COMPUTING INTEGRITY
INCORPORATED
3102 Valley Brook Drive
Champaign, IL 61822-6112
217.355.4469

PROGRESS | Application Partner

ABL in a Pacific World

Dr. Thomas Mercer-Hursh
21 October 2013

With the announcement of Progress Pacific[1] in the wake Progress Software's acquisition of Rollbase, it is natural for those in the OpenEdge world to wonder what the long term rôle is for ABL in this evolved product. Of course, most people coming from an OpenEdge background know little yet about what Rollbase is and how it changes the overall Progress offering. Moreover, it is understandable that many OpenEdgers are skeptical about how much Rollbase adds since OpenEdge's own tools for web and mobile solutions are substantial. Currently, Progress marketing is strongly emphasizing the new tool and packaging and it is easy for someone in the OpenEdge world to perceive this as a de-emphasis of OpenEdge and ABL.

In on-line forums and other conversations there have been concerns expressed that ABL was being downplayed in the company's future in favor of JavaScript. This concern was perhaps reinforced by a statement by Phil Pead during the last conference call referring to ABL as an "impediment" (possibly not the best choice of words). On the other hand, Progress has made many strong statements and actions in the last year which show a renewed and strengthened commitment to the OpenEdge product line.

Therefore, I would like to briefly review how I see RollBase and how ABL fits in to the larger picture of Progress Pacific. My hope is that this review will both provide some answers to the concerns of OpenEdge people as well as to provide some guidance for ways that ABL can add more to the offering of Progress Pacific.

What is Rollbase, really, and what will it become?

There is always a certain difficulty in penetrating the marketing positioning of any product to ascertain what it is and is not. This is, perhaps, exaggerated when the product is positioned as being part of a relatively new and unfamiliar category – in this case Platform as a Service (PaaS)[2]. Just as with software which is offered in a SaaS offering, it is useful to make a distinction between the product itself and the package in which it is offered.

With a SaaS ERP package, for example, one can analyze the capabilities and features of the ERP software separately from the subscription licensing or cloud delivery, even though the licensing and delivery will clearly be important in the final product selection. Thus, to evaluate Rollbase as a product in relationship

[1] Pacific is the name Progress has given to the entire Platform as a Service offering including Rollbase, OpenEdge, Corticon, OE BPM, and Data Direct Cloud. Those who have not been to Exchange or been otherwise exposed to the full message on Pacific may have the misimpression that Pacific is a Progress rebranding of Rollbase.
[2] PaaS means providing both the cloud infrastructure and the development platform to create applications on that infrastructure. Thus, it is midway between IaaS (Infrastructure as a Service) offerings like those from Amazon and SaaS (Software as a Service) offering such as a specific ERP application sold on a subscription model.

to ABL, we need to consider Rollbase as a tool separate from the delivery or licensing. This is not to suggest that the delivery or licensing are in any way unimportant, but, rather, that delivery and licensing are properties which are somewhat independent of the tool itself. Thus, while Rollbase is presented as itself a Platform as a Service, specifically a tool on an infrastructure which allows very rapid development and deployment of browser-based applications, we can consider the Rollbase tool separate from the infrastructure which is used to deliver it. Indeed, that infrastructure might well be used to deliver other tools.

Based on presentations and a workshop at the 2013 Exchange conference, my bottom line impression of the Rollbase tool is that it is a nice way to quickly develop and deliver web-based applications, if those applications are basically CRUD[3] applications. Yes, one can include logic via JavaScript, but that is putting business logic in the client, which is not consistent with modern design practice, i.e., OERA. With the continued development of integration with Corticon and OE BPM, it will be possible to add more complex processes and rule evaluation to those CRUD operations, but this is still a long distance from the kind of complex business logic which characterizes “enterprise class business applications”, Progress’ stated market.

To be sure, even an ERP application has components with limited business logic, so that one could be able to supply much of the application with Rollbase, but for any core functionality, one is going to require a connection to server-side complex logic outside the scope of Rollbase and for which ABL is ideal.

In discussions I have had over the years with people who were unfamiliar with OpenEdge, some have dismissed all 4GLs and related tools as “mere” RAD (rapid application development) tools. By this they refer to any one of a variety of tools to allow the very rapid development of CRUD applications, i.e., where the “application” is little more than a conduit between the user and the database. The sophistication of thousands of ABL applications makes it clear that this is not a fair characterization of ABL, but it is certainly more correct as a description of Rollbase.

Yes, the infrastructure makes it special. Yes, there are provisions for providing additional logic, but if that logic becomes substantial and complex, then not only is one putting the logic on the wrong end of the client-server connection, but one is developing that logic in JavaScript, which is not rapid and productive. Yes, one can connect the Rollbase application to Corticon and OE BPM to provide some types of business logic, but in no way do BRM[4] and BPM[5] provide complete coverage of the kind of logic required by a typical ABL application. And, yes, one can connect a Rollbase application to ABL logic in the AppServer, but then one is also not solving the whole problem in Rollbase.

Thus, one can imagine Rollbase and ABL interacting in different ways in the future. In some cases a customer or ISV will have an existing ABL application suite and have a need for a new, browser-based application with limited business logic. Thus, they might use Rollbase to create that application and interface it primarily at the database level with the rest of their applications, possibly providing any needed business logic without the use of ABL. In other cases, such an application might start out appearing to be fairly simple, but as it grew one discovered a need for more sophisticated logic. This could lead to interfacing the Rollbase application with an ABL application via the AppServer. Or, one might have an existing ABL application which needed a web aspect that it did not currently have and use Rollbase through the AppServer to provide that new interface on an existing application. In some cases, a

[3] CRUD = Create, Read, Update, and Delete.
[4] BRM = Business Rules Management, i.e., Corticon.
[5] BPM = Business Process Management, i.e., OE BPM derived from the former Savvion product.

customer might start with an independent Rollbase application and move to adding an ABL back end to address growing complexity of business logic, but this seems less likely.

Where is the future of ABL?

It is useful to briefly consider the history of ABL in order to look at its future. Back in the 1980s, one of the key distinguishing characteristics of ABL when compared to competing 4GLs was that it was possible to write all of most applications in ABL. With the other 4GLs, one typically had to write 5 to 15% of the application in some 3GL such as C. Worst yet, the parts where one had to resort to C were the hard parts of the application so that one gained productivity in the easy part, but gained nothing in the hard part. Using the usual 80/20 type rule, this meant that, for example, a 4X boost in productivity applied to 80% of the code, but that code normally took only 20% of the development time, thus, there would be only a 15% boost in overall productivity because 20% of the code was still taking the full 80% of the time that it would without a 4GL. E.g., given an application which took 1000 hours to develop in a 3GL, the 80% of the code suitable for a normal 4GL, but which only took 200 of those hours, might be done in 50 hours. But, the remaining 20% or 800 hours would still take 800 hours, so the total improvement would be 850 hours versus the original 1000.

Being able to develop 100% of the application in ABL meant that the productivity gain was experienced throughout. I.e., the 1000 hour application could now be done in 250 hours or less. In many cases, the gain with ABL was greater.

Those gains were, of course, on simpler applications than we are dealing with today. Between then and now, the sheer growth in keywords in ABL is staggering. Parts of the ABL have become less 4GLish in response to a need for greater power and control. For example, there is no question that dynamic queries add enormous power to the language, but they also produce code which is less easy to read and more complex to write than the old static methods. While I know of no studies, it seems likely that this has resulted in a more modest productivity advantage for modern ABL over other languages, especially since competing 3GL languages have also acquired some higher level features.

In addition, there has been at least a potential migration of part of the solution out of ABL. One of the more common expressions of this is for the UI. With the advent of browser-based interfaces and OpenClient Java and .NET clients and most recently with OE Mobile it is not uncommon in a modern ABL application for some or even all of the UI to not be written in ABL. According to OERA principles, one should not find this disturbing since the UI proper should be a thin layer which can easily be swapped from one technology to another without disturbing the core system. Similarly, in good OO design, one recognizes a division of an application into highly discrete subsystems or domains and allows for the possibility that particular subsystems may use different technology than the rest of the system. One common example of this separation is that even strongly model oriented developers will not use models for the UI since visual designers are so much more productive for creating a UI.

In more recent years we have seen the introduction of other “foreign” technology subsystems in the form of BPM and BRM systems. In each case, a certain part of the business logic is removed from the ABL code and entrusted to the new domain, not for the sake of reducing the ABL code, but because rules and processes are more easily managed in an environment where they can be modified by visual tools without the need for programming changes.

Thus, we have gone from a historical context where the unique feature of ABL was being able to write 100% of the application within the same language to a context in which best practice indicates that we should partition the solution into multiple subsystems or domains and some of those subsystems will not be ABL. This is not a diminution of the importance of ABL, but rather an indication that modern systems are more complex than ones 20 and 30 years ago and this complexity is best addressed by using multiple tools in combination.

Therefore, in the context of Pacific, where one has the OpenEdge database, ABL, Rollbase, Corticon, OE BPM, and Data Direct Cloud all available as a part of a solution, ABL still has a major rôle to play in any enterprise class business application as the tool by which the core, complex business logic is performed. Rules will be in Corticon; Processes in OE BPM, the web presence possibly in Rollbase, other UI in various visual designers, connectivity to external data via Data Direct Cloud, but everything else is in ABL.

Can we generate ABL from Rollbase to make ABL more "sexy"?

Corticon and OE BPM are "sexy" because they are based on visual models which are readily understood by non-programmers and those models can be rapidly modified in response to changed requirements. Data Direct Cloud is "sexy" because it provides a very simple, visual interface that allows connecting to a very large universe of potential data sources without special development.   Rollbase is "sexy" because it provides an easily learned, highly productive way to build attractive web interfaces with significant functionality. While ABL is more productive than popular 3GLs, it is worth asking if there is a way to make it more "sexy" as well, especially if this would also provide the productivity gains and the rapid responsiveness we associate with Corticon and OE BPM.

