PharmaSUG 2017 - Paper AD18

Leveraging Centralized Programming Resources Using SAS Integration Technologies

Tony Chang, Amgen, Thousand Oaks, CA

ABSTRACT

Remote access from multiple locations to centralized SAS® services can greatly benefit to the globalization of pharmaceutical and bio-tech industry. SAS Integration Technologies allows client application to interact with SAS services from any remote sites where different system and infrastructure may be used. This paper describes a custom client application that we developed using SAS Integration Technologies. The application was used for a subsidiary company running medical dictionary coding for their clinical study data by leveraging SAS modules that invoke Amgen's centralized coding system. The tasks accomplished by our client application are explained in this paper, which includes dynamically generating SAS program by the client application, running SAS program on the remote server, and transferring data sets between the client machine and the SAS server. The benefits of the client application using SAS Integration Technologies are also discussed.

INTRODUCTION

The globalization of pharmaceutical and bio-tech industry raises a number of challenges for statistical programming. Statistical programmers often work with the team members located globally; a CRO company may need to support their clients across the borders; a subsidiary company may have to follow the same programming process defined by its global headquarter, and etc. Therefore, accessing centralized programming resources, such as SAS macros and utilities will benefit the remote sites in many ways. However, different system platform and infrastructure may be used by different sites. Duplicating the same programming components, such as macros and utilities may require identical system and infrastructure to be installed and maintained in every site, which will not only increase the cost, but also introduce inconsistencies if those components were modified locally without global consensus.

SAS Integration Technologies allows users build client application to integrate with SAS services in where the remote server is located. It provides a set of application programming interfaces (APIs) and let users access SAS features on the remote SAS servers through client applications. These features include running SAS programs, invoking stored procedures, transferring data sets between the client machine and the remote server, and etc.

ABOUT OUR USE CASE

Before acquired by Amgen Inc., Onyx Pharmaceuticals was working with multiple vendors on the medical dictionary coding for their clinical studies. Basically, Onyx programmers need to extract un-coded medical event and medication terms from study data into text files then transfer the text files to the coding vendors. The vendors will code the terms and turn them back to Onyx. Onyx programmers will then convert the coded files to SAS datasets. The coding vendors also will ask sponsor review some of coded terms to ensure that the terms were coded correctly. The feedback raised by the sponsor must be sent back the vendors for necessary recoding. Sometimes, it takes several cycles of these kinds of communications before coding can be finalized.

The coding process involves multiple manual steps. The entire process is very timing consuming and prone to human errors. A typical coding cycle may take up to 2 to 3 weeks. Thus study data can only be coded weekly or monthly, which may delay study database lock, especially for the large clinical studies. The regular coding and upversioning was also costly. In addition, different vendors often use different coding conventions, which might introduce inconsistencies among the studies.

On the other hand, Amgen has a central Medical Coding Group (MCG) with global expertise in standard dictionary coding. The MCG also maintains its own dsNavigator database. Global Statistical Programming group developed SAS macros that interact with the dsNavigator and perform auto-coding for clinical studies during data extraction from RAVE database. Therefore, it will be beneficial for Onyx to adopt Amgen's standard coding process and utilize Amgen Central Coding System. However, Onyx was running different network system and using different platform of SAS server from Amgen. The Onyx users were not able to directly access Amgen's Central Coding system.

ABOUT THE SAS INTEGRATION TECHNOLOGIES

SAS Integration Technologies provides a set of APIs that enable users build client-server infrastructure on which to implement SAS distributed processing solutions. The SAS Integration Technologies allows users to integrate SAS with other applications and extend the capabilities of SAS to meet their specific needs. Users can also develop their own distributed applications using the SAS Integration Technologies to leverage the analytic and reporting powers of SAS.

Figure 1 illustrates the client-server architecture using SAS Integration Technologies. The SAS Integration Technologies supports several interoperability standards, including SAS/IOMP, COM/DCOM, and CORBA/IIOP based client-server architecture. Therefore, multiple programming languages can be used for developing client applications (on the left side of the diagram), such as Visual Basics, VBA, C++, Java, Cobol, Perl, etc. The client applications will need to connect SAS Integrated Object Model (IOM) servers (on the right side of the diagram) to access SAS services provided by the remote SAS servers.

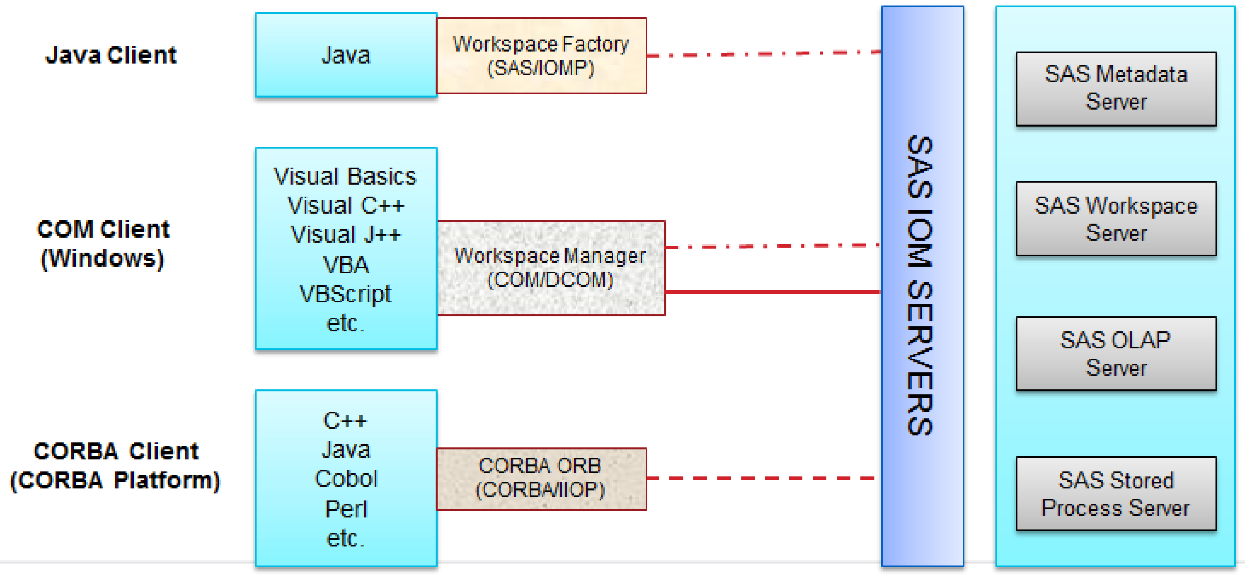

Figure 1. The SAS Integration Technologies Architecture

The coding problem that we had in Onyx was a great use case for us to use SAS Integration Technologies. Basically, we can develop a custom client application using SAS Integration Technologies to access the central coding resources located in Amgen.

In this paper, we will look at the Java Client API provided by the SAS Integration Technologies that we used for developing our Onyx Coding Client.

ABOUT THE ONYX CODING CLIENT

In order to take advantages of Amgen Central Coding System and adopt standard coding process, the Onyx Coding Client (OCC) will perform the following tasks:

- Transferring study data sets from Onyx to Amgen's file system.
- Invoking Amgen SAS Coding Module which will perform auto-coding for Onyx study data.

- Calling Amgen SAS Post-Coding Module which will generate data query reports for data management team.
- Sending coded study data sets and data query reports back to Onyx server.

Figure 2 demonstrates the data flow of targeted coding process, and how the OCC will invoke the SAS modules in Amgen which will further interact with the dsNavigator for clinical study data coding.

