Title: Presenting the Case for the Free Data Object – FDO

Authors: Geoff Crawford with Hansdip Singh Bindra

The Progress world is on the threshold of "Opening Up". Yes, with the POSSE initiative, Progress will provide access to 4GL based source code for the ADM, AppBuilder and other related Frameworks under some sort of an "Open Source" license. This would allow designers and developers like us to come up with new and robust Frameworks that provide Environment Independent Data Access, Security, User Session Management, etc. Keeping with the spirit of this Open Source initiative, we propose a community based design and development of the Free Data Object – FDO.

The information in this article is the "initial" collaborative effort of a group of Progress and PEG community members, namely Julian Lyndon-Smith, Brian Laferte, Bob Mirro and ourselves. We have been communicating on designing and developing a Data Object for "Non" Smart Object (SO) environments. We have shared some preliminary information and ideas, and have reached a stage where we feel that the project can be implemented by the collaborative genius of the Progress (POSSE) community as a whole.

What is the Free Data Object?
The non-SO environments do not have a generic and universally accepted equivalent of the Smart Data Object for the SO world. The FDO is basically an API based object framework that allows the management of data between the User Interface (UI) and Database (DB) in a non-SO environment. The FDO is UI independent, so Web Client, WebSpeed, N-Tier Client/Server, and Host based systems can all use the FDO.

Why is it needed?
In the current programming environment where UI independence from the DB is of the utmost importance, where UI's themselves are changing rapidly, it is extremely important to remove the Business Logic (BL) and direct DB access from the UI itself. Thus the FDO will allow the Progress 4GL programmer to achieve this UI abstraction in an effortless manner under non-SO environments. Unlike the SO environment, which is closed to itself, and uses a "Top-Down" design approach, the FDO is based on an "Open" philosophy. Its basic Framework will embody a **"Bottom-Up"** design approach, where smaller components – "Atoms" will be combined to build a larger API. The importance of the **"Bottom-Up"** approach, and its contract with the "Top-Down" approach is shown in ***Figure 1***.

Technical/Tools Requirements
The latest Version of Progress – 9.1, has all the items needed to build the FDO Framework. We will need the following constructs to create such an API:

1. SUPER and Internal Procedures.
2. AppServer for the N-Tier "middle-layer" that holds the Business Logic.
3. Regular 4GL statements for Data Access, e.g. FIND, FOR EACH, etc.
4. Dynamic Temp-Tables and Dynamic Queries.

5. Communication between the UI, AppServer and DB will be handled through the normal Progress channels, and furthermore, if needed through TCP/IP Sockets, HTTP tunneling, and URL hits.

Design/Framework – Initial Ideas/Approach

Data Update Object

Currently it is quite easy to totally remove the Data Update (i.e. Add, Update, and Delete) functionality from the UI – place it on the AppServer (both Synchronously and A-Synchronously), on another machine using the DB as an application Manager. The ability to use Temp-Tables as Input/Output Parameters between applications facilitates this requirement. The UI simply works off a single Temp-Table record that represents a Table in the DB. The User works with the Temp-Table, and on completion of the "updates" simply "submits" the Temp-Table to the Business Logic layer – AppServer or DB Application Manager.

Data Retrieval Object

This is somewhat more challenging. One has to build something similar to the Smart Data Object – SDO provided by Progress, where functionality has to be split between the Client and the AppServer, depending on the deployment. This requires three pieces: Communications, Data Query, and Data Management, each of which could be again made into Component Objects.

Data Communications Object

This component should have the "brains" to determine the Data Transport Configuration based on the deployment environment. This environment could be any of the three – split between Client and AppServer, totally on Client, and finally totally on the AppServer. Furthermore, in the split and totally on AppServer, whether it is Stateless (mainly for WebSpeed and Web Client) or State Aware. It should encapsulate the entire context associated with this process.

Data Query Object

This should simply get the data from the DB. This data is then "filled" into the associated Temp-Table(s). We use the word Temp-Table(s) – plural for "real" word environments where one could have an Order Object that contains Order Line Objects. *Data Query Object* would be made up of smaller Objects – *First*, *Previous*, *Next*, *Last*. These in turn would call the actual *Data Find Object*.

