CLOS - A Perspective

The Common Lisp Object System

By Richard Barber

What is COMMON LISP AND CLOS?

Common Lisp (CL) is a consolidation of the many dialects of Lisp (a programming language created over 40 years ago by John McCarthy at MIT) — the first ANSI standard object oriented programming language. Unlike its aging contemporaries COBOL and FORTRAN, Lisp is entering its prime. PC hardware is now ready for Lisp, which stays young by continually absorbing new ideas and evolving to meet the needs of its users.

Object oriented programming entered Lisp in the early 1980's. CLOS, the Common Lisp Object System, was designed to combine and improve on the ideas from these early systems. It is CLOS, which earned an ANSI Standardization, that makes Common Lisp object oriented. Although there are applications written in Common Lisp, which do not use CLOS, it is the newest development in Lisp. CLOS made a radical advance on other object systems, and influenced later languages, notably Java. Yet CL/CLOS is still a leader in expressive power and flexibility, and it remains one of the most powerful mainstream object oriented languages available today.

Who uses CL/CLOS?

CL/CLOS stands out in its ability to solve complex problems or where requirements change rapidly. Traditionally associated with research, CL/CLOS is now built into products and is in mission critical use by large corporations. A few examples of commercial successes include creating video games, web authoring applications and semiconductor design systems (See side bar).

Common Myths

Many shrink from using CL/CLOS because of myths. Maybe true fifteen years ago, they are now outdated.

- *Special machines are needed.* CL/CLOS is found across the whole spectrum of computers from Windows PCs to Unix workstations - even Linux. Like most other languages, you can link CL/CLOS applications to other systems such as the Web, COM and CORBA components, SQL databases, and dynamic link libraries (DLLs). Indeed, these technologies have evolved since 1985 specifically to allow applications written in different languages to communicate.
- *CL/CLOS is slow.* Since 1985, PCs have become 100 times faster. Much slower languages, such as Java and Visual Basic, have become popular. Today, CL/CLOS can run effectively on any standard PC and its implementations are faster.
- *CL/CLOS is too large.* Since 1985, PCs memories have become 100 times larger. Office applications are now 20 times larger. CLOS implementations are actually smaller than many other office applications.

Benefits of CL/CLOS

There are benefits throughout the development lifecycle from design, to coding, to testing and finally to end user execution. Many of these stem from its extreme flexibility. Rather than forcing a particular style of programming, CL/CLOS integrates object, metaobject, imperative, pure functional programming, higher order functions, and embedded domain specific languages. Problems are solved in the most appropriate.

Design

The expressive power of CLOS allows the design to be concise and simple. Its flexibility allows a close match to the problem, without being distorted to fit the limits of the programming language. With CL/CLOS macros (embedded domain specific language) programs can be written in terms of the problem domain. CLOS metaobjects enable the extension of the object system — objects become their own data dictionary. Persistence, constraint propagation, and transactions can be added.

CLOS classes are reusable through multiple inheritance and the

Real World Applications of CL/CLOS

Lisp in the Web
SchemaText, Web Authoring Technology (Schema GmbH)

SchemaText, an application designed by Schema GmbH, manages very large web documents including technical and legal documents, online help systems and product catalogues for Internet and Intranet sites. CLOS was chosen because of its high level of functionality and problem-solving capability. The runtime type dispatch enabled the collection of various data types while preserving the integrity of the object. The CLOS implementation enables SchemaText to revise a design to fundamental levels without compromising the data. Accommodating special customer requirements was also simplified. Visit: www.schema.de

Lisp in Risk Analysis
Fault Tree Analysis with FaultrEASE (Arthur D. Little, Inc.)

FaultrEASE, a large-scale fault tree analysis application, allows users to perform quantitative risk assessment operations dynamically. Fault tree analysis is a technique used to assess the likelihood of a given event. A fault tree is a model that graphically displays all possible faults leading to a given event. It is used by many industries: chemical, nuclear, transportation (automotive, aviation, and rail), pharmaceutical, and insurance. It is also used in the medical device industry (heart pacemakers, respirators, etc.) because the FDA requires risk analysis documentation for these items. With FaultrEASE, you create and build fault trees graphically. Editing is also done graphically, and it is very quick since you can operate on whole branches at a time. After quantifying your tree, you can calculate statistics and cut sets. You don't need to know or do quantitative risk assessments; the program performs them in real time. CLOS was used in developing three critical portions of the application: the placement algorithm, which enables dynamic fault tree layout; the graphical user interface; and the mathematical component (direct evaluation). Visit: www.process-safety.com

"anonymous" next method — calling the method defined on the superclass without the need to specify its class. Attaching methods to several classes at once provides CLOS programs the flavor of logic programming or pattern matching. Higher order functions and closures can encapsulate complex control flow. Runtime code construction and compilation allow scripting languages without writing interpreters. Runtime typing and a comprehensive type system allows full polymorphism.

Coding

CL/CLOS speeds coding, whether by teams or individual. Application development is interactive even when working with compiled code. A running program can be modified during a debug section and the change immediately tested. A class can be redefined without recompiling other parts of the system — and without other team members rebuilding or recompiling their code, either. Methods are added to a class without accessing or modifying its source definition. Namespaces ("packages") ensure team development without the danger of name clashes.

Testing

By removing the need to write elaborate test harnesses for each unit, CL/CLOS potentially lowers the cost of unit testing. In addition, tests can be scripted directly in CL/CLOS. Traps can be safely performed along with the analysis of any runtime exceptions including those caused by testing. Since complex data structures (such as trees and array) are read and saved as text, coding the test data structures and expected results aren't needed.

Execution

Runtime type checking and automatic memory management prevent programs from crashing. Any unexpected exceptions that do occur are reported in English, not in hex, and usually do not cripple the application. Since it's possible to modify a program when running, upgrades and enhancements can be made in the field.

CLOS and Object Oriented Development

CLOS is compatible with other standard object oriented design methodologies and tools. Component-based development is supported either through writing COM or CORBA components in CLOS, or by calling components from CLOS.

By providing the "smart" portion of the solution, CLOS may directly solve the hard part of the problem, or may orchestrate other components. This leverages existing investment in software and development skills.

Fifteen years ago, PC applications were small and relatively simple, and CL/CLOS would have overloaded the many computers, especially PCs. Today, applications are large and complex. With their pace of change accelerating, developers battle to keep up. Even PCs easily run CL/CLOS and CL/CLOS based applications, so its power is now ready to be exploited.

Richard Barber is a consultant in the fields of object-oriented development, software engineering, and business development. Founder and CEO of Procyon Research Ltd, he developed the first commercial implementation of CLOS. Richard researched medical expert systems at Cambridge University, and has degrees in Computer Science and Physics from Cambridge. He can be emailed at: *richard@jsb1.freeserve.co.uk*.

Further Information on CLOS

Object-Oriented Programming: The CLOS Perspective by Andreas Paepcke, MIT Press, 1993. 400 pages, ISBN 0-262-16136-2

Understanding Clos : The Common Lisp Object System by Jo A. Lawless, Molly M. Miller, Digital Press, 1991. 193 pages 5-5558-064-5

Some references on LISP and CLOS include:

http://www.apl.jhu.edu/~hall/lisp.html

http://www.franz.com