There has been some mention in discussions of Pacific about producing ABL code from templates, presumably in similar fashion to what is done with Rollbase for the code it produces. I would like to suggest this is not a good avenue for development.

Back in the 1980s I was a Varnet VAR and Varnet had a template-based program generator as part of their toolset. This was not a particularly sophisticated tool, but I quickly decided that it was not worth trying to extend it.   My key issue was responsiveness to change. If one changed one's mind about how a function should be implemented or about what functionality should be provided in a particular program type one could only implement this changed requirement in the existing code by manual modification of each program, with the consequent labor and potential for error.

In response to this problem, in 1992 I developed a system called Specification-Driven Development (SDD) which used templates and specification files to generate finished programs. This system was fundamentally different from the Varnet system since one could alter a template, rerun the generator, and very, very rapidly incorporate the change in the entire relevant code base. Similarly, changes could be quickly made to the specifications in response to changed requirements and new code generated.   This allowed very rapid initial development[6] and even faster modification in response to changed requirements. In the years following it was not uncommon to spend significantly longer determining the specifications for a particular new piece of functionality than it took to implement.

However, the templates in SDD were very specific to the architectural model which I had at the time, i.e., ChUI and procedural. Modifying them for event-driven GUI code or AppServer separation or OERA layering would have not only meant a whole new set of templates, as well as having to recreate all the specification files as well because each specification file was closely tied to the architecture of a particular

[6] Productivity on a pair of new applications over a two month period averaged 1000 lines of ABL code ready for integration testing per programmer per hour.

template.[7] Thus, I think a template-based approach for generating ABL, while more productive than hand-coding, provides a limited ability to adapt to future architectural and general functional requirements. Without regenerability, it is even more limited in its potential.

If not template generation, then what?

Is there an alternative? I would like to suggest the Model-Based Development (MBD) provides a much better alternative. In MBD one creates a UML model of the application which is "computationally independent". This means that the model is not only independent of the specific implementation, but it is independent of the platform and any other computational assumptions. The model is going to be the same regardless of whether one is creating Java to run on J2EE or ABL with an OERA structure.

That this is possible may seem surprising to some, but it is standard practice in some problem domains. In fact, only a tiny part of UML is required to completely specify the application:

- The Component Diagram to specify the structure and interrelationships of the parts,
- The Class Diagram to specify the details of each class,
- The State Chart to specify dynamic behavior, and
- Abstract Action Language (AAL) to specify algorithms and similar logic.[8]

Other diagrams may be useful in deriving and documenting requirements or to provide a different view, but only these diagrams are required to fully specify the system. This model then translated into the desired implementation language. The translator I am using creates all language and architectural structures during the translation process.

The implication of this approach is that one gains significant productivity during initial development, but more importantly, dramatically increased productivity when revising requirements in much the same way that these gains have recently been reported for Corticon. If one is happy with the current implementation design, all one has to do to implement a change is to make simple changes in a visual UML tool and the launch the translator to produce new code. If one changes one's mind about the implementation design, then one has to change the translation templates, but the work required is a tiny fraction of the work required to manually retrofit the application code. With MBD, one makes any desired modification to the translation templates, then runs the translator and has new code end to end. Even major shifts in architecture can be accomplished without altering the model.[9]

Summary

I urge Progress Software to explore the potential for Model-Based Development in ABL since:

- MBD for ABL provides the same kind of productivity advances that Rollbase, Corticon, and OE BPM provide;
- MBD for ABL provides the responsiveness to changed requirements that other model-based system like Corticon and OE BPM achieve;

[7] A lack of regenerability and architectural dependence concerns me to some extent about Rollbase. While we don't yet know what the likely evolution will be of web applications, it seems likely that they will evolve in significant ways and thus it seems likely to require significant redevelopment.

[8] See http://www.cintegrity.com/content/Model-Based-Development-Day-Life for a short presentation on the nature of Model-Based Development.

[9] I should note that it is common to "color" parts of the model. This "coloring" is not a part of the model, per se, but is provided to give hints to the translator about which templates to use for particular parts or which "flavor" of implementation to use. A major change in architecture may thus require a certain amount of recoloring the model to get optimum results, but the model itself will remain unchanged. Any such requirement is still a tiny effort compared to migrating the application by other means.

- MBD for ABL produces a coherent message since both Corticon and OE BPM are touted for being model-based[10]; and
- Providing MBD for ABL will make ABL “sexier”.

Not only will MBD for ABL improve OpenEdge and Pacific for developing new applications, but MBD can provide reduced and improved transformation of legacy ABL applications though further development of code-to-model tools[11].

I will be speaking on MBD and showing some models and their transformation to ABL at the PUG Challenge EMEA in November 2013.

[10] Also see two older white papers about rapid change and ABL, http://www.cintegrity.com/content/Rapid-Business-Change-and-ABL-Productivity, and specifically on a path for ABL translation, http://www.cintegrity.com/content/Path-Model-Code-Translation-ABL

[11] See http://www.oehive.org/ABL2UML for tools that provide the foundation for this approach.