DELTA Redevelopment Options Paper

David Baird

Atlas of Living Australia

August, 2010

Version 1.0

Table of Contents

Introduction

This paper presents an outline of the technical options regarding a possible port or redevelopment of the DELTA (DEscription Language for TAxonomy) system. It does not attempt to describe the goals or detailed functions of DELTA, but rather presents a technical overview of the key programs that comprise DELTA, a summary of the issues regarding running DELTA in its current state on contemporary operating systems, and recommends a number of options regarding moving DELTA forward in order to support new and existing users of the DELTA system.

DELTA overview

A detailed introduction to the DELTA system can be found at http://delta-intkey.com/www/overview.htm. For the purposes of this paper DELTA is considered a suite of computer programs designed to record and process taxonomic descriptions, and the DELTA file format. The DELTA file format has been ratified as a technical specification by Biodiversity Information Standards (TDWG)[1].

The core DELTA programs (CONFOR, DELFOR, KEY and others) were originally written in FORTRAN during the 70's and 80's. Additional DELTA components (Delta Editor, IntKey and Intimate), where developed during the late 90's up until the early 00's, at which time CSIRO ceased funding for the development of DELTA. The Delta Editor, IntKey and Intimate where developed in C++, but because of the use of a proprietary object library called OWL, which is now defunct and not compatible with 64 bit Windows, the C++ code is not portable to other platforms (i.e. tied specifically to the Microsoft Windows platform).

Below is a brief simplified description of each of the core components under consideration by this document:

- CONFOR (Format Conversion): CONFOR is used to manipulate the various types of data files used by the DELTA system. CONFOR translates input 'directives' and data files into output files in a variety of formats. Such formats include Natural Language descriptions in text, RTF and HTML, INTKEY format, NEXUS and others;
- DELFOR (Data Maintenance): DELFOR is used to reformat data and directives. Most of the functions of DELFOR have been included in the Delta Editor;
- KEY (Key generation): KEY produces identification keys from DELTA input files, and includes provisions for typesetting and formatting;
- The Delta Editor: A GUI tool used to generate and manipulate DELTA data sets. The Delta Editor can import and output standard DELTA data and directives files, and also maintains its own internal (binary) data format. The primary nouns manipulated by the Delta Editor are Taxa, Characters (or features) and Character States (or simply States). The delta editor uses CONFOR internally to produce data files suitable for IntKey. It was envisaged that eventually all the functions of CONFOR would reside in the editor, but development was halted before this could be realised;
- IntKey (Interactive Identification): IntKey is an interactive diagnostic/identification tool that uses specially formatted DELTA data to lead a user through the optimum number of

[1] http://www.tdwg.org/standards/

steps (characters) in order to identify a specimen. IntKey can also produce various reports, including diagnoses, descriptions and the differences and similarities between taxa.

Problems

16 bit vs. 32 bit vs. 64 bit

DELTA, as a software suite, has been under development for over 30 years, and now includes a mix of technologies ranging from Fortran 77 augmented with assembly code to a somewhat proprietary C++ dialect with dependencies on deprecated libraries.

The current distributions of the FORTRAN based programs (CONFOR, DELFOR and KEY) have been compiled for 16 bit windows, and although the programs will still execute on 32-bit Windows, they will not execute on a 64 bit Windows installation. This is becoming more of a problem as 64 bit Windows installations become ubiquitous. Intel has been producing commodity 64 bit processors since 2004[2], and the majority of Windows based PCs manufactured since 2006 comprise a 64 bit processor, even if the majority of these PC's have 32 bit versions of Windows installed. The popularity of Windows 7, however, is driving 64 bit uptake, with around 46% of Windows 7 installations being 64 bit[3]. In addition, Microsoft is signalling that future support for 32 bit processors may be dropped altogether in future versions of Windows. Windows Server 2008 R2, for example, only supports the x64 architecture[4].

Of the C++ based programs, IntKey and Intimate both launch correctly on 32 and 64 bit editions of Windows; however the Delta Editor will not start at all under any version of Windows 7, even in compatibility mode.

Program	Windows 7 (x86)	Windows 7 (x64)
CONFOR	OK	Fails to start
KEY	OK	Fails to start
DELFOR	OK	Fails to start
IntKey	OK	OK
Delta Editor	Fails to Start	Fails to Start
Intimate	OK	OK

DELTA compatibility with 32 bit (x86) and 64 bit (x64) Windows 7[5].

To summarize, the problems are twofold:

- The FORTRAN based programs are not supported under Windows 7 x64
- The Delta Editor will not execute under any edition of Windows 7.

