Creating “Plug-n-Play” Applications
by
Hansdip Singh Bindra

In today’s day-and-age of computing solutions for businesses the concept and usage of integrating “Best of Breed” applications is extremely popular and cost effective. Often termed as Enterprise Application Integration (EAI), this bundling of applications to provide a “Total Solution” can successfully be achieved, only with properly designed and architected software.

Progress Software has also realized that EAI is the premiere solution for businesses, and thus has launched its Quadrant II Business Empowerment initiative where ISV to ISV application integration is the key for growing revenues. On the technical end of the spectrum, the advent of the Web Services Tool Kit (WSTK) to get on the .Net bus is another key facilitator towards EAI.

In order to get an application to integrate with another application, or in other words to create “Plug-n-Play” applications that hook into each other with ease, one has to ensure that their application is designed and architected properly. Moreover, their applications need extremely well documented Open API’s, which allow integration specialists to easily link them together.

What is Plug-n-Play?
The answer is actually quite simple – any application with extremely well documented Input and Output Points. In addition to the I/O Points, a Plug-n-Play application will normally have a “Centralized Manager” for those Points, and these Points will be at the “Business Process” and not Business Logic level.

Traditional integration techniques usually involve extensive programming changes to “Both” applications. This involves the use of intermediary Tables or Files and is often referred to as the “Middleman” approach.

Plug-n-Play applications on the other hand have a notion of a “Plug” and a “Socket” application. The addition of a minimal programming layer on the Plug without a middleman or match-maker layer is more than sufficient to plug it into the Socket. In order to facilitate this easy link, the Socket on its part must have a well “Documented” I/O.

Plug-n-Play Advantages
These applications simplify the EAI process, and thus minimize the costs to integrate applications. Since they lower the threshold for integrating applications, software vendors can provide more “Total Solutions” based on “Best of Breed” applications. Plug-n-Play architecture adheres to the concept of “Future Proofing” as they allow the applications to keep up with the latest integration platforms such as: Web Services,
Creating “Plug-n-Play” Applications
Published in Progressions – Spring 2003, Number 53&54.
Published in PRO’s Progress Market Magazine Benelux - 2/2003.

Service Oriented Architecture (SOA), Enterprise Service Bus (ESB), .Net and J2EE. In fact a Plug-n-Play application is “ready” for messaging systems like Sonic XQ/MQ.

Having recently gone through the process of creating such a “Plug-n-Play” application – StepUp Accounting™ with the use of Open API’s it would be appropriate to share the details of this approach. StepUp Accounting™ contains robust G/L, A/P, and A/R Modules; architected as a “Socket” for other ISV’s to “Plug” in their vertical applications.

Open API Approach
In order to incorporate this approach into creating Plug-n-Play applications, one has to transform existing 4GL Business Logic into Processes. A set of such atomic Processes combine to form what is commonly known as Business Logic. This transformation will also involve the independence of the UI from Business Processes. Separation of UI from Business Processes is an absolute must, as the concept of Plug-n-Play also applies within the application. In this case the Plug is the UI and its Socket is the Business Processes.

Once this UI separation is done, the next step is to “encapsulate” the Process inter-dependency, thereby making each such Business Process an independent entity. This layered technique to “Openness” is also known as the “Inside→Out” process. The primary focus of the Open API is “Back-End” integration where Business Processes in one application are linked to such Processes in another application, with little or no UI interaction. Finally, in order to complete the incorporation of this approach into an application, a “Centralized Controller” is needed to manage ALL such Business Processes.

Documentation
The Open API approach to creating a Plug-n-Play application is totally dependent on the availability of thorough design documentation.

“…The MOST important part in Creating a Plug-n-Play Application is the Design Documentation of the Open API Business Processes…”

Design Documentation is typically 2/3rd of the total project cycle. In case of the StepUp Accounting™, our US team spent 6.5 on the Design Documentation, and our Indian team spent 3 months on the Programming and supplemental Documentation.

There are four levels of such Documentation needed for a Plug-n-Play compliant application: Specification, Design, Design Detail, and Open API Detail. These constitute the application “Blue Print” and are supposed to be language independent.

The Specification Documents should justify the business “need” for functionality being provided in the software. Design Documents should expand on that justification, and provide a general overview of the functionality – the business processes in a non-technical manner. The next level – Design Detail Documents, provide a detailed
Creating “Plug-n-Play” Applications.
Published in *Progressions* – Spring 2003, *Number 53&54.*
Published in *PRO's Progress Market Magazine Benelux* - 2/2003.

discussion of the functionality, and describe “How the Application will Solve the Functionality”. These Detail Documents are technical in nature and contain data descriptions and business process layouts.

Open API Documentation
This is undoubtedly the “Core” of the Plug-n-Play approach. It describes the Layered API, and thus we need a separate Document for each Layer – 4GL, Web-Services Wrapper, etc.

Since the target audience for this Document is the integration specialist (architect or programmer), and the goal is to minimize the time and costs associated with the integration process, the suggested approach is a Question & Answer format. The Document should describe the I/O for each Business Process and provide thorough examples for its usage.

A Plug-n-Play Open API Document must provide answers to the following questions:

<table>
<thead>
<tr>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. What are the different types of API’s available in the Application?</td>
</tr>
<tr>
<td>2. What are the Names of available API’s?</td>
</tr>
<tr>
<td>3. What are the I/O Parameters and Parameter Types?</td>
</tr>
<tr>
<td>4. What is the Description and Format of those Parameters?</td>
</tr>
<tr>
<td>5. How to call these API’s (Syntax)?</td>
</tr>
<tr>
<td>6. Is there any Sample Program available?</td>
</tr>
<tr>
<td>7. Is there any sequence to be followed while calling the API’s?</td>
</tr>
<tr>
<td>8. Can a given task be accomplished by calling a single API or a group of API’s?</td>
</tr>
</tbody>
</table>

Once the documentation is in order, the actual 4GL architecture needs to be designed and developed.

Open API Environment
In terms of the 4GL each Business Process translates to an Internal Procedure. These Procedures interact with other Procedures though the exclusive use of Input and Output Parameters. Thus there is no sharing of Variables of Buffers. Larger data sets are handled though a mix of dynamic and static Temp-Tables, and Database Tables as needed. The notion is to create these Internal Procedures, which usually reside on the AppServer as “Self Contained” but “Universally Accessible”. This is achieved though the wonderful SUPER PROCEDURES in the 4GL!

Open API Management
Simply creating compliant SUPER procedures will not solve the needs of creating a Plug-n-Play application. These procedures need an effective manager that controls the entire integration process. Thus, one would need to create a Centralized Manager per Business Function (Module). In case of our application, for the A/R Module we have ARModuleAPIManagerSuper.p. This manager provides a single activation and loading mechanism for all the “Inner-Core” SUPER’s, each of which is programmed in “UI-
Creating “Plug-n-Play” Applications.
Published in Progressions – Spring 2003, Number 53&54.
Published in PRO’s Progress Market Magazine Benelux - 2/2003.

Less” 4GL. We suggest an effective usage of constructs such as “flags” to handle error and status handling, since these procedures should be portable across deployment environments. When these SUPER’s are linked to non-Progress applications, the outer “Layer Interfaces” are provided by WSTK, etc. The Open API manager is also known as the Business Process Manager (BPM).

“Black-Box” Approach – Total Parameterization
The entire Open API process is based on “Inside-Out” architecture and thus provides a layered “Black-Box” approach to the design of the Plug-n-Play application. This approach is normally deployed in a Thin-UI and AppServer environment. However it is flexible enough for both the Client-Server and Host-Centric deployments too; leveraging approximately 80% of the existing Progress applications.

In this layered approach, all “Interaction” between Business Processes takes place through Procedure Signatures only. As indicated earlier, SUPER PROCEDURES are the means to that end. SUPER’s facilitate the maintenance of a “Single Set” of programs, which can be deployed in both “Front-End” and “Back-End” integrations, and allow for easy customization through the concept of procedure overloading. It would be appropriate to describe this layered process in more detail.

In the 4GL environment, the innermost layer is simply a group of Internal Procedures. These procedures are both SUPER (which are used for integration and local application needs), and PRIVATE (which are used for local application only).

Example of Innermost Internal Procedure Layer:

```
PROCEDURE superAPIAPBatchBillItemCreateProc:
   /* Define I/O Parameters */
   . . .
   /* Validates complete contents of A/P Batch details, Bill details & its Item details. */
   IF outAPICreationStatus THEN
      RUN privateValidateAPAPIBatchProc(INPUT  inAPBatchTaskID,
                                          OUTPUT outAPICreationStatus).
   END PROCEDURE.

   /* Calls the Main procedure which Creates A/P Batch, Bill & Bill Items details */
   IF outAPICreationStatus THEN
      RUN privateAPBatchBillItemCreateUpdateDeleteProc(INPUT  "New":U,
                                                        INPUT  inAPBatchTaskID,
                                                        INPUT  ",
                                                        INPUT  "New":U,
                                                        INPUT  0,
                                                        OUTPUT outAPICreationStatus,
                                                        OUTPUT outAPBatchBillRowID,
                                                        OUTPUT outAPBatchBillNo).
   END PROCEDURE.

   /* Clears Temp Table Contents */
   RUN superAPITempAPBatchBillItemEmptyProc.
END PROCEDURE.
```

The next outer layer is groups of .p’s by Application Function. In our product, examples of these are .p’s for A/P Module API Manager, A/P Module Posting Manager, etc. All such .p’s are activated and managed by the BPM.

© 2003-2005 – Innov8 Computer Solutions, LLC.
**Example of .p Outer Layer:**

```plaintext
/** Super Procedure HANDLE’s **/
DEF VAR handleAPAPISuper AS HANDLE NO-UNDO.
DEF VAR handleAPPostSuper AS HANDLE NO-UNDO.

/** Run Persistent Procedures and SESSION: SUPER-PROCEDURES **/
/* A/P Module API and Posting Managers */
RUN SuperProcedures\APModuleAPIManagerSuper.p PERSISTENT SET handleAPAPISuper.
RUN SuperProcedures\APPostManagerSuper.p PERSISTENT SET handleAPPostSuper.

ASSIGN handleAPAPISuper:PRIVATE-DATA = "APModuleAPIManagerSuper.p":U
handleAPPostSuper:PRIVATE-DATA = "APPostManagerSuper.p":U.

/* Add Session SUPER’s */
SESSION:ADD-SUPER-PROCEDURE(handleAPAPISuper).
SESSION:ADD-SUPER-PROCEDURE(handleAPAPISuper).
```

The above two layers of the Open API approach are sufficient for integration between applications written in the 4GL. In order to integrate with “external” non-Progress applications, appropriate “Wrappers” can be added as needed, such as XML, Sonic XQ, Web-Services (using WSTK), .Net, J2EE, etc.

The entire “Inside-Out” Open API layered approach is shown in **Figure 1**.
Figure 1