Data Transformation Object

The Transformation Object would have the ability to manipulate the data's format for sending to the UI depending on the API call into it, that is whether the Client is requesting data for Fill-In Fields, or for a Grid (Browse), etc. Data Transformation represents the individual differences various UI visualizations have on the data.

Data Management Object

This Object encapsulates the Transaction Management associated with DB Tables. In fact the *Data Update Object* would in encapsulate this Object within it, to handle Table Adds, Updates, and Deletes. Furthermore, this component will contain lower Atoms like *Data Add Object*, *Data Update Object* and *Data Delete Object*.

Validation Object

The Validation Object would contain the validation logic and link to the appropriate *Messaging Obj*ect (whose scope is outside the requirements of the FDO). The need to separate the *Validation Object* from the larger *Data Update Object* is because we use a **"Bottom-Up"** –

Atomic approach. In comparison, the Progress based SO use a "Top-Down" approach, and has the validation routines within the SDO itself. In separating the *Validation Object* from the *Data Update Object*, we provide more flexibility in validation overrides, and allow direct calls to the Object from the UI itself without the overhead of the *Data Update Object*, if business requirements necessitate such a need.

Once these Objects are in place, data handling for the non-SmartObject developer would change from statements like FOR EACH Customer, or FIND NEXT Customer, to simply FOR EACH Temp-Table, etc.

Design/Framework – Further Ideas/Approach

Having presented the initial ideas and approach, it would be appropriate to discuss some of the above stated Objects in more detail.

Data Communications Object

Simple data passing can be done over a Socket, whether a URL or another Socket at the other end. A possibility for actual protocol is XML but it raises another level of complexity. It also requires a dependence on a Generic Data/XML interface. Switching from one Message type to another, or from one Communication method to another would be by a different procedure overloading using a different SUPER procedure as needed.

The URL approach is simply a "partition alias". A Temp-Table contains all of the partitions and their details. One type is traditional AppServer where you get a Handle as done in the SO environment. Another is our method where you get a URL instead of the Handle. A third might use the Sonic MQ messaging mechanism.

Encapsulation is a must for the *Communications Object*. The issue of Stateless and State Aware that comes up in the WebSpeed and Web Client demands this "black-box" approach. Normally, when in doubt, one would impose State Aware on the Stateless. This is the case in the SO environment – pushing the State on a Stack and popping it later. In our design considerations this must be avoided at all costs. The design of the FDO must be such that it no longer requires State under all environments.

Data Retrieval Object

The main issue that comes up with this Object is the time taken to copy data from the DB Table into a Temp-Table is tremendous in terms of "click-click" computing time. The current SDO model of Data Retrieval based on passing Temp-Tables from AppServer to Client displays this inherent overhead of Temp-Table Creates and Deletes. This problem could be eliminated with the advent of Sever Side Joins in Progress. If this were available then, it would be appropriate to have the Web Client simply get the "targeted" data (as the Server Side Join would give the exact subset) directly from the DB – making data retrieval fast, or even get it from an AppServer where the number of Temp-Table rows being sent back would be less than now.

Having unleashed the 30,000 ft description and design of the Free Data Object – FDO and having disclosed the initial nature of the discussions that we have had on this topic, it is now open for the larger Progress (POSSE) community to make its own, to give it a definite shape and form, to get this collaborative Open Source initiative off the ground. Latest updates on the Free Data Object Project will be posted on the PEG POSSE Mailing List – posse@peg.com.

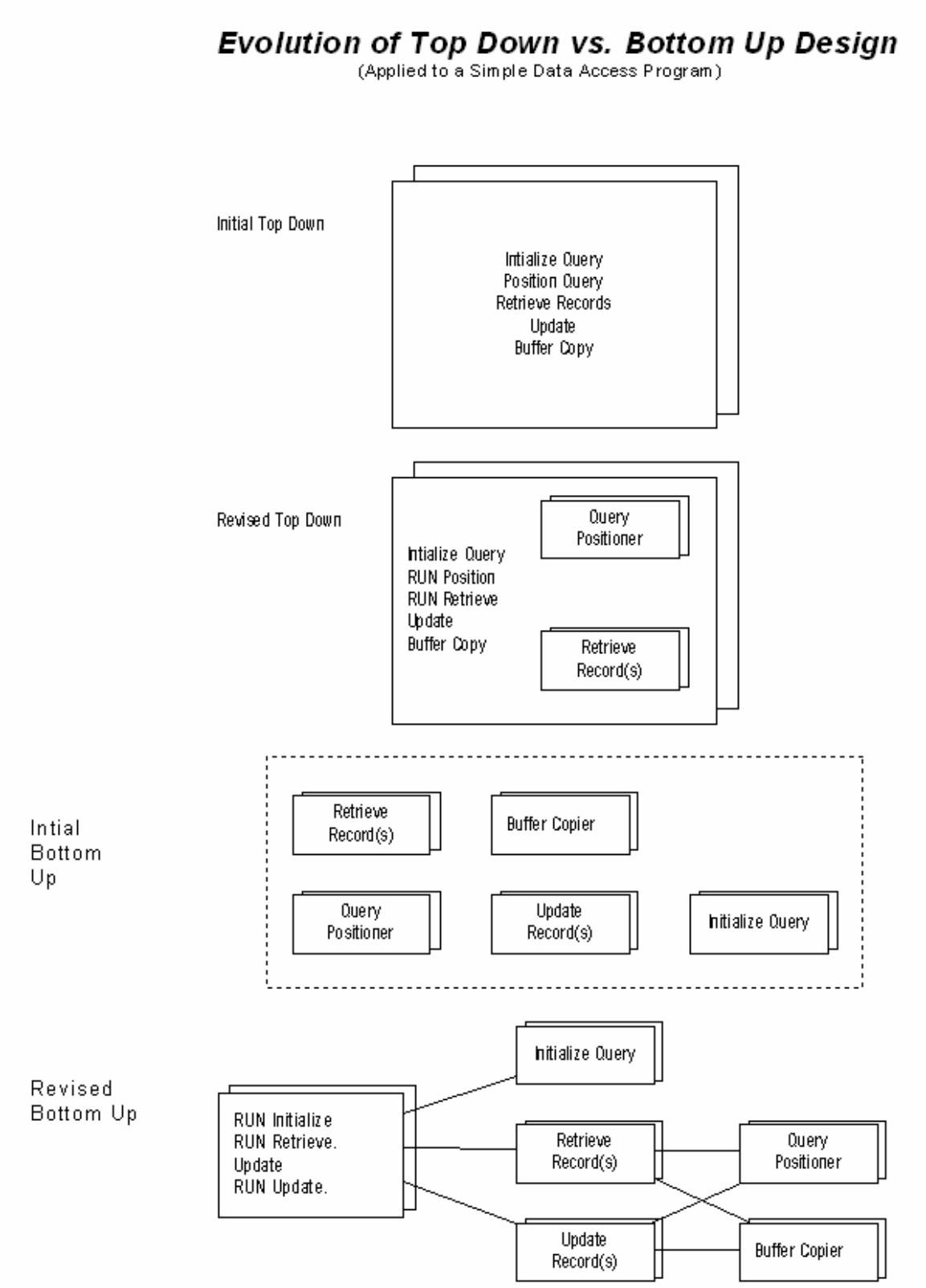

Figure 1. A representation of how a simple Data Access (i.e. Retrieve and Update a record) program evolves from the initial program. Note how the "Top-Down" program starts as a single

procedure and then repeating functionality becomes internal procedures. Compare this with the **"Bottom-Up"** approach, where a set of procedures is called by a controlling program. The more "Atomic" those initial procedures are, the greater chance that no procedures get modified, only called in different combinations.

Phone: (973) 361-4224 Fax (973) 537-6946
E-Mail: info@innov8cs.com Web: www.innov8cs.com