Development Platform

The FORTRAN programs (CONFOR, DELFOR and KEY) have been compiled under Microsoft's Fortran

[2] http://en.wikipedia.org/wiki/X86-64
[3] http://windowsteamblog.com/windows/b/bloggingwindows/archive/2010/07/08/64-bit-momentum-surges-with-windows-7.aspx
[4] http://www.microsoft.com/windowsserver2008/en/us/system-requirements.aspx
[5] As tested by the author on either a physical machine or in a virtual machine. Also note that this only tested if each of the programs would launch successfully. There may exist operational issues with not identified by these tests.

Compiler (Microsoft FORTRAN 5.1) and utilizes a Microsoft library called QuickWin[6] in order to operate within Windows (by basically wrapping the command line in a surrogate window). This version of FORTRAN can only produce 16 bit executables. In addition the FORTRAN programs make use of routines written in x86 assembler, mainly concerned with the vagaries of the x86 DOS environment. The assembler code is compiled to linkable object code using the Microsoft Assembler (MASM). Both of these development environments where last published in the early 90's, and are now obsolete and no longer supported by Microsoft.

The Delta Editor, IntKey and Intimate were all written using Borland C++ (version 5.0x, last published in 1997), which is no longer available or supported, and, in fact, have been superseded many times over by various other products[7].

Apart from the dependence on an obsolete development environment, the Delta Editor, IntKey and Intimate each have a heavy reliance on Borland's proprietary Object Windows Library (OWL) framework, which at the time was a viable competitor to, and improvement upon, Microsoft's Foundation Classes (MFC). OWL was deprecated in 1999, being replaced by Borland's competing VCL framework, and is no longer supported[8].

Consequences of the nature of the legacy development environments include:

- the difficulty to acquire the correct tools to build each of the DELTA programs, should revisions be required;
- The difficulty in procuring programming resources with the requisite skills and experience to undertake such work.

Goals

The goals of any port or redevelopment of the DELTA system would include:

- A consolidation of the code base to a single contemporary language or development environment;
- Remove any impediments to DELTA running on 64 bit Windows;
- Address any outstanding bugs or issues that may have been identified in the current code base;
- To make the source code freely available under an Open Source licensing scheme;

A secondary goal is:

- The ability of DELTA to be executed on multiple platforms (either via targeted compilation, or through write-once, deploy anywhere technologies) – such platforms would include Windows, Linux and Mac OS X;

Opportunities

There exists within the scope of a port or redevelopment of DELTA the possibility to take opportunities to improve the overall usability and maintainability of the DELTA suite. Such

[6] http://en.wikipedia.org/wiki/QuickWin
[7] http://en.wikipedia.org/wiki/Borland_C%2B%2B
[8] http://en.wikipedia.org/wiki/Object_Windows_Library

opportunities include:

- A common language/platform would allow for code reuse. For example, in the current DELTA there exists code in both FORTRAN and C++ that is concerned with the parsing and processing of DELTA directive and data files. By using a single development platform such code could be written once and shared as a library across all consuming applications;
- Improvements to the UI design and paradigm;
- Code simplification and increased "Signal to Noise" ratio. For example, in the existing C++ code there is a lot of code that is not directly related to the business concern of DELTA, but rather exists as an artefact of the language and operating environment itself (for example, memory allocation, Windows message handling, primitive string handling etc.). The selection of a development environment that sufficiently abstracted from the physical environment allows for a clearer articulation of the solution within the problem domain;
- The use of included and open source libraries. As an example, the existing C++ projects include explicit code for the decoding and rendering of JPEG, GIF and Bitmap images. This functionality is now widely available as either a built in library, or via a third party open source library.

Limitations and restrictions

- ALA is only funded until July 2012
- Existing features and use cases must be catered for.

Approach Options

Option 1 - “Minimalist Port”

Simply translate existing code from source language to destination language.

Pros

- Might be viewed as the cheapest way forward (no interpretation of intent required, just a mechanical translation), BUT…

Cons

- Might work ok for the C++ programs, but not feasible for FORTRAN code (application architecture and design not compatible with modern development environments);
- Some degree of interpretation is unavoidable;
- A direct translation of obsolete design is not optimal nor desirable;
- No real scope for improvement;
- High likelihood that the resulting code will be difficult to understand and maintain.

Estimated development effort: 10 months

Option 2 - Redesign from scratch

This option has been put here mainly for completeness. It cannot be seriously considered because of time and budget constraints.

