The Development History of Quark

Bo Ilic
last modified: November 15, 2007

1. Introduction

This is a short history of the Quark project, including the CAL language, since its inception in the year 2000 through to the release of Open Quark 1.7.1 on November 15, 2007. Quark is interesting as an example of a large scale research project done within the setting of a medium-sized commercial software company (initially Seagate Software, then Crystal Decisions, and finally Business Objects).

This paper is a work-in-progress and is in its early stages. Its main current content is a chronology of the tangible features, and an estimate of the development effort involved in the creation of Quark so far. A more complete discussion as to the motivating ideas, how they have changed with time, what ideas are new contributions and what are derived from external sources and speculations on the future directions of the project are not yet included. There is some more discussion on these points in the Wikipedia article: http://en.wikipedia.org/wiki/Quark_Framework.

CAL is a strongly-typed modern functional programming language for the Java virtual machine. It is hosted at http://labs.businessobjects.com/cal/ with a discussion forum at http://groups.google.com/group/cal_language.

Similar projects within this period have been for the languages F# (http://research.microsoft.com/fsharp/fsharp.aspx), done at Microsoft Research for the .NET platform, and Scala (http://www.scala-lang.org/), at the Ecole Polytechnique Fédérale de Lausanne for the JVM.

Some of the highlights that distinguish CAL/Quark from these other efforts:

1. CAL is lazy. An efficient implementation of a lazy language on the JVM is part of our research contribution. CAL is the only full-featured high-performance lazy language on a dynamic platform. In fact, as of this date, CAL is also the fastest alternative JVM language http://shootout.alioth.debian.org/gp4/benchmark.php?test=all&lang=all.

2. The Gem Cutter is a novel and highly productive visual programming language for creating CAL code. It also is illustrative of CAL's support for functional meta-programming. The ideas behind the Gem Cutter are discussed more in the paper The Gem Cutter - A Graphical Tool for Creating Functions in the Strongly-typed Lazy Functional Language CAL. CAL has been designed from the outset to support mixed language programming where users can create applications involving both Java and CAL, as well as functional meta-programming, where users can create, compile and run CAL code from a running Java application.

3. While not completely pure (in the sense of Haskell), CAL can certainly evolve in that way. It is the most pure of the current crop of "practical" languages for a dynamic platform. It is also the main one with a strong Haskell influence as opposed to a strong ML influence.

4. The evolution of CAL has, almost from its outset, been guided by real developers using it for the purpose of creating commercial business applications. This history focuses on Open Quark. However, much of the history of Quark as a project revolves around those who made use of Quark in their work at Business Objects, and contributed significant input as users of the system. These individuals helped keep Quark focused on being a practical and useful language for creating real applications.

2. Quark team

(in order of appearance)
present = Open Quark 1.7.1, November 15, 2007

Luke Evans, head of Research, 1/2000 – present, 6 months
Bo Ilic, lead of Quark project, 5/2000 – present, 81 months
Michael Cheng, coop, 1/2001 – 8/2001, 8 months

Edward Lam, 5/2001 – present, 78 months
Raymond Cypher, 11/2001 – 4/2007, 65 months
Steve Norton, coop, 1/2002 – 8/2002, 8 months

Ken Wong, coop, 9/2002 – 4/2003, 8 months
Frank Worsley, coop, 5/2003 – 12/2003, 8 months
Alan Macek, intern, 6/2003 – 8/2003, 3 months

Iulian Radu, coop, 1/2004 – 8/2004, 8 months
Peter Cardwell, coop, 9/2004 – 4/2005, 8 months
Joseph Wong, 1/2005 – present, 34 months

James Wright, 2/2005 – 4/2006, 14 months
Greg McClement, 4/2005 – present, 31 months
Magnus Byne, 9/2006 – present, 14 months

Neil Corkum, coop, 9/2006 – 12/2006, 8 months
Jennifer Chen, coop, 5/2007 – present, 6 months
Andrew Eisenberg, intern, 6/2007 – 8/2007, 3 months

Notes on the people:
Luke has actively put in approximately 6 months of development on Quark. The rest of the time, he has been involved primarily from the point of view of managing the project as part of the activities of Research.
Bo initially divided his time between Quark and his previous duties as a team lead in Crystal Reports. This subtracts about 9 months from his time on the project.
Everyone else essentially worked full-time on Quark during the periods listed above.
There were 18 people who worked primarily on Quark, 10 of the 18 being coop students (undergraduate students on a work term to gain industry experience), or interns (graduates taking a break in their studies before returning to school). The energy and enthusiasm of our coops and interns have contributed much, both in terms of features, as well as adding spark and novelty to our days.

Total work = 391 person-months = 32.6 person-years

Roles:
(i.e. primary areas of work)
Luke- project vision, interaction with other Research and Business Objects projects
Bo- CAL compiler, project architecture, team management, CAL libraries
Edward- Gem Cutter, CAL Eclipse Plug-in, Builds, CAL services
Ray- CAL runtime (tim, g and lecc)
Alan- .NET version of CAL using IKVM
Joseph- CAL services, CALDoc
James- CAL services, CAL User's Guide
Greg- CAL global optimizer, CAL Eclipse Plug-in
Magnus- CAL services, shootout benchmarks
Andrew- Embedded CAL

All coop students worked primarily on the Gem Cutter and CAL services (such as refactoring support)

3. Acknowledgements

As mentioned in the introduction, those who have used Quark and CAL for the purposes of developing applications or evaluating the technology have been of great help to the project. We'd like to acknowledge (chronologically within a grouping) the following people. For those we've missed, we apologize- please let us know and we'll fix.

Research team #1:
Trevor Daw, Richard Webster, David Mosimann, Sandra Trace, Kevin Sit, Dawn Baikie, Tom Haggie, Robin Salkeld. Andrew Casey (coop), Gregoire Cacheux, Daniel Ferstay. Alex Bradley (coop), Malcolm Sharpe (coop)

Research team #2:
Rick Cameron, Louis Wald, Doug Janzen, Chris Balavessov, Davor Cubranic

Engineering team #1:
Krzysztof Bacalski, Paul Bartlett, Adam Redgewell, Alex James, Paul Woods

Engineering team #2:
Mark Allerton, Mone Hsieh, Roland Urbanek

External:
We'd also like to acknowledge all the posters on the CAL Language Forum and thank them for their interesting questions and bug reports.

4. Chronology:

This is a listing of the various main events in the history of Quark. For dates prior to the release of Open Quark, this chronology was created by going over the check-in notes and the dates track when a feature was checked in to a certain degree of completion, rather than when work on it first started. For dates after the release of Open Quark, the chronology is grouped by release, and indeed the content is based on the release notes. In both cases, the name is that of the primary person who implemented the feature- often there were others involved in terms of generating the idea for the feature, aspects of the implementation, etc.

Note some of the history of the various optimizations in the CAL runtime is covered in the CAL Benchmarking document included with Open Quark. The comment at the start of CAL's main ANTLR grammar file, CAL.g, is a good short place to see when various language features were introduced into CAL.

Areas in which there has been a gradual evolution over time have received light focus here, even though much effort was put into them. Examples of this are with the CAL libraries, the Value Entry system in the Gem Cutter, and various performance optimizations (in this last case the CAL Benchmarking doc provides more details).

Year 2000:

March:

- (Luke) creation of a core functional language in Java 1.2 following Simon Peyton Jones and David Lester's book "Implementing functional languages: a tutorial". This is a non-type checked language, with local variables only (no local functions or lambda

expressions), data types based on tag and arity integers and no syntactic sugar. It used the TIM machine as its interpreter. The parser was generated using ANTLR.

