdl:operation name="MeanVar">
  <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Calculate the mean
 of a provided column in a dataset</wsdl:documentation>
  <wsdl:input message="tns:MeanVarSoapIn" />
  <wsdl:output message="tns:MeanVarSoapOut" />
 </wsdl:operation>
……
```

In the first WSDL segment shown in the preceding example, you can see the standard XML schema is used to show that the Stored Process has two input prompts: datasetname and colname – both are strings; and the returned value is MeanVarResult, which is a double. The second segment in the preceding example demonstrates how the name of the stored process (MeanVar) matches the name of the operation.

CALLING THE SERVICE

In order to call a service, a developer must implement code in the client. This could be SAS code that uses PROC SOAP; SAS code that uses the new XML libname engine; Java code that uses Axis2; .Net code that uses the Windows Communication Foundation; or one of many, many other options. In all cases, the client application needs to create an XML document that looks like this:

```
 <?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
  <MeanVar xmlns="http://tempuri.org/dan">
    <datasetname>string</datasetname>
    <colname>string</colname>
 </MeanVar>
 </soap:Body>
 <soap:Envelope>
```

When using PROC SOAP with SAS, you can actually copy this into a fileref to call the service. PROC SOAP gives you the option to either include the SOAP envelope and body elements or just pass the contents of the body and PROC SOAP will add the SOAP for you.

SAS CODE EXECUTION

SAS BI Web services are a part of the multi-tier SAS® System which runs on the middle tier in a standard Web service container (JBOSS, WebLogic, WebSphere, or IIS). SAS BI Web services transform the XML document received from the client to a call into a stored process, passing along the name and the described inputs. The SAS BI

Web service implements sections A and B of the main loop diagram shown in Figure 2; calls the regular SAS® Stored Process Server to handle section C, and implements returning the result as the XML SOAP message payload in section D.

GETTING DATA BACK

SAS BI Web services wait for execution of the stored process to complete; then it transforms the returned data from the SAS Stored Processes into XML. Here's what the client would receive:

```
 <?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <MeanVarResponse xmlns="http://tempuri.org/dan">
      <MeanVarResult>double</MeanVarResult>
    </MeanVarResponse>
  </soap:Body>
</soap:Envelope>
```

UPDATING THE SERVICE

A fundamental rule of Web services is that you must not change the WSDL for a service once someone is using the service. This is school-yard politics! A deal between a client and a server is a deal – no welching, going back on your word, or changing terms of the deal (of course, if no one uses the service, you can have at it).

However, it is easy and expected that you can change the underlying implementation of a service. In SAS Stored Processes terms, this means that you *must not* change the prompt, parameter, data source, or data target metadata for any stored process deployed as a Web service. It also means that you *may* change the SAS code that is used to implement the stored process. If you change the SAS code that is used to implement the SAS Stored Processes (either by changing the file that contains the SAS code or by changing the metadata that points to the SAS code), that change will be picked up the next time the stored process is executed. This ability to easily change a service is one of the tenets of real-time computing.

PERFORMANCE CHARACTERISTICS

Because SAS BI Web services are a thin wrapper around stored processes, the performance of the Web service matches the performance you get for SAS Stored Processes (see above.) One particular note for both SAS BI Web services and PROC SOAP is that you don't want to move a lot of data through Web services. Web services incur the overhead of XML as well as the advanced messaging features. Your goal should be to move a lot of data through SAS and provide summaries of that data through Web Services.

SAS REAL-TIME DECISION MANAGER ARCHITECTURE

SAS Real-Time Decision Manager is a new product that became generally available in December of 2007. The product provides 24/7/365 real-time, interactive, process-based decision support with enterprise-level scalability and availability. It leverages business rules and SAS analytics to provide intelligent recommendations to customer-facing and interactive applications, such as Web sites, call centers, point-of-sale systems, and internal legacy systems. It is a component of the SAS Customer Intelligence suite.