- No budget or resources for this scale of undertaking;
- An optimal redesign may necessarily break compatibility with existing data sets;

Estimated development effort: 2+ years

Option 3 – Reimplementation or port using existing programs as a specification

Pros

- Can use existing programs as a baseline for regression testing
- Retain scope to translate critical algorithms verbatim (i.e. the “Best” algorithm)
- Allow for platform appropriate design (i.e. not trying to shoehorn FORTRAN code into an object oriented paradigm)
- Allows the developer to operate efficiently within the targeted development environment

Cons

- Requires that the developer can understand the intent of the original code, and duplicate without introducing subtle errors

Estimated development effort: Approximately 12 months (See accompanying project plan)

Technology Options

Java

- Write once, run anywhere, depending on JVM availability
- Major language, easy to find programming resources
- Lacking some popular language features, and is relatively verbose;
- Fast – optimized, JIT compiled
- Managed environment (garbage collected)
- Strong free tool and library support
- Can use Groovy (or other JVM language) if desired

Swing

- UI elements rendered in Java, so can be bit slow and clunky
- Some support for UI designer tools (e.g. Matisse and Instantiations Swing Designer)
- Will run just about anywhere, but may not look exactly the same everywhere;
- Skin-able via "look and feel" themes

SWT

- Supported under the majority of major platforms (restricted to the same set as is supported by Eclipse (Windows, Linux and Mac OS X is supported, however);
- Uses native widgets, so is performant;
- Not as well understood as Swing
- Lack of designer tools for GUI layout

Groovy

Consideration has also been given to alternate languages that run on the Java Virtual Machine. One such language, described by JSR241, is Groovy:

- Executes on the JVM, so gains the benefits of that platform (managed environment etc.);
- Better language constructs and features (closures, object literals, concise and freer syntax);
- Limited tool support;
- Poor GUI designer options;
- Dynamically or statically compiled, providing flexibility;
- Although an official language for the JVM, it is still somewhat of a fringe language, ranking 44th on the TIOBE index[9].

[9] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

.NET

- Multiple language support (C#, VB.NET, IronPython etc.)
- Fast, JIT compiled, comparable to or better than Java;
- Managed environment, garbage collection etc.;
- C#, in particular, offers language features resulting in more concise code, such as closures, automatic property implementations, default arguments etc.;
- There are two variants to choose from, however…

Microsoft .NET (Windows Only)

- Excellent support for UI design (Windows Forms or WPF);
- Good free tool support (Visual Studio Express);
- Heaps of community and library support;
- Potentially use WPF for richer UI experience.

Mono (Windows/Linux/Mac)

- GUI is more or less restricted to using a library such as GTK (see Note regarding .NET below);
- Smaller library of components and smaller community;
- Is disliked and distrusted by some elements of the free software movement for fear of Microsoft enforcing patent claims.

Other

Python

- Interpreters available for most platforms (Windows, Linux and Mac OS X);
- Strong language community;
- Has bindings for GTK, SWT, Qt and many others, although the maturity of such frameworks is unknown;
- Is generally interpreted to maintain cross platform portability, which may have performance implications.

C/C++

- Typically acknowledged as fast executing, but arguably slower to develop in (fiddly, low level, error prone etc.)
- Not managed, nor abstracted from hardware, requires code to manage memory etc.
- Large community and support base. Many libraries, although care must be taken to ensure platform neutrality, if desired;
- Support for cross platform UI bindings such as GTK, Qt etc.
- Good free tool support

Recommendations

Approach Option 3 - Reimplementation/Port using existing programs as a specification

The rationale for this option is that it allows for the migration of the code to a contemporary development platform whilst maintaining the feature set of the Delta suite. The existing software and its source code can be used as a specification, and as a baseline for regression testing. Efficiency of development is improved through judiciously selecting which functions to port directly (such as the BEST algorithm), and which functions can be redeveloped using modern programming techniques and paradigms. A degree of interpretation by the developer is unavoidable when porting code to one language to another, and by allowing some flexibility (i.e. not being restricted to a line for line translation of source code) permits the use of techniques and libraries that were previously untenable due to the restrictions of the original target environments (memory, CPU speed, etc.), but now allow for far greater programmer efficiency (in terms of development time).