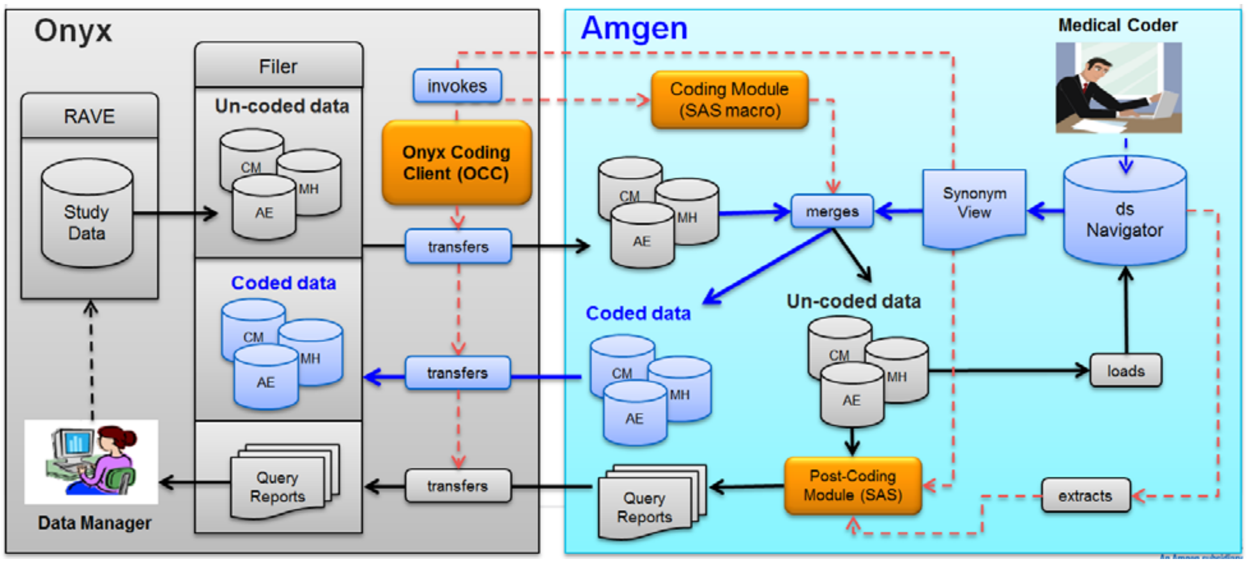

Figure 2. Targeted Coding Process and Data Flow Diagram (User's Perspective)

The OCC is a standalone Java client application using the SAS Integration Technologies Java Client API. It contains the following object classes.

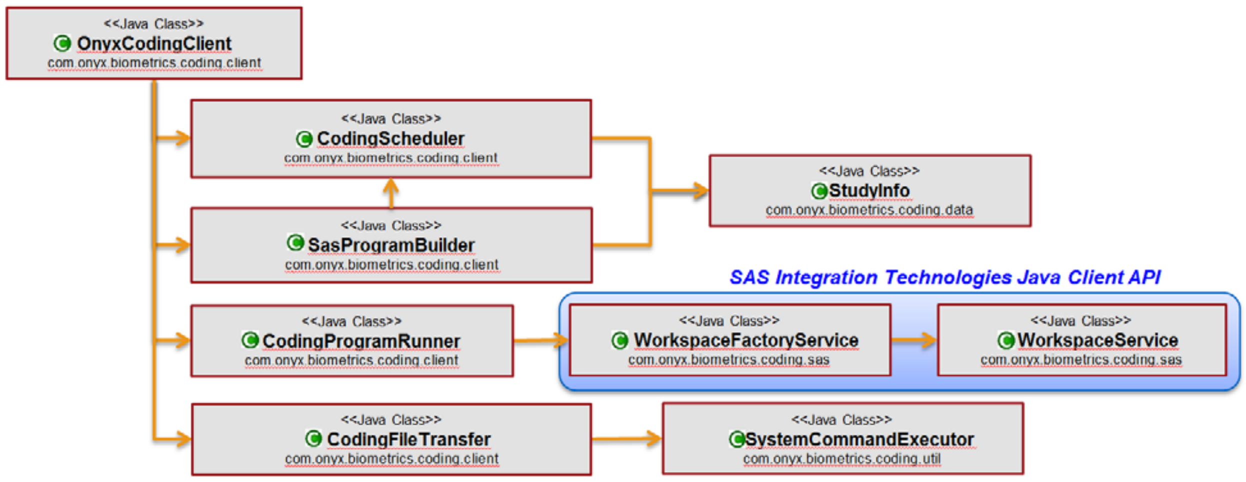
<<Java Class>> OnyxCodingClient com.onyx.biometrics.coding.client
<<Java Class>> CodingScheduler com.onyx.biometrics.coding.client
<<Java Class>> StudyInfo com.onyx.biometrics.coding.data
<<Java Class>> SasProgramBuilder com.onyx.biometrics.coding.client
SAS Integration Technologies Java Client API
<<Java Class>> CodingProgramRunner com.onyx.biometrics.coding.client
<<Java Class>> WorkspaceFactoryService com.onyx.biometrics.coding.sas
<<Java Class>> WorkspaceService com.onyx.biometrics.coding.sas
<<Java Class>> CodingFileTransfer com.onyx.biometrics.coding.client
<<Java Class>> SystemCommandExecutor com.onyx.biometrics.coding.util