As a service-oriented architecture driven by Web service requests, SAS Real-Time Decision Manager can easily be integrated into an organization's topology at almost any point. For example, it could be configured to field requests directly from a call center to orchestrate interactions between customers and customer service representatives; or, it could be driven by an in-house CRM system to enforce global customer contact policies; or, individual SAS Real-Time Decision Manager decision flows could be included as Business Process Execution Language (BPEL) diagram nodes within a system that manages long-running, collaborative business processes. The possibilities are broad and varied.

SAS Real-Time Decision Manager is specifically designed for real-time decision support where the task at hand is to traverse a "Tree of Alternative Choices." Some decisions, with respect to the branching points in the tree, are simple

and expressed by the business user designing the tree of choices (we call it a flow). Some decisions are more involved and can be referred to a Web service, or alternatively, to a specific scoring model built and exported using SAS® Enterprise Miner™ and SAS® Model Manager – for example a flow that decides what kind of credit card to offer a caller might consult a predictive model to decide which customer segment this prospect is likely to fall into.

SAS Real-Time Decision Manager is a value-added layer that builds on top of the concepts of the SAS Stored Process Server, of Web services, and orchestrates the execution of a series of SAS programs "on demand" – all based on business rules specified by the user.

For the business user:

SAS programs are presented as black-box building blocks, called *activities*, which can be combined by business users to construct decision flows using simple drag-and-drop techniques.
A rich set of activities is included in the box.
A central repository supports business user collaboration in the design of decision flows.
Decision flow testing and debugging features are provided.
Recursive composition of flows is supported, allowing complex decision flows to be constructed from simpler ones – and promoting reuse.
As an option, a scoring activity allows some or all of a model's predictors to be interjected in real time.

For the IT/administrative people in an enterprise's data center:

Workflow for promotion of artifacts through development, test, and production environments is enforced.
Centralized control and monitoring of multiple distributed deployments is implemented using the JMX open standard.
Alerting functionality notifies system administrators of any failures or timeouts.
Push-button generation of WSDL is provided to aid publishing of new Web services.

For the statistician/SAS programmer and data mining practitioner:

Any arbitrary SAS program, including DATA steps, PROC steps, and descriptive metadata, can be published as a black box activity for business users to include in their decision flows.
Integration with SAS Model Manager enables score code to be published into SAS Real-Time Decision Manager from the SAS Model Manager application.
SAS Real-Time Decision Manager provides automatic synchronization and distribution of SAS program code across servers in a SAS server cluster.

PERFORMANCE CHARACTERISTICS

SAS Real-Time Decision Manager offers the following performance features:

Hot deployment of decision flows with zero downtime is provided..
Server clustering is supported in both the Java J2EE middle tier and SAS server tier.
Multiple heterogeneous decision processes, distributed within and across server boundaries, execute concurrently.
Hardware and software failover is supported, including loss of the management MBean server, decision-process timeout, task cleanup, compensation transactions, and delivery of optional default responses. Servers can be removed or added with zero downtime.

A detailed discussion of the rich feature set of SAS Real-Time Decision Manager is beyond the scope of this paper. For additional information, refer to the white paper located on the Web at http://www.sas.com/apps/forms/index.jsp?id=wp&cid=3679 or contact your SAS representative.

The remainder of this paper drills down into the core production engine and examines the processing of a single request. Finally, the paper describes the performance characteristics of SAS Real-Time Decision Manager.

DECISION FLOW EXECUTION

For the purposes of this discussion, assume that a decision flow has been created, tested, promoted into production, and activated. Activation causes the middle-tier engine, which is a J2EE application, to read the flow metadata from the repository, parse its control logic, and cache the results in memory. Cache contents are then synchronized across all servers in the middle-tier cluster, causing each server to be ready to field Web service requests for the newly activated decision flow. The middle tier flow code is reentrant to support multiple concurrent executions. Note also that multiple heterogeneous decision flows can process requests concurrently.

