JazzScheme: Evolution of a Lisp-Based Development System

Guillaume Cartier
Louis-Julien Guillemette
SFP 2010
Outline

- What is JazzScheme?
- Why JazzScheme?
- Some applications
- History
  - Port from C++ to Gambit
- Jedi IDE
- Future work
Why JazzScheme?

- Good question!
Roots

- Little Lispe
- Lisp machines
  - Complete programming environment
- Common Lisp
  - Macintosh Common Lisp
- Scheme
- Prisme (1990) (screenshots)
  - Highly graphical applications for real-life clients
  - Full access to the source code of the system
Les hommes

Esquisse d'un pays

Pérou

Utilisation du sol

- forêts (54,4 %)
- terres à vocation agricole (24,1 %)
- autres utilisations du sol (21,5 %)
What is JazzScheme?

Development system based on Scheme and Gambit

- Module system
- Hygienic macros
- Object-oriented programming
- Optional typing
- Cross-platform UI based on Cairo
- Binaries for Mac OS X, Windows and Linux
- Lisp-based IDE
Why (contd.)

- Build commercial software in Lisp
- Promote Lisp
  - Not by trying to convince people of its advantages
  - By creating a Lisp-based development system to write complex applications that would have been extremely difficult to develop using mainstream languages (time, cost, feasibility, …)
Requirements

- Year 1998
- Large-scale enterprise development support
- Open-source
- Built entirely in its own language
Built entirely in Lisp

• Rapid development cycle
  – High-level language and tools
  – Only one language to learn and master
  – Fast evolution

• Live by your word
  – New features & optimisations
  – Constant testing
  – Tribute to Lisp

• Openness to the community
Some applications

- MetaModeler
  - Database modeling
- Scheduler (screenshot)
  - Automated rule-based scheduler for hospitals
- Uranos
  - Enterprise Resource Planning (ERP)
- Jedi
  - Lisp-based IDE all written in Jazz
Birth of “classic” Jazz

- Year 1998
- C++-based
Birth of JazzScheme

- Year 2006
- Meeting Marc Feeley
- Jazz becomes open-source JazzScheme
- Scheme-based
  - (Chicken, PLT, Bigloo but not Gambit!)
- Port to Gambit
Why Gambit?

- Lightweight, high-quality Scheme implementation
- Conformance (R5RS and IEEE Scheme standards)
- Portability
- Performance
- Reliability
- Debugging
- Rich API
  - C foreign-function interface
  - Lightweight thread system that can support millions of concurrent threads
  - Networking
  - Unicode support
Port to Gambit

- Scheme was just too great!
  - Jazz becomes a radically different language

- We ended up having to
  - Port the language from C++ to Gambit
  - Port the libraries to the new incompatible language
  - Port the UI code from Windows to X11 and Mac OS X

- Lisp’s syntax saves the day

- Port
  - 200,000 lines of C++
  - 15,000 lines of Scheme
Optimisations

- First working version
  - 95x slower than C++-based Jazz

- Statprof
  - Statistical profiler used to identify all the hotspots

- The current version
  - Gambit-based kernel now faster than the old C++-based kernel
Class-of

- class-of
  - Edit definitions
  - Edit references
- class-of
  - Multi-scheme Jazz (Chicken, PLT, Bigloo)
- Back / forward navigation
The present

- Auphelia
  - Christian Perreault
  - QT vs Jazz
  - Enterprise Resource Planning (ERP)

- Team

- Continuous evolution of the JazzScheme platform

- Part-time collaborators
  - Marc Feeley
  - Alex Shinn
C4

- Project
- Remote debugger
  - Backtrace
  - Inspect variable
  - Dynamic redefinition
Jedi

- Code walker
  - Compile time highlight of errors
- Snapshots
  - Uses the same debugger infrastructure as the Jazz, Gambit and SWANK debuggers
- Inspector
  - Full support for Gambit data types
- Profiler
- View explorer
  - F8 / Edit Action Handler on Start Profile
Future work

- Deterministic profiler
- Code coverage
- Console-based REPL
- Stepper
- Designer
- SWANK debugger
Resources

- Website: www.jazzscheme.org
  - Documentation
  - Tutorials
  - FAQ
- Discussion Group
  - groups.google.com/group/jazzscheme
- IRC on Freenode
  - #jazzscheme
  - #gambit
Thank you!
Jazz as a macro

- Usual language development approach
  - Write a compiler that generates to a target language
  - Write the runtime support (GC, memory management, ...)
  - Write an interpreter (optional)

- As a macro
  - Reuse of all the work invested in Gambit
  - Written in a high-level language
  - Only a code walker needs to be developed
  - No performance penalty
Text

- Find definitions & references
- Dynamic redefinition
- Incremental search
- Search & replace with IrRegex
- Syntax highlighting
- Clipboard ring
- Mouse copy
- Emacs