Development Strategy for 2013 - 2015

October 12th, 2012

Introduction

The last few years have seen a significant increase in the size of the development team at Dyalog. Most of the recent arrivals now have sufficient experience in the art of working within the team to be truly productive, and we have been able to improve our "processes" to adapt to the increased size of the team.

There is always too much to do, and recent changes in the strategies of companies whose work we build upon (in particular Microsoft), and the types of hardware and software that are becoming popular, have led to some of our recent work being a little tentative. We have explored a number of ways forward on a number of fronts, such as the RIDE and APL#, the experimental MiServer web framework, secure TCP communications, parallelization of scalar dyadic functions, models of "parallel each" implemented in APL and an internal "new parser" project. With a few exceptions, most of these projects have been peformed without fully committing our energies due to fundamental uncertainties about what is feasible and/or desirable – while we continued working on overall quality and helping our customers solve short-term issues.

As 2012 draws to a close, the development strategies of Microsoft, Google and Apple are feeling a bit more predictable (this has proven to be an illusion in the past, of course). We have had time to rethink some of the research we have been doing over the last few years, and are able to put together a plan that should provide a sustained focus for two to three years. This document is being distributed to all participants of Dyalog'12. On the last page there is a voting form which allows you to inform you of our priorities. We invite you to fill in this form and hand it in at the front desk, ideally by Wednesday, so that we can examine the results and use them in the last sessions on Thursday.

A pair of supporting documents provide additional details of two key avenues of research and development mentioned in this document. These can be found in the "Road Map" folder on the conference web page:

- Isolates, Futures and the Parallel Operator
- Extremely Portable Dyalog APL

Focus Areas

The following six focus areas are proposed:

1. Performance
2. Parallel Computing
3. Portability
4. Expressive Power and Functional Notation
5. The Needs of New Users
6. Application Frameworks

The remainder of this document will describe three key targets in each of the above areas.

1. Performance

Increasing the speed of the most widely used primitive and system functions as we learn about new usage patterns and improved algorithms will always be a high priority. We want APL to have a reputation for speed, and Dyalog to be known as the fastest APL available.

a) Develop the experimental "new parser" technology to the point where it can provide a factor of ~2 improvement for "functions with no side-effects, on small arguments" – using customer utility functions to drive the order of implementation of new features[1].
b) Improve memory management, for example: Repeat the experiment of separating pocket fields out into individual bytes (allowing rank up to 255), to verify whether this will give a significant performance boost. Look into results of modern research into more sophisticated "generational" memory managers. Look into NaN boxing, "structs", and other extensions to the language which might increase the efficiency of memory management when running applications.
c) Continue to enhance a handful of primitive operations with every release, forevermore.

2. Parallel Computing

Harnessing parallel hardware could be viewed as a sub-topic under "Performance" – but is a sufficiently large problem space to deserve a separate set of goals.

a) Implement "isolates" (previously known as "asynchronous namespaces"), which are separate workspaces managed by independent interpreters - but with the ability to "dot" into each other. Allow asynchronous function calls to generate arrays of "futures" to simplify synchronization of multi-threaded applications – and a new "parallel" operator. See the separate document for more details.
b) Work on establishing "basic research" partnerships with players like Robert Bernecky, Sven-Bodo Scholz at Heriot-Watt University, the "HIPERFIT" project at the University of Copenhagen, and any other parties who are interested in advancing the field of compiling array operations for massively parallel execution, where APL code might run several orders of magnitude faster on GPGPUs or other parallel hardware.
c) As part of the "new parser" project, look at performing "parallel loop folding" on small trees of expressions involving large arguments – with an aim to improving cache utilization (e.g. A+B×C).

3. Portability

It has become clear that smartphones, tablets and embedded devices, typically based on Linux-like operating systems and ARM processors, are going to play a very important role in the next decade. In the short term, it may not always make sense to actually run APL on the devices, and web-based architectures which allow APL developers to "project" user interfaces to the new devices, may be more important (see the "Application Frameworks" section). However, the iPod, Android- and Windows-based tablet computers - and very small machines similar to the Raspberry Pi - will see widespread use. It is time for us to invest in making sure that good APL interpreters will there to meet the demand when it arises.

We should completely separate the core interpreter from the development environment, so that we do not always need to produce a "rich" IDE on all the machines where APL is going to run. This will make many of the new interpreters easier to build and more immediately useful.

[1] All customers are invited to submit their most CPU-consuming utility functions to help guide this research.

In addition, in order to make our users sufficiently productive when they start building applications on platforms where we only provide a simple interpreter without its own GUI tools (etc), we will have to provide "Bridge" components that will hook in with the most popular object oriented frameworks available on each platform (Java, WinRT, Cocoa, etc).

a) Produce a cut-down build of the interpreter that can run under WinRT (Metro[2]), Android, iOS and OS/X. Deliver support for WinRT, Android and MacOS (and possibly iOS) based on regular Dyalog APL (rather than APL#, which is henceforth an experimental platform for language design).
b) Produce WinRT and Java bridges (for Windows 8, Android and Mac/OSX), and a Cocoa bridge (for iOS). To accompany the new bridges, develop mechanisms in the interpreter to use bridges to implement "data binding", to allow the APL workspace to share variables with external database or GUI components.
c) Abandon the current RIDE prototype which has been implemented on Silverlight, and produce "native" versions of the RIDE for WinRT, OSX, Android, Linux and iOS (a Java-based RIDE might give us a cross-platform solution for some of these platforms). Note that, once the first couple of RIDEs are built, the remaining projects are prime candidates for both out-sourcing and open-sourcing (for example, we should build our own open-source RIDE in APL).