The major runtime components include: The SAS Real-Time Decision Manager engine server, which is deployed across multiple machines in a middle-tier WebSphere application server cluster; a SAS connection server, whose multiple processes are distributed within and across SAS server tier machines; and SAS® Management Console, which provides control and monitoring. Third-party software includes IBM WebSphere Application Server, IBM WebSphere MQ, Xythos WebDAV, and typically a relational database management system.

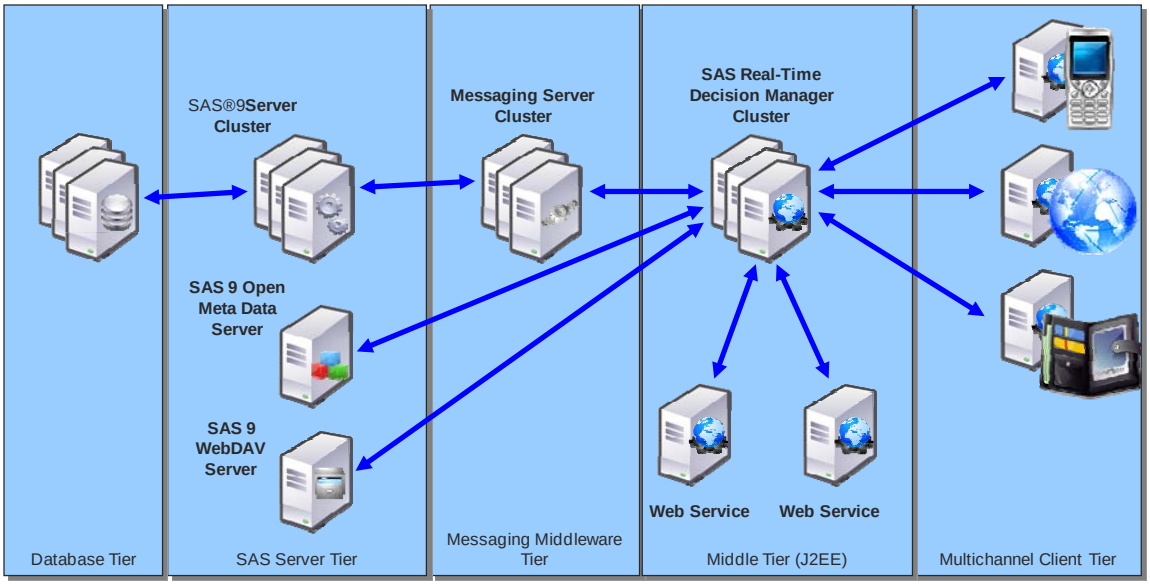

Figure 4: Major runtime components of the decision flow

Suppose we have a retail business with a SAS Real-Time Decision Manager system installed to support a Web site and an inbound call center. Let's examine a simple cross-sell example.