Figure 3. The Major Java Object Classes in OCC Design

Figure 3 illustrates the relationship among the Classes.

1. **OnyxCodingClient**: the main class drives the Onyx Coding Client.
2. **CodingScheduler**: manages coding schedule (day and frequent) for individual studies.

3. **SasProgramBuilder**: dynamically generates SAS program based on the coding schedule that invokes coding and post-coding SAS modules.
4. **CodingProgramRunner**: establishes a connection to a remote SAS server, then submits and runs a SAS program, finally retrieves the SAS log from the SAS server.
5. **CodingFileTransfer**: transfers study data sets to Amgen and moves coded data sets and post-coding reports back to Onyx filer.

WORKING WITH SAS INTEGRATION TECHNOLOGIES JAVA CLIENT API

The Java Client API provided by SAS Integration Technologies enables users to develop Java-based distributed applications that are integrated with the SAS platform. Let's walk through the major steps in our OCC implementation.

STEP 1: CONNECTING TO A SERVER

In order to use SAS services in a remote server, the first step is to configure a Java Connection Factory and obtain a connection to an IOM server.

There are a few ways that the users can use to establish a connection from client to an IOM server.

1. **Connecting with Directly Supplied Server Attributes**: user can place required information directly in the client program. The connections can be made one at a time on an as-needed basis, or you can set up a pool of connections.
2. **Connecting with Server Attributes Read from a SAS Metadata Server**: user can obtain the required information from managed, secure SAS Metadata Server using indirect logical names. We used this approach in our client application.
3. **Connecting with Server Attributes Read from the Information Service**: if user configures SAS Foundation Services, then user can obtain the required information from a SAS Metadata Server by using Information Service.
4. **Connecting to a Zero-Configuration Workspace Server**: in Windows environments, user can create a connection to a local server without specifying connection information.

We used the second approach and read the IOM Server Attributes from a SAS Metadata Server. The following example code was used in our client application to initialize and use the Java Connection Factory with information from a SAS Metadata Server directory. The Metadata Server connection information was captured in a utility configuration file that is passed into the application.

```
// Get the SAS BI environment configuration properties
String metadataHost = configProp.getProperty("metadataHost");
Integer metadataPort =
Integer.valueOf(configProp.getProperty("metadataPort"));
String reposId = configProp.getProperty("reposId");
String logicalServerName = configProp.getProperty("logicalServerName");

// 1. Create a connection factory configuration for the Metadata server
Server omrServer = new BridgeServer(Server.CLSID_SASOMI, metadataHost,
metadataPort);
ConnectionFactoryConfiguration cxfConfig_omr =
            new ManualConnectionFactoryConfiguration(omrServer);

// 2. Create a connection factory for the Metadata server connection factory configuration
ConnectionFactoryInterface cxf_omr = cxfManager.getFactory(cxfConfig_omr);

// 3. Get a connection to the Metadata server
ConnectionInterface cx_omr = cxf_omr.getConnection(username, password);
```

```
// 4. Narrow the connection from the Metadata server
org.omg.CORBA.Object obj_omr = cx_omr.getObject();
IOMI iOMI = IOMIHelper.narrow(obj_omr);

// 5. Create a connection factory configuration for the server by passing
// the server logical name to the Metadata server
ConnectionFactoryConfiguration cxfConfig =
            new OMRConnectionFactoryConfiguration(iOMI, reposId,
logicalServerName);

// 6. Get the connection factory and administrator interface
getConnectionFactory(cxfConfig);

// 7. Get a connection to the server
ConnectionInterface cx = cxf.getConnection(username, password, authDomain);

// 8. Narrow the connection from the server
org.omg.CORBA.Object obj = cx.getObject();
IWorkspace iWorkspace = IWorkspaceHelper.narrow(obj);

// 9. Get a language service instance for the workspace
ILanguageService languageService = iWorkspace.LanguageService();
workspace.config(cx, languageService);
```

Once the connection is established, the client program can invoke SAS services provided by the IOM server, including submitting a SAS program to the IOM server.

STEP 2: RUNNING A SAS PROGRAM AND RETRIEVING THE LOG

The SAS language component of the IOM server enables user to submit SAS code for processing and to obtain output and information in the SAS log.

We created a Java Class “**SasProgramBuilder**” to dynamically build SAS programs for executing coding tasks for individual studies based on the coding schedule. The following code shows an example of coding program built by the **SasProgramBuilder** that invokes the coding module located in Amgen server.

```
/*
 * Program Name : \\filer02\apps\coding_client\programs\occ_sas_coding.sas
 * Purpose      : Coding calls for all scheduled studies
 * Author       : Onyx Coding Client (auto-generated, please do not edit)
 * Date         : Mon Jun 29 12:03:09 PDT 2015
*/

*--- macro calls for study 001 ---*;
%let study = study001;
%let dataInLocation  = /filer10/coding/&study/uncoded_data;
%let dataOutLocation = /filer10/coding/&study/coded_data;
libname in001  "&dataInLocation" access=readonly;
libname out001 "&dataOutLocation";

%m_util_coding(study=&study, libin=in001, libout=out001, ds=AE);
%m_util_coding(study=&study, libin=in001, libout=out001, ds=MSH);
%m_util_coding(study=&study, libin=in001, libout=out001, ds=PCM);
%m_util_coding(study=&study, libin=in001, libout=out001, ds=MMHC);
%m_util_coding(study=&study, libin=in001, libout=out001, ds=MH2);
%m_util_coding(study=&study, libin=in001, libout=out001, ds=CON_DST);
*--- end of macro calls for study 001 ---*;
```

```
*--- macro calls for study 002 ---*;
<Macro call for study 002 ……>
```

After the SAS programs were created, the OCC will submit the SAS code though its language services to the IOM server. The following example code shows the methods with the **`languageService`** object for submitting the SAS code and retrieving the SAS log.

```
// Read a SAS program to a string array and submit it
String [] code = readLines(sasProgramFilename);
languageService.SubmitLines(sasCode);

// Retrieve the SAS log
CarriageControlSeqHolder logCarriageControlHldr = new
CarriageControlSeqHolder();
LineTypeSeqHolder logLineTypeHldr = new LineTypeSeqHolder();
StringSeqHolder logHldr = new StringSeqHolder();

languageService.FlushLogLines(Integer.MAX_VALUE, logCarriageControlHldr,
            logLineTypeHldr, logHldr);
String[] logLines = logHldr.value;

// Output SAS log to a file
FileWriter sasLog = new FileWriter(sasLogFilename);
for (int i = 0; i < logLines.length; i++) {
      sasLog.write(logLines[i] + "\n");
}
sasLog.close();
```

STEP 3: USING OTHER SERVICES IN JAVA API

You may use a few of other services provided the Java Client API to manage the connection activities between your client application and IOM servers.