August:

- (Bo) a core functional language in Haskell using the Hugs interpreter and making use of the graph-reduction machine.
- (Bo) conversion of CAL's ANTLR grammar to generate a parse tree instead of using actions.
- (Luke) initial version of the Gem Cutter. Gem arity is based on lexical arguments and the table-top is untyped. Function Gems and Code Gems make their first appearance.

September:

- (Bo) initial type-checker for CAL. Luca Cardelli's paper "Basic Polymorphic Typechecking" was very helpful here.
- (Bo) added case expressions to CAL.

October:

- (Bo) Gem Cutter becomes type aware i.e. arity based on type, connections only succeed if types unify.

Year 2001:

January:

- (Bo) CAL language features: type declarations for top-level functions, string and char literals, data declarations.
- (Michael) initial value entry mechanism in the Gem Cutter. List and Double are available. Calculator as the Double value panel.

February:

- (Bo) CAL language features: local functions, lambda expressions, Int and Double primitive types, list, tuple and unit syntactic sugar.

March:

- (Bo) CAL libraries begin. Implemented many functions from the Crystal Formula Language in CAL, in particular those related to Date and Time.
- (Michael) many new value editors: Tuples, Date, Time, Lists of Tuples, Colour, FileNames, String, Enumeration, Boolean.

April:

- (Bo) first version of type classes. Type class and instance declarations are built-in (i.e. only a fixed set). No contrained instances.
- (Michael) constant gems appear. (They are now called Value Gems). Intellicut sensitive to the types of connections appears.

July:

- (Bo) the initial CAL module system. There is only 1 file (prelude.cal), but it can contain multiple modules.

August:

- (Edward) burning of argument inputs appears in the Gem Cutter

October:

- (Edward) collector and emitter gems appear in the Gem Cutter. Note: there is no argument targeting- this is just for representing let variables graphically.

Year 2002:

February:

- (Bo) we start understanding space usage issues better and create constant space versions of length, sum and product, as well as a foldLeftStrict.

March:

- (Steve) deployment metadata for Gems makes its first appearance
- (Ray) work on fixing space issues in the TIM machine. It can't be done!

July:

- (Ray) initial implementation of the g-machine. By August it is our official machine and the TIM is abandoned.
- (Bo) CAL foreign functions for Java methods, fields and constructors and foreign types (an earlier version a few months prior supported only a few fixed primitive types)
- (Steve) a help system for the Gem Cutter based on JavaHelp
- (Edward) Gem Cutter saving and loading of gem designs.

August:

- (Bo) initial experiments with marshaling functions written in CAL for converting a Java value to a CAL value (e.g. Java lists to CAL lists) and vice-versa.
- (Ray) ICE command line environment for evaluating CAL expressions interactively
- (Luke) basic SQL data access gems

October:

- (Ray) the seq function appears.
- (Bo) comments in CAL change from the Miranda/Haskell style to the Java style

November:

- (Edward) start of CAL services layer to make use of CAL's Java APIs easier

December:

- (Ken) graph arranger to tidy up the presentation of Gem Cutter layout
- (Bo) type class and instance declarations can now be entered in CAL syntax e.g. Enum, Bounded, Outputable, Show etc.
- (Ray) continued work on g-machine to improve performance. Updated schemas to those in Peyton Jones' book without Lester, which are different, and fix some space leaks.

Year 2003:

January:

- (Ken) initial Gem Cutter Scope (the tree view of the table-top)
- (Luke) initial version of yet another runtime machine: LECC. This is a compiled graph reducer, where each core function (roughly) corresponds to a generated Java class.
- (Edward) much refactoring to make the value editor mechanism more robust

February:

- (Bo) default patterns in case expressions and wildcards as pattern bound variables implemented

March:

- (Edward) Previously all CAL modules existed in a single file (prelude.cal). Now multiple files supported, each containing a single module. Separate compilation via workspace declarations.
- (Ray) Changes so that CAL uses Java's String as its basic string type instead of a list of chars as in Haskell.

June:

- (Alan) Work on getting Quark to run on the .NET platform. The approach was to use IKVM and GNU Classpath. This project succeeded in getting a POC version of ICE with its standard libraries at the time running on the .NET platform. Performance was within the ballpark of the Java version.
- (Ray) Enhancements to the lecc machine so that at this point it is able to correctly run the full CAL regression suite (for the g-machine).

July:

- (Bo) added support for type declarations for local functions
- (Bo) added support for constrained instances
- (Edward) initial support for nested lets in the Gem Cutter via the concept of Aggregations and Aggregators. This changed substantially in the future to the current mechanism of Collectors, Emitters with Argument Targeting.
- (Frank) extensively enhanced UI for editing CAL deployment metadata.

September:

- (Bo) added type-safe casts (Typeable type class and TypeRep type) and the dreaded yet powerful unsafeCoerce.

October:

- (Bo) added the concept of implementation visibility for foreign types
- (Edward) Lecc code generation now is implemented via a Java model which can either generate Java source, or generate bytecodes using the BCEL bytecode library.

November:

- (Bo) new client-side input/output mechanism based on the Inputable and Outputable type classes. The previous mechanism involved Java clients being able to directly interact with internal CAL values as they were being reduced to weak-head normal form and involved clients having particular knowledge about the internal representation of CAL values.

December:

- (Bo) new modules for collections in CAL (Map, Set) ported from Daan Leijen's DData.

Year 2004:

January:

- (Edward) concept of the Target Collector in the Gem Cutter. Previously a design was less associated with a particular Gem.

February:

- (Ray) updated the Value Panel mechanism to use compiled IO i.e. input and output policies where marshaling is determined by CAL code.

April:

- (Bo) added support for record types in CAL. Initially tuples were algebraic types (Tuple2, Tuple3, etc), but they eventually were replaced by records with ordinal fields.
- (Ray) added memorization support to CAL (currently part of the Memoize module).

May:

- (Bo) local dependency-order transformations to maximize the polymorphic potential of local functions in CAL.

June:

- (Bo) added strictness annotations for data constructor arguments and function arguments (plinging).
- (Bo) support for record instances, such as Eq {r} in CAL. Adding new record instances involves implementing the record instance functions as primitive functions.
- (Edward) arguments pane view in the Gem Cutter

July:

- (Ray) optimizations to make use of unboxed primitive values in lecc when a function is called all of whose arguments are available as unboxed primitive values.
- (Iulian) added the refactoring command to rename a CAL function.

August:

- (Iulian) added record value entry panels to the Gem Cutter

September:

- (Ray) the lecc runtime compiles tail recursive functions to loops.

October:

- (Bo) foreign data declarations can now work with Java primitive types (such as int and char) as well as the previous Java reference types.
- (Bo) expression type signatures are now supported.

November:

- (Bo) the SourceModel (a low-level Java model of the entire CAL language) appears.
- (Edward) added the ability of the Quark to make use of Gems located in the Crystal Enterprise repository.
- (Peter) CAL internationalization (error messages in CAL are internationalized).

December:

- (Ray) initial changes for making the lecc runtime thread-safe (for evaluations in which each concurrently running thread has its own execution context).

Year 2005:

January:
- (Bo) The CAL bytecode generator (converting CAL source directly to Java bytecodes) now uses ASM for both thread-safety reasons and speed.

February:
- (Bo, Ray) Technique of allowing foreign functions to work with CAL values e.g. as used in the implementation of the sortByExternal function which sorts a CAL list using a CAL comparison function (a -> a -> Ordering) where the underlying work is done by the overload of java.util.Collections.sort that takes a java.util.Comparator.

March:
- (Peter) rename refactoring now updates Gem Cutter designs as well as metadata.
- (James) initial version of the CAL Language User's Guide.

May:
- (Bo) automatic generation of Typeable instances for all types where possible.
- (Bo) Deriving clause to generate certain boilerplate foreign and algebraic type instances.
- (Greg) Port of the Parsec parser library to CAL. This replaced our previously hand-rolled parser libraries.
- (Greg) Added back-quoted operators to CAL to support infix use of CAL functions.

July:
- (Joseph) initial version of CALDoc
- (Edward) syntax for field names of a data constructor in a data declaration (previously the positional syntax of Haskell was used). Ability to use field-name based extraction in case expressions.
- (Ray) deepSeq primitive function.

August:
- (Edward) data constructor field selection syntax in the CAL language
- (Edward) syntax for pattern groups (i.e. multiple data constructors) within a single case alternative.

September:
- (Edward) case expressions for Int and Char types. Earlier case expressions only supported algebraic data types.
- (James) added search functionality for functions, types, etc to ICE and the Gem Cutter.

October:
- (Edward) JFit UI in the Gem Cutter to automate the creation of foreign types and foreign functions corresponding to Java classes.
- (Ray) eager primitive function added.

November:
- (Bo) friend modules and protected scope.
- (Greg) initial work on the CAL Global Optimizer. This functionality of the CAL compiler is written in CAL and does global optimizations of CAL programs involving inlining, fusion, etc.

December:
- (Ray) lifted let variables definitions to separate functions. This is to deal with the 64K limit for the bytecode of a generated method.

Year 2006:

February:
- (Bo) The Debug.showInternal function, which enables inspection of any CAL value, including those not in weak-head normal form, without triggering evaluation.
- (Joseph) Car (CAL archives) support to support bundling of CAL resources similar to that supported by JARs in Java.

March:

- (Bo) added run-time configurable function tracing support to CAL
- (Ray) optimization to the lecc machine to handle unboxed return values.
- (Edward) lecc dynamic runtime mode

May:

- (Edward) Initial work on the CAL Eclipse Plug-in, mainly focused on allowing Eclipse projects to use CAL modules i.e. the CAL builder.
- (Ray) debugging functionality added to ICE

June:

- (Joseph) added the ability for CAL modules to use localized resources and system properties.

July:

- (Ray) Java binding generator- generates a Java class with (among other things) the names of the entities in a CAL module. Helps correctness when doing CAL based meta-programming from Java.

August:

- (Bo) added support for default class methods.

October:

- (Magnus) port of Quickcheck to CAL

Open Quark version 1.2.0-133 (Oct 12, 2006)

- the initial release of the Quark Platform

Open Quark version 1.2.0-135 (Oct 27, 2006)

- the initial release of the CAL Eclipse Plug-in
- (Rick) document on using Quark's Java apis ("Java meets Quark")
- (Neil) the Business Objects GemCutter manual has been extensively expanded to document more features, as well as incorporate revision suggestions
- (Bo) new sample CAL module Cal.Tutorials.CalIntro (a quick tutorial for CAL, showing lots of code and not assuming knowledege of functional programming)
- (Magnus) new sample CAL module Cal.Tutorials.QuickCheck and other improvements to the QuickCheck testing modules in CAL

Open Quark version 1.2.0-140 (Nov 30, 2006)

- (Magnus, Rick) the Business Activity Monitor sample, which is an extended example of an application program written partially in Java and partially in CAL, is now fully included. (Magnus) Cal.Tutorials.DataProcessing is a new tutorial CAL module showing how to connect to a relational database (in this case the Apache Derby database), perform SQL operations, express the results as CAL records, and perform various reporting and summarizing operations with those records.
- (Bo) can declare foreign functions whose implementations are Java cast operators or Java instanceof operators. See the CAL User's Guide documentation on foreign functions for details.
- (Bo) CAL Lists can be converted to and from Java Iterators in a lazy fashion. See List.toIterator, List.toIteratorWith, List.fromIterator and List.fromIteratorWith.
- (Bo) support for CAL foreign functions making callbacks into CAL to trigger further CAL evaluations. See the Prelude.CalFunction type and the functions Prelude.makeCalFunction and Prelude.evaluatedCalFunction.

Year 2007:

Open Quark version 1.3.0-0 (January 24, 2007)

- (Joseph) hierarchical module names
- (Neil, Magnus) new Record Field Selection Gem in the Gem Cutter

- (Bo) foreign functions can be created corresponding to Java null values, Java null checks, and Java primitive array operations (creation, subscript, update and length).

Open Quark version 1.4.0-0 (April 11, 2007)

- (Joseph) lazy pattern matching: new CAL syntax to lazily unpack data constructors and records within a let expression.
- (Ray) improved readability of generated Java sources for understanding the Java code actually generated from CAL sources:
- (Magnus) implemented the Computer Language Shootout Benchmarks for CAL
- (Greg) new features in the CAL Eclipse Plug-in: CAL Workspace view, CAL Outline view shows the CAL entities in a particular module, organize imports refactoring
- (Rick) CAL metadata editing support in the CAL Eclipse Plug-in
- (Edward) CAL branding plug-in for the CAL Eclipse Plug-in

Open Quark version 1.5.0-0 (June 15, 2007)

- (Joseph) standalone Utility JARs
- (Magnus, Ray) improved formatting of CAL source code by tools
- (Jen) organization and appearance of the ICE help has been improved
- (Edward) inclusion of help in the CAL Eclipse Plug-in
- (Greg) CAL Eclipse Plug-in features: New Module dialog, New Quark Binaries wizard, shift-hover to show CAL source code for a symbol, templates for common CAL syntactic structures, insert type declarations action.

Open Quark version 1.5.1-0 (July 19, 2007)

- (Joseph) lazy loading of foreign entities from compiled CAL modules
- (Bo) concurrent evaluation of CAL functions on a single execution context

Open Quark version 1.6.0-0 (August 29, 2007)

- (Ray) IO source generator
- (Jen) Connections in the Gem Cutter can now be automatically split to form a collector/emitter pair.
- CAL now requires Java 5. The implementation and Java API have been updated to use Java 5 features such as generics.
- (Edward) Console page for CAL expressions in the CAL Eclipse Plug-in.

Open Quark version 1.6.1-0 (September 21, 2007)

- (Jen) new document *Getting Started with Open Quark* provides an introduction to Open Quark for new users and an overview of the other documents and resources available for learning more.
- (Greg) new Quick Outline view in the CAL Eclipse Plug-in.

Embedded CAL 1.6.1_0 (September 28, 2007)

- (Andrew) Embedded CAL is an experimental add-on to the CAL Eclipse Plug-in that allows CAL expressions and modules to be embedded directly into Java Editors in Eclipse. This provides a very natural way to combine CAL and Java.

Open Quark version 1.7.0-0 (October 26, 2007)

- (Jen) new Record Creation special Gem in the Gem Cutter.
- (Magnus) ability to create record instance functions without adding new compiler primitives.
- (Joseph) Standalone CAL libraries
- (Bo) smaller-footprint deployment options. The JAR previously containing the CAL compiler and its runtime support has been split into 2 separate JARs.

- (Joseph) improved refactoring and symbol identification support in the CAL Eclipse Plug-in, as well as the CAL services layer for local variables, type variables and data constructor field names
- memory-use improvements in the CAL runtime:
    - (Ray) elimination of need of the execution context to hold onto the rootnode.
    - (Malcolm) experimental bytecode optimization to reduce memory usage in certain cases by nulling out the last reference to a method argument or local variable

Open Quark version 1.7.1-0 (November 15, 2007)

- (Ray) experimental code generation optimization to reduce memory usage in certain cases by nulling out the last reference to a method argument or local variable.