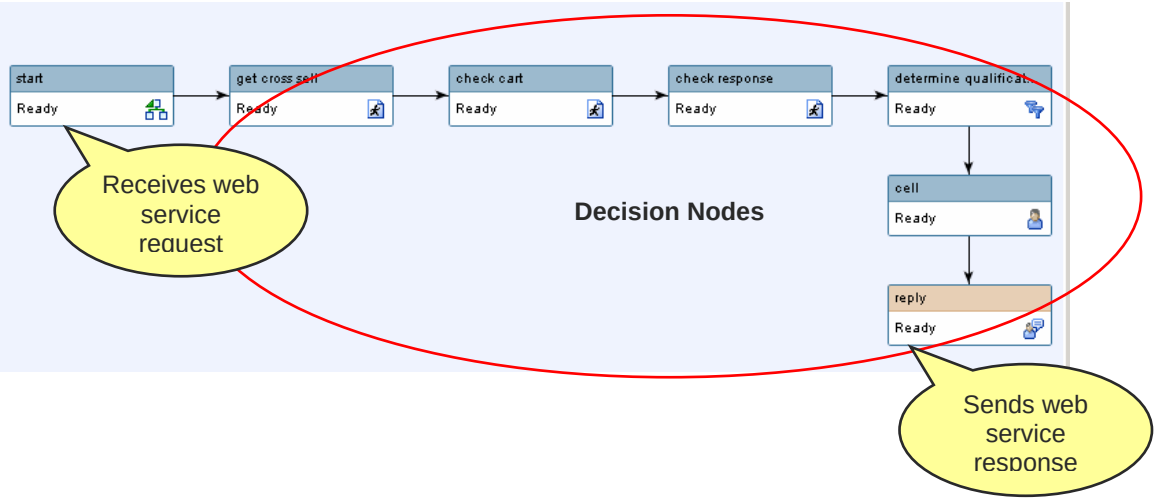

Figure 5: The decision flow for a simple cross-sell example

When a customer calls the call center and purchases a product, the call center application sends a Web service request to SAS Real-Time Decision Manager, requesting the best cross-sell offer to present. Each active decision flow in SAS Real-Time Decision Manager handles one uniquely named Web service request. So when a cross-sell request is received at the Web service endpoint, SAS Real-Time Decision Manager runs the appropriate decision flow.

When the cross-sell event is received, the engine spawns a timer thread and a worker thread. The decision flow is executed by the worker thread. A timeout value, based on configuration metadata, is used to detect if the decision flow exceeds its execution time window. Each time an activity completes execution, the timeout value is reduced by the elapsed time. If a timeout occurs, the timer kills the process and performs cleanup. Depending on choices made by the business user, a default offer might be returned or a user-defined compensation action performed.

A *Start* activity places the request data into a block of in-memory variables known as *process variables*. The decision flow executes one activity after another until a *Reply* activity is reached, which sends the results of the decision flow back to the call center via the Web service response message. Each activity performs an action. The activity reads the data needed to perform the action from the process variables and it writes the results of the action to the process variables. In this way, downstream activities can use the results of upstream activities as inputs.

Some activities are pure Java and perform control functions, evaluate expressions, format messages, and so on, but the majority of heavy lifting is performed by SAS. Each activity performs a unique set of actions. In the case of a "SAS activity," each action corresponds to a unique SAS program deployed in the SAS server tier.

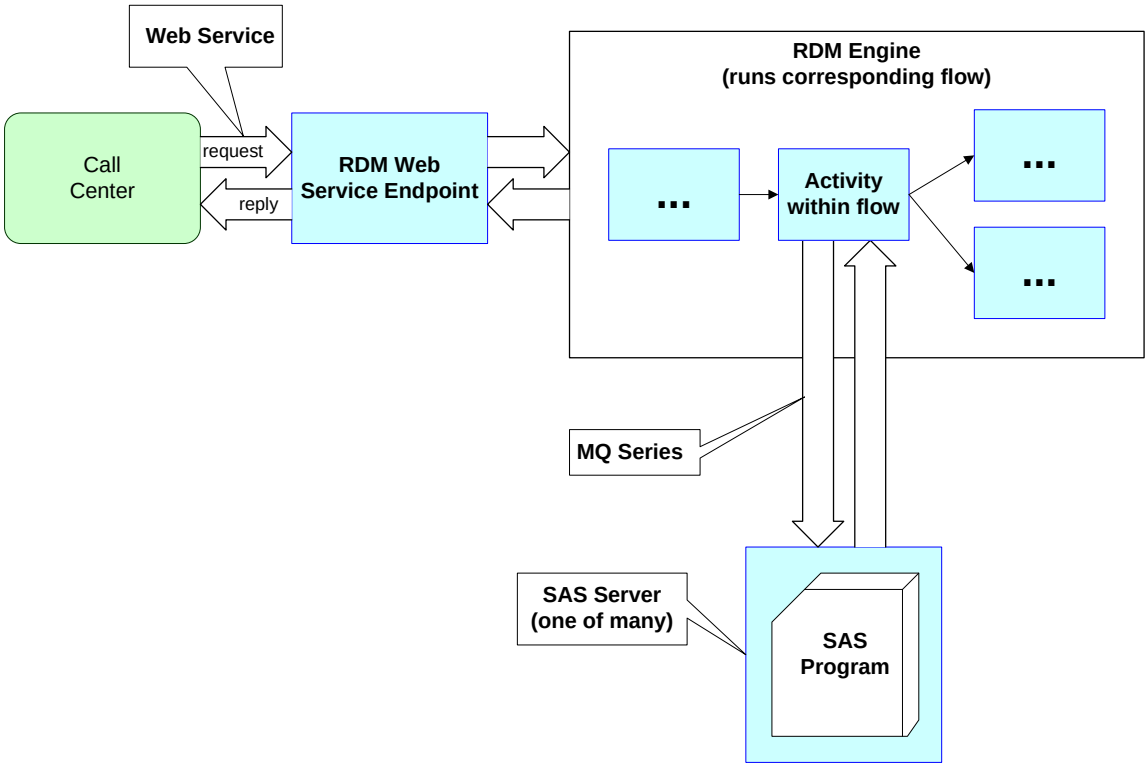