- **Logging Java Connection FactoryActivity**: in SAS 9.3 and later, the connection factory uses `Log4j` to perform logging tasks. User may configure `Log4j` for their client application.
- **Using Failover**: if user develops mission critical client application, the SAS Integration Technologies provides failover that enables a Java Connection Factory to redirect connection requests in the event of server unavailability.
- **Using Load Balancing**: if the client application will deal with heavy load business, the SAS Integration Technologies provides load balancing that enables a Java Connection Factory to distribute server load between a cluster of redundant servers.
- **Using Connection Pooling**: user can also create a pool of connections to IOM servers.

Since our OCC is a simple client application that will be mainly used by one single system service account. Therefore, we did not implement those features in our client application. You can find related information from SAS Website.

STEP 4: RETURNING CONNECTION TO JAVA CONNECTION FACTORY

Lastly, when you are finished using a connection that you have obtained from the Java Connection Factory, you must return the connection to the factory so that it can be either reused or canceled. To return a connection to the Java Connection Factory, call the `close()` method on the connection that was returned when you called the `getConnection` method.

When you are finished with the instance of the Java Connection Factory itself and you no longer need to request connections from it, you must shut it down so that any remaining connections can be canceled and other resources can be released. To shut down the Java Connection Factory, either `shutdown()` or `destroy()` method can be called.

```
// Close SAS work space services
cx.close();

// Shutdown the connection factory
if(onyxWsFactory != null) {
      admin.shutdown();
}
```

STEP 5: TRANSFERRING DATA SETS

The SAS Workspace provides **`FileService`** API, which allows you to transfer file-based content between your local application and the SAS session. The **`FileService`** defines methods for managing SAS file references (**`FILEREFs`**) and reading or writing files on the server's host file system.

We actually did not use the FileService API for transferring data sets between client machine and remote server. We created a Java Class **`CodingFileTransfer`** that executes the system command to copy the data sets between client's and server's filers.

DISCUSSIONS

Fast Client Development: with very limited effort, we were able to develop a Java client for our use case. Onyx was able to take advantages of central programming resources and services well developed in Amgen. We built the OCC utility within a short period of time. Total 14 clinical study data were switched to Amgen Central Coding System and followed standard coding process shortly after the utility was released in production.

Financial Gain: after switching to Amgen Central Coding System, majority of medical event terms can be automatically coded with very minimum manual interactions. The utility also produces data query reports for data management to review and query un-coded terms. The turnaround time for data coding can be in daily basis, and also ad hoc coding can be performed as needed. This new process has reduced the coding time dramatically in preparation for a study database snapshot or a database lock. Furthermore, this project also reduced company's coding expenses greatly.

Flexible Architecture: the client-server based architecture provides flexible access to SAS services in different operation systems. OCC was running in a Windows based programming environment, but it can submit SAS programs to a UNIX based SAS server and run the SAS programs on the remote server that further invoke other SAS components, such as SAS macros and utilities on the server side. This architecture allows end users from any remote sites to access centralized programming resources in a consistent manner.

CONCLUSION

SAS Integration Technologies has been used by the industry for many years. It provides users easy access to SAS services through SAS IOM servers. The technology allows users to build client applications in many programming languages. We've successfully developed a Java client application using SAS Integration Technologies for integrating clinical study data dictionary coding with Amgen's Central Coding System for a subsidiary company. With SAS Integration Technologies and limited development effort, you can easily build powerful client applications that utilize the central programming resources for all your remote sites.

REFERENCES

SAS Integration Technologies Java Client Developer's Guide (v9.4)
http://support.sas.com/documentation/onlinedoc/inttech/

ACKNOWLEDGMENTS

The author would like to thank David Edwards and Mark Stetz for their technical support during the OCC development.

CONTACT INFORMATION

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

Tony Chang
Amgen, Inc.
One Amgen Center Drive
Thousand Oaks, CA 91320
Work Phone: (805) 447-3457
Email: tochang@amgen.com