Technology Option - Java and Swing

The rationale for using Java as the target development platform is:

- Automatically cross platform to any operating environment for which there is a conformant Java Runtime Environment available. This includes Windows, Linux and Mac OS X;
- Managed environment with automatic resource reclamation (garbage collection) which removes the need for explicit code to manage such things;
- Abstracted from hardware so that the proportion of code that is concerned with the problem domain outweighs code required as an artefact of the language and platform;
- Performance. Just In Time (JIT) compilation to native code, and other advances with optimizing Java compilers have led to Java being compared favourably to other compiled languages(Including C/C++[10][11]) in most areas;
- Swing (a graphical user interface library) is a part of the Java runtime library, and is available anywhere where Java is, thus simplifying packaging and distribution. There are a number of 3rd party tools available to aid the design of user interfaces, although due to the relatively simple design of the Delta Editor and IntKey design, the user interface can most probably be built 'by hand' just as easily;
- Relationship with ALA and Identify Life. Java is the language of choice for the ALA, and it may prove beneficial to have the Delta components developed under this project be reusable by other ALA projects;
- Java is ubiquitous, and is one of the most popular contemporary development platforms[12]. The ability find resources able to contribute to a Java version of Delta post ALA involvement will be aided by this.

[10] http://www.idiom.com/~zilla/Computer/javaCbenchmark.html

[11] http://blog.cfelde.com/2010/06/c-vs-java-performance/

[12] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

A Note regarding .NET

If a cross platform solution were not being considered, and DELTA was to be a Windows only application, then .NET/C# would be the recommended technology option, for the following reasons:

- In my experience with both Java/Swing and C#/WinForms, Microsoft's Visual Studio and its Windows Form Designer offer the most productive GUI development environment for Windows;
- C# boasts language features that allow for powerful yet concise code, such as lambda/ closures, automatic property implementation, default arguments etc., which can greatly reduce the amount of boiler plate code required;
- In 2006 Microsoft introduced an alternative to Windows Forms called the Windows Presentation Foundation (WPF), which can interoperate with Windows Forms and offers a powerful and flexible approach to design rich, media focussed user interfaces.

I estimate that these features would offer C#/.Net a 5-10% advantage, in terms of development effort required, over Java and Swing. As stated previously, however, a secondary goal of any redevelopment effort would be to make DELTA cross platform. In the context of cross platform development the advantages gained are nullified by the follow factors:

- The use of Mono is required for cross platform .Net; Mono lags behind Microsoft.NET in terms of compliance to current language and runtime specifications;
- Visual Studio is only available on Windows, and MonoDevelop, its nearest competitor in the cross platform space, is lacking features and functionality in comparison, especially with regard to GUI design tools. The support for WinForms under other platforms is sketchy[13], with incompatibilities being noted, and other competing toolkits, such as GTK# adopt a lowest common denominator approach, which may result in some components, such as a Grid control, being unavailable out of the box;
- The Mono project has no current plans to support WPF;
- The currently stable version of Mono does not yet support version 4.0 of the C# specification, meaning that some language features would be unavailable.

These reasons, coupled with the controversy regarding Mono and patents held by Microsoft created by the Free Software Foundation[14] make Java/Swing a safer recommendation for a cross platform development of DELTA.

[13] http://www.mono-project.com/Guide:_Porting_Winforms_Applications
[14] http://www.fsf.org/news/dont-depend-on-mono

Appendix 1 - DELTA Codebase Analysis

"Lines of code" (LOC) analysis was performed using the CLOC program (sourced from http://sourceforge.net/projects/cloc/) against a copy of the source code of the various DELTA components. LOC measurements ignore blank lines and comments. LOC should only be used as a rough guide of program complexity, and the following caveats should be considered:

- A proportion of the code counted will be concerned with artefacts of the language used (e.g. explicit memory allocation/de-allocation, and "manual" GUI construction in the case of C++) which may not be applicable to a redevelopment in another development platform;
- More modern languages possess features that allow for more concise expressions of intent, and also include (as libraries) features that have had to have been explicitly coded in the original source (e.g. image decoding, collection classes etc.);
- The Lines of Code count may also include files present in source directories that are not included in the actual set of files required for the build of the project. Because of the difficulty in locating a working development environment for these projects, however, it is assumed that all source files contribute to the build.

CONFOR

Language	**Lines of Code**
FORTRAN 77	21852
Assembly	910

KEY

Language	**Lines of Code**
Fortran 77	7047
Assembly	414

Delta Editor

Language	**Lines of Code**
C++[15]	54206
C/C++ Header	16065

Intimate

Language	**Lines of Code**
C++	24506
C/C++ Header	5985

IntKey

Language	**Lines of Code**
C++	59643
C/C++ Header	11636

[15] Excluding source for two third party libraries – Ultimate Grid and a JPEG decoding library