Figure 6: The decision flow of a "SAS activity"When a SAS activity is encountered during flow execution, the engine formats a JMS request message that specifies the SAS program to run along with any input arguments. Each SAS server cluster has a pair of WebSphere MQ queues associated with it: a request queue and a response queue. The worker thread posts the message to the request queue and then waits on the response queue. Because many decision flows can use the SAS server cluster concurrently, the worker thread sets a unique ID in the Correlation ID field of the JMS request message header. It then uses a message selector when waiting on the response queue to

ensure that the response matches the appropriate request.

Because the name and the form of the inputs to the SAS activity program are known at the time one designs a flow, there is no need to navigate the metadata for each request, nor is there any pressing need to use the SOAP protocol to communicate to the SAS server. For performance reasons, XML is not used in the message payload. Rather, easily parsed delimited strings are used. Here is an example request:

```
true^exe^VariableTest^1^false^false^6^inputFloat^double^inputInt^integer^inputStrin
g^string^inputStringArray^list^string^inputDate^date^inputBoolean^boolean^25000.0^7
^DebtConsolidation^3^ccounts^reqd^ning^1.81977163546E8^true^6^outputFloat^double^ou
tputInt^integer^outputBoolean^boole
an^outputString^string^outputStringArray^list^string^outputDate^date^
```

And an example response:

```
0^Successfully Executed

Program^exe^1^6^outputFloat^double^outputInt^integer^outputBoolean^boolean^outputSt
ring^string^outputStringArray^list^string^outputDate^date^25001.11^9^false^noitadil
osnoCtbeD^3^ning^reqd^ccounts^08OCT1965:05:12:44^
```

Note: Although the examples are simple, the protocol supports a rich set of data types, multiple record responses, and error reporting.

On the SAS server tier, the SAS® Integration Technologies MQ polling server is leveraged along with a universal SAS program and a macro library specific to SAS Real-Time Decision Manager. These components are collectively known as the SAS connection server. During the publishing process, SAS programs are converted to SAS macros and persisted in an autocall macro library. This strategy was chosen for performance because autocall macros are loaded on demand and stay resident in memory.

The SAS Real-Time Decision Manager universal SAS program within the SAS connection server is loaded by the SAS object spawner and runs continuously. Two patterns of execution are supported.

The first pattern is as follows. It contains an infinite loop, which:

1. waits for a message to appear on the request MQ queue, and pulls the message off the queue – Section A of the main loop diagram shown in Figure 2
2. unmarshals the input parameters into macro variables – Section B in Figure 2
3. invokes the SAS macro indicated by the input arguments (that is, the SAS program assigned to a macro variable, which represents an activity) – Section C in Figure 2
4. marshalls the response values, copying them from macro variables into the response message payload – Section D in Figure 2
5. posts the response to the response queue
6. loops back to step 1 and continues

This pattern of execution is capable of running any arbitrary SAS program that exists in the autocall library. The program must be complete and might contain DATA steps and PROC steps. There is a performance penalty for this flexibility, however, as considerable overhead is associated with crossing a step boundary in SAS and recompiling a SAS DATA step or re-interpreting the options and statements used for a SAS procedure.

This leads us to the second pattern of execution. The second pattern is similar to the first, but processes the entire loop (steps 1 through 6) with a single DATA step. The autocall macro library is not used, and each DATA step has a dedicated queue pair associated with it.

PERFORMANCE CHARACTERISTICS

Performance testing has been performed on modest, inexpensive hardware, namely one Sun 4 CPU UltraSPARC 1.5 Ghz running Solaris 10 for the middle tier and one identical machine for the SAS tier. Requests were sent from multiple concurrent client applications. Several representative flows of medium complexity were tested. The flows

performed I/O to an Oracle database. Gigabit network connections were used to connect the tiers.

In this environment, pattern 1 conservatively supported 1500 requests per minute while maintaining sub-second response time for each. At 1500 requests per minute, the average response time was 0.147 seconds. At 3000 requests per minute, the average response time was 0.516 seconds. Pattern 2 supported just over three times the throughput of pattern 1.

The factor found to influence performance the most was database I/O bandwidth. The second most significant factor, as you can see by the preceding numbers, was crossing a step boundary in SAS. Flow complexity was also a significantly influential factor.

SAS FRAUD ENGINE ARCHITECTURE

SAS has partnered with a leading financial institution to develop a scoring and logic engine to meet the business needs of financial institutions for real-time transaction processing. The characteristics required for such an environment are as follows:

- extremely high reliability and availability
- ability to host very large and complex multi-part mathematical models
- ability to implement user logic-fragment constructs that represent business rules
- very low latency (that is, fast response times)
- high throughput and linear scalability – the system must make a “isProbablyFraudulent/isNot” decision at the point of sale for each credit card transaction presented to the financial institution
- CPU efficient operation

The result of this effort is the SAS® OnDemand Scoring Engine. The market demand for this engine has manifested first for the IBM z/Series mainframe servers. The overall architecture and approach will translate to other hardware platforms as market demand arises.

The key components of the SAS OnDemand Scoring Engine are as follows:

- Client Interface Program
- OnDemand Scoring Engine Controllers
- OnDemand Scoring Engine Servers/Engines

Here are the details of these components:

CLIENT INTERFACE PROGRAM

The SAS OnDemand Scoring Engine has been developed as part of a larger vertical solution called SAS® Fraud Management. That larger solution includes a full-function client component that provides card authorization systems based on CICS with “one-stop shopping” for a fraud scoring service. This component is beyond the scope of this paper. At the scoring engine level, there is a single client stub program designed to be CALLed by client application programs to perform the following functions:

- Start = Establishes the logical connection between the client address space and the named controller. Establishes cross-memory environment.
- Ping = Validates connection between client and server via token message flow.
- Shut = Terminates logical connection.
- null = Standard message flow between client and named server.

SAS ONDEMAND SCORING ENGINE CONTROLLER:

As the name implies, the controller is the heart of the SAS OnDemand Scoring Engine system. It functions as a task started by z/OS, and is typically started at system IPL time and remains active continually. At initialization, the Controller reads a configuration file, which defines the servers to be established, and the authorized clients for each.

Then, according to the configuration the following processing occurs:

A console communications environment is established.
SAS OnDemand Scoring Engine components are started, always in redundant pairs.
A z/OS cross-memory communications environment and transaction queue are established for each logical server.
SAS OnDemand Scoring Engine readiness is confirmed.

SAS ONDEMAND SCORING ENGINE:

As previously explained, the SAS OnDemand Scoring Engine is started by the controller, as specified in the controller parameter file. These engines run as started by z/OS and are simply Base SAS ® DATA step batch jobs running in an infinite loop with a structure approximately as follows:

```
data _NULL_;
    ConnectToXMS_environment();
    Do while not END_SIGNAL;
           Receive_Client_ByteStream_via_OSE_custom_function();
           Parse byte-stream into SAS variables;
           Invoke one or more fraud detection models supplied as compiled macros;
           Invoke customer supplied SAS logic fragments built via Rules IDE;
           Send_modified_segments_of_ByteStream_via_OSE_custom_function();
    end;
```

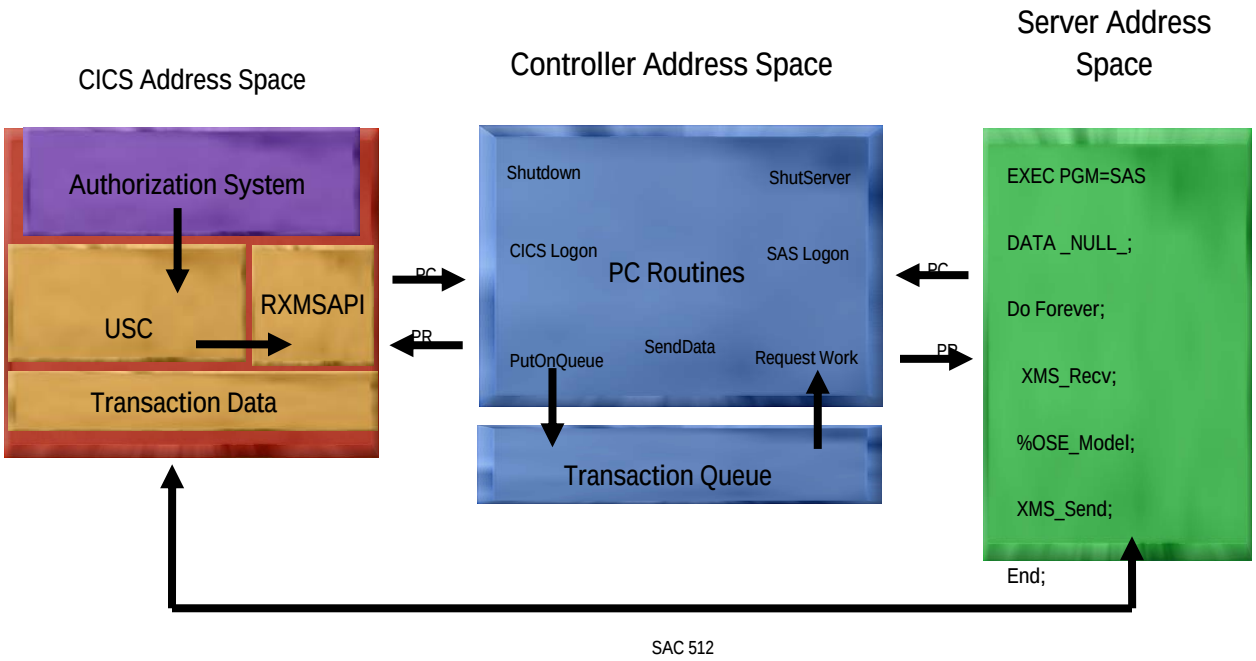

Figure 7: SAS OnDemand Scoring Engine environment schematic and transaction flow

DESIGN CHOICES MADE:

1. *Client expression of target program and data:* The stub client program presents a fixed-format parameter list for the client to populate. Key among those elements are the following:
    a. controller name
    b. server name
    c. pointer to byte stream input transaction

2. *Request transfer from the client to the server:* When the client program calls the SAS stub client the following events occur:
    a. The caller is blocked via system WAIT.
    b. A pointer to the client byte-stream transaction is placed on the controller queue (but not the data itself, which remains in the client address space).
    c. The controller inspects the array of engines available for this request and dispatches an engine as needed.
    d. When an available engine issues the XMS_receive(), the byte-stream transaction is moved en masse to the engine address space.

3. *Server scheduling of the request:* The controller dispatches engine address spaces according to parameters supplied in the server configuration file. Once dispatched, each engine remains active until the XMS_receive() blocks, meaning that the input server queue is depleted.

4. *Response return by the server:* At the bottom of the server infinite loop DATA step is a custom function XMS_Reply(). Invocation of this function causes the following to occur:

a. Those byte stream segments of the input transaction identified (by definition) as read/write are written directly into the client transaction buffer.
b. The client caller is resumed.

5. *Handling of customer-provided logic fragments:* SAS Fraud Management includes the complete Web-based SAS® Rules Studio, which caters to the complete life cycle of logic development, testing, and deployment. When deployment is invoked via this IDE, a script is invoked which triggers a refresh of the rule source code and recycling of the engines. Because engines are always deployed in redundant pairs at a minimum, non-stop operation is assured. The controller orchestrates the sequential cycling of the individual engines so that service is maintained at all times.

PERFORMANCE CHARACTERISTICS

As stated at the beginning of this section, the SAS OnDemand Scoring Engine environment has been designed from the beginning to exhibit robust performance characteristics. The initial customer has been in production with their credit card authorization system for over 18 months.

Throughput: The customer typically process 4 – 5 million transactions daily and encounter peak transaction volumes in the 400 – 500 requests-per-minute range.

Latency: In production experience at fairly high throughput rates, the customer typically sees the transit time for these large byte streams (12 K bytes) of less than 1 millisecond. Engine processing time for very large SAS models and customer-provided SAS DATA step logic run in the 2- to 4-millisecond range.

Processing Efficiency: The customer has reported an overall reduction in operating costs as compared to a competing scoring engine, even though the total transactions processed has increased. Base SAS has proven to be an effective, efficient, highly reliable transaction processing engine, even when very large transactions, containing many hundreds of individual fields, are processed.

WRAPPING IT ALL UP

There are several ways to skin a cat. The perl community is fond of TIMTOWTDI, but SAS folks are not far behind. We hope the description of these architectures helps you design yours – it might be the case that it maps directly onto one of these, or it might happen that you have to use a blend of these techniques.

The SAS stored process architecture and the BI Web services that enable you to export SAS Stored Processes are general-purpose tools that support a wide range of clients and usage patterns. The lookup and dispatch of the logic at the core of each process is “late bound” – determined at the time of each request. This flexibility carries a modest performance cost, as does the fact that each request is re-interpreting the SAS 4GL.

The SAS Real-Time Decision Manager component of SAS’s Customer Intelligence suite adds more “business user” value to the assembly of a real-time decision-making process from a library of SAS programs. Essentially, SAS Real-Time Decision Manager enables the business user (who might not be a crack programmer) to assemble programs and deploy them for real-time usage.

The SAS OnDemand Scoring Engine is a true “hot-rod;” It is purposely built for handling logic expressed purely as SAS DATA step and is used in a “compile once, execute many” fashion.As you move away from the general (SAS Stored Process Server and BI Web services) and towards the specialized (SAS OnDemand Scoring Engine), you can realize orders-of-magnitude more throughput by your application. There are situations, like with SAS Real-Time Decision Manager, where you should consider isolating the business user from your technology choice, and focus on the layer above, enabling users to assemble “programs” visually, and using a vocabulary that has more in common with the business domain than the programming domain.

BIBLIOGRAPHY

Hugos, Michael. Building the Real-Time Enterprise: An Executive Briefing. ISBN: 978-0471678298

CONTACT INFORMATION

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

Paul Kent, Bryan Wolfe, Dan Jahn, James Carroll, Dan Dotson
SAS Institute Inc.
Box 8000
Cary, NC, 27513
USA
919 677-8000
E-mail: First.Last@SAS.COM