See the separate document titled "Extremely Portable Dyalog APL" for more details.

4. Expressive Power and Functional Notation

If we are to interest new generations of developers in APL, it is very important that (Dyalog) APL is seen to be moving towards functional and parallel high ground. There are a number of language constructs like function trains and operators like *rank* and *key*, which at the same time (and this is no accident) increase the expressive power that the APL user can wield, and move us in the direction of more easily optimizable and parallelizable code.

a) Implement Forks.
b) Implement Rank and Key (and perhaps Merge and Tesselate).
c) Continue to do research into additional language enhancements such as: Closures, Rational Numbers and other higher precision numeric representations (including 64-bit integers and the grand unified theory of exact versus inexact numbers).

5. The Needs of New Users

For many years, Dyalog APL has been the APL of choice for the "professional" APL developer. Many of the companies who actively use APL to develop software for sale or run APL-based services have migrated to Dyalog APL over the past two decades. Our focus on professionals was the correct choice at a time when the big players (IBM, IPSA, STSC) were still marketing APL to the general public.

Today, Dyalog is the biggest player in the APL market, and it time for us to use a portion of the revenues resulting from our growth to attract the next generation of users.

a) Provide integrated, animate-able and editable graphics in the session (IDE and RIDE).
b) Partner with universities to perform a "usability study" of how today's students react to Dyalog APL, and take steps to deal with any issues that arise.

[2] Apparently, this should now be referred to as "Windows 8 Modern UI".

c) Ensure that R, MatLab, LaPack and other relevant tools that new users view as important building blocks for analytical applications can easily be integrated with Dyalog APL. Possibly also (?) provide a basic statistical library written in APL.

6. Application Frameworks

With every passing year, the requirements for integration of APL-based applications are growing. The bar keeps being raised with respect to producing "slick" user interfaces. On the Windows desktop, our own "⎕WC" GUI has struck an excellent balance between ease-of-use for the non-technical user and providing the professional developer with a tool capable of producing very decent-looking applications. However, we have to accept that it will not be possible for us to repeat this exercise on the wide variety of user interface platforms that our users will be faced with in the near future. We need to seek out tools like JQuery and other cross-platform frameworks, and find ways to put them at the fingertips of our users.

a) Develop a general-purpose "applet framework" or "process manager" which is able to run as Windows and Linux services, with Logging (and "Management" interfaces). This will be able to very quickly launch Dyalog processes on demand (using a pool if necessary), and with time provide built-in Load Balancing capabilities
b) Develop the MiServer and SAWS into "industrial strength" application frameworks, available on all platforms where Dyalog APL will run, completely stand-alone or integrated with the "applet framework" mentioned above.
c) Develop other common templates or "patterns" that will provide a big "leg up" to users trying to make the transition from using APL as a tool of thought to Dyalog APL as a delivery mechanism. This should include: Simple Database Patterns based on Component Files, Simple GUI and SQL Samples illustrating "best practices". Error and Application Logging tools – all the components required to build a simple web or desktop application.

Conclusion

Even with our increased resources, it is unlikely that we can complete all of the projects described above in 2-3 years. However, we should be able to make very significant progress on most of them.

Please help us prioritize the work by filling in the form on the last page of this document and handing it in at the conference desk by lunch-time on Wednesday!

Thanks in advance,

Morten Kromberg
CTO, Dyalog Ltd.

Road Map Feedback Form

Please pick the five (5) areas of research that are most important to you, without considering the amount of work that you imagine is involved in achieving the stated goals. Where relevant, circle the alternatives in the main column (such as target platform) that are most valuable to you.

	Area of research	Priority
1a	Significantly improve speed of small functions	
1b	Improve overall performance of memory manager	
1c	Improve performance of primitive functions case by case	
2a	Isolates and Futures to support user-controlled parallelism	
2b	Basic research into massively “fine-grained” data-parallelism	
2c	Improve speed of chains of parallel computations, e.g. A+B×C	
3a	Dyalog APL on WinRT/Metro, Android, Mac OS/X	
3b	Bridges to Java, WinRT, Cocoa	
3c	RIDE on Windows Desktop, WinRT/Metro, iOS, Android, OSX, Linux	
4a	Forks	
4b	Rank, Key	
4c	Closures, Rational Numbers	
5a	Graphics in the session	
5b	Usability study / revision of IDE user interface	
5c	Interfaces to R, MatLab, LaPack and other tools. Statistical Library for APL.	
6a	APL Applet / Process Manager	
6b	Industrial Strength MiServer and SAWS	
6c	Tools and Patterns	

Section 3: New Platforms

If you would consider purchasing Dyalog APL for one of the new platforms, please write a few words about your potential application and the framework components that it will need.

Other Comments

If you feel that there are areas of potential research and development that we have completely ignored, or have any other input, please enter it below: