Mozart-Oz
Multi-paradigm Programming System

Boris Mejías and the Mozart community

www.mozart-oz.org

boris.mejias@uclouvain.be

Mozart-Oz

- Mozart is an implementation of Oz, a multi-paradigm programming language supporting
  - declarative
  - functional (lazy and eager)
  - object-oriented
  - concurrent
  - distributed
  - logic
  - constraint programming

  as part of a coherent whole

Mozart-Oz

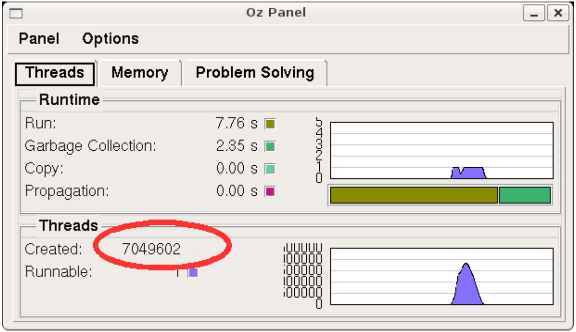
- Mostly used in academia but also in industry
- It runs on GNU/Linux, Solaris, MacOSX and other operating systems
- From Mozart Consortium (Belgium, Germany, Sweden) to an open Mozart community organized by a Board governance model with MEPs
- It provides the Oz Programming Interface (OPI)
- Strengths:
  - Concurrency: ultra lightweight threads, dataflow synchronization
  - Inferencing: constraint and logic programming
  - Distribution: network transparent, open, fault tolerant
  - Flexibility: dynamically typed, incremental compilation

A bit of Oz code

```
declare
fun {Fibo N}
   if N < 2 then 1
   else
      Fm1 Fm2
   in
      thread Fm2 = {Fibo N-2} end
      thread Fm1 = {Fibo N-1} end
      Fm1 + Fm2
   end
end
{Browse {Fibo 32}}
```

A bit of Oz code

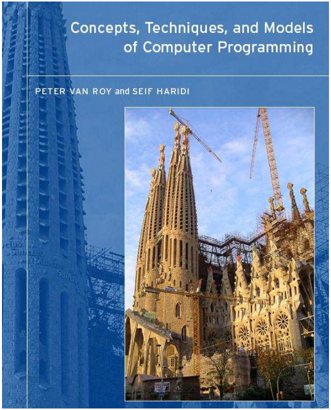
```
declare
fun {Fibo N}
   if N < 2 then 1
   else
      Fm1 Fm2
   in
      thread Fm2
      thread Fm1
      Fm1 + Fm2
   end
end
{Browse {Fibo 32
```



Teaching programming using Oz

"Concepts, Techniques, and Models of Computer Programming" by Peter Van Roy and Seif Haridi, published by MIT Press in 2004. (900 pages)

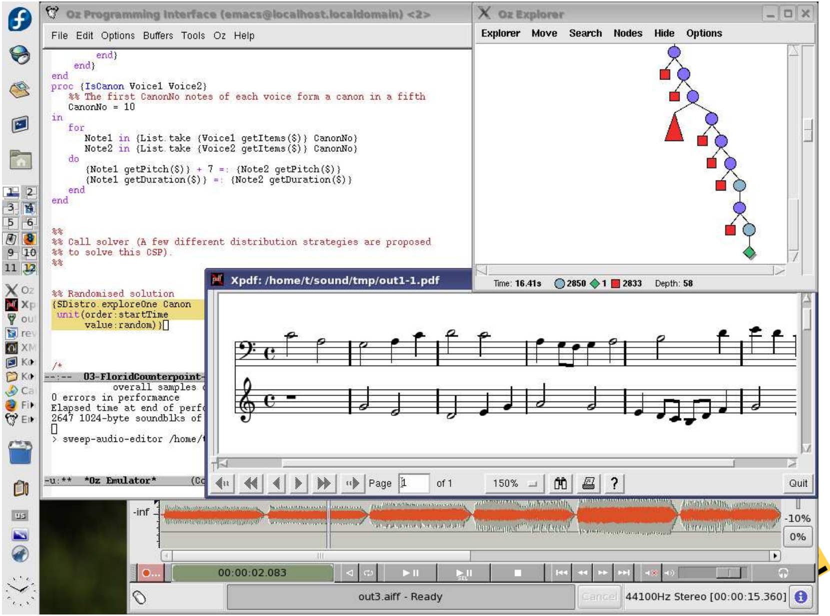
- One language to teach many concepts involved in all major programming paradigms
- Used for teaching in more than 20 universities worldwide
- The book is available in English, Polish, and soon in Spanish, Japanese and French



Strasheela
by Torsten Anders

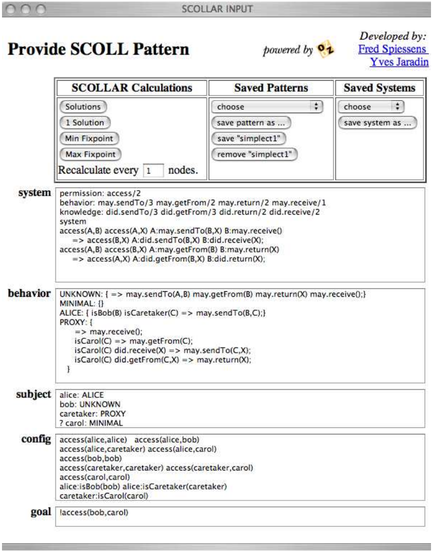
- A constraint-based music composition system
- Users declaratively state a music theory model (as Oz code) – computer generates music which complies with this theory
- A theory model is implemented by a set of compositional rules (constraints) applied to a music representation in which some aspects are expressed by variables
- Results are output into various formats, e.g. music notation and sound synthesis
- Strasheela is highly programmable and extendable, e.g. users control what information is stored in the music representation

Strasheela


Oz Programming Interface (emacs@localhost.localdomain) <2>
File Edit Options Buffers Tools Oz Help
end}
end}
end
proc {IsCanon Voice1 Voice2}
%% The first CanonNo notes of each voice form a canon in a fifth
CanonNo = 10
in
for
Note1 in {List.take {Voice1 getItems($)} CanonNo}
Note2 in {List.take {Voice2 getItems($)} CanonNo}
do
{Note1 getPitch($)} + 7 =: {Note2 getPitch($)}
{Note1 getDuration($)} =: {Note2 getDuration($)}
end
end
%%
%% Call solver (A few different distribution strategies are proposed
%% to solve this CSP).
%%
%% Randomised solution
{SDistro.exploreOne Canon
unit(order:startTime
value:random)}
03-FloridCounterpoint-
overall samples
0 errors in performance
Elapsed time at end of perf
2647 1024-byte soundblks of
> sweep-audio-editor /home/
-u:** *Oz Emulator*
Oz Explorer
Explorer Move Search Nodes Hide Options
Time: 16.41s 2850 1 2833 Depth: 58
Xpdf: /home/t/sound/tmp/out1-1.pdf
Page 1 of 1 150% Quit
-inf
-10%
0%
00:00:02.083
out3.aiff - Ready
Cancel
44100Hz Stereo [00:00:15.360]

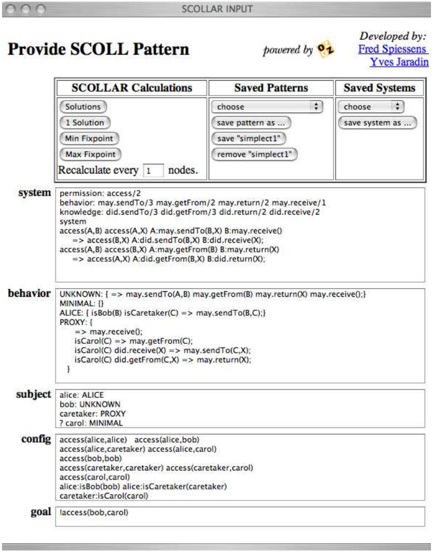
SCOLL Safe Collaboration Language

by Fred Spiessens and Yves Jaradin



SCOLL Safe Collaboration Language

by Fred Spiessens and Yves Jaradin

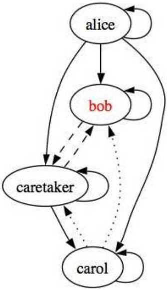


SCOLLAR RESULTS

Successfully parsed SCOLL program *powered by*

Search was completed

Calculated 2 solutions (all alive) in 0 Seconds.



Solutions	1	2
carol:may.receive()	1	**0**
carol:may.return(carol)	**0**	**0**
carol:may.sendTo(bob carol)	**0**	1

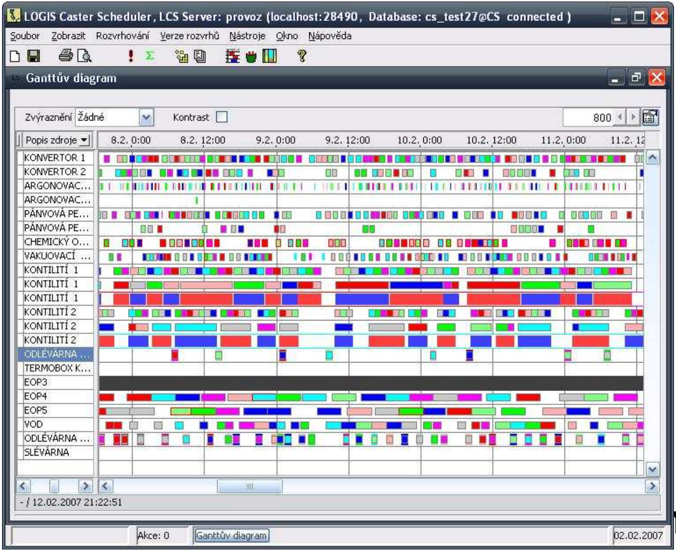
LOGIS Caster Scheduler

by Filip Konvička and LOGIS, s.r.o.

- It is a commercial planning/scheduling tool for continuous ingot steel casting plants
- Client/server application (Oz-based server, Java-based GUI clients)
- Users provides business and technological constraints from metal industry, and the application produce a schedule for the plant
- Able to produce a month's schedule for a medium-sized steel plant (about 200,000 tons/month) within 20 minutes. Previous methodologies never allowed plants to produce month's schedule.
- Developed and used in Czech Republic

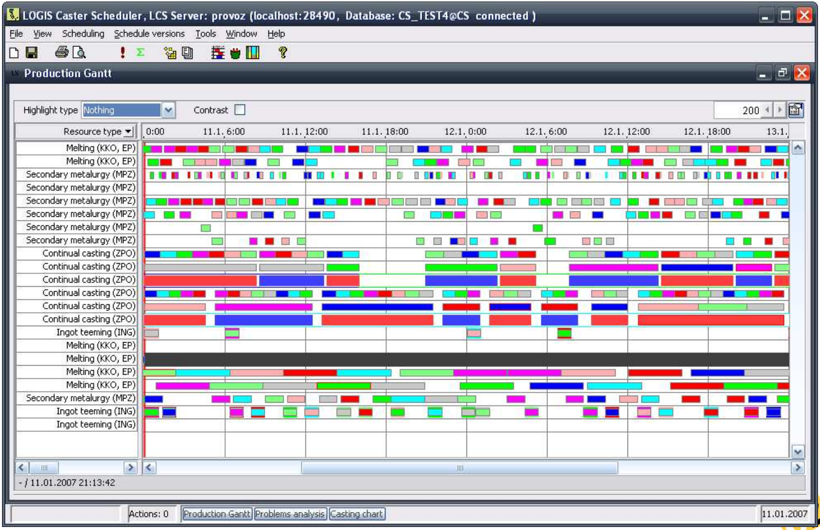
LOGIS Caster Scheduler

by Filip Konvička and LOGIS, s.r.o.



LOGIS Caster Scheduler

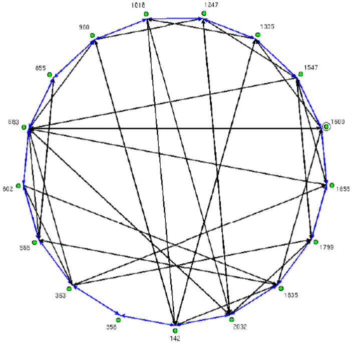
by Filip Konvička and LOGIS, s.r.o.



Peer-to-peer libraries P2PS/P2PKit

by Valentin Mesaros, Bruno Carton and Kevin Glynn

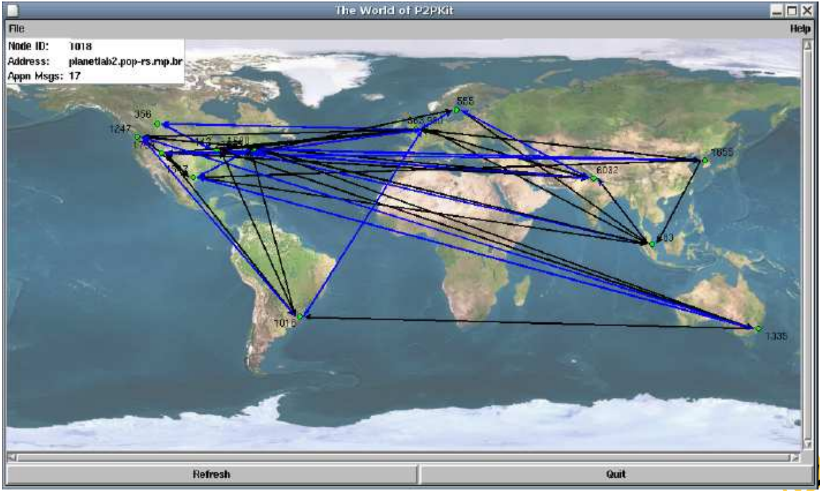
- Self optimized Chord-alike structured overlay network organized by successor, predecessor and finger-table
- Tolerant to link and processes failures



Peer-to-peer libraries P2PS/P2PKit

by Valentin Mesaros, Bruno Carton and Kevin Glynn

P2PS/P2PKit running on PlanetLab



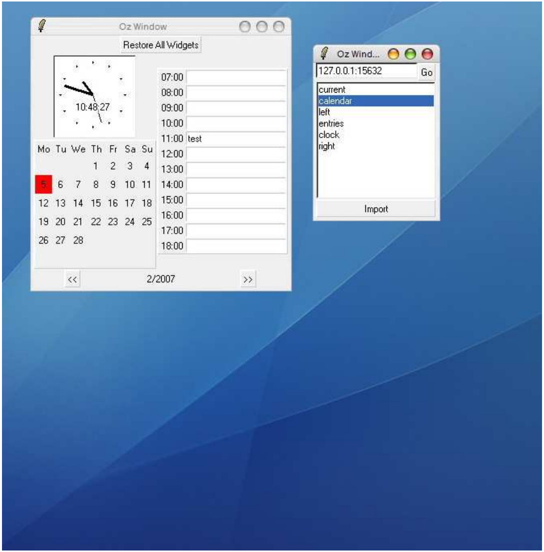
Enhanced Binding Library EBL/tk

by Donatien Grolaux

- State-of-the-art toolkit for graphical interfaces
- It mixes declarative and object-oriented approaches
- 1/3 lines of code compare to standard toolkits (Swing, AWT, GTk, etc.)
- Each window component is freely detachable from its original place, and can be dynamically attached to any other EBL/tk window
- This dynamic migration process is completely transparent to the running application itself
- Migration of UI can be done to a different machine
- Seamlessly integrated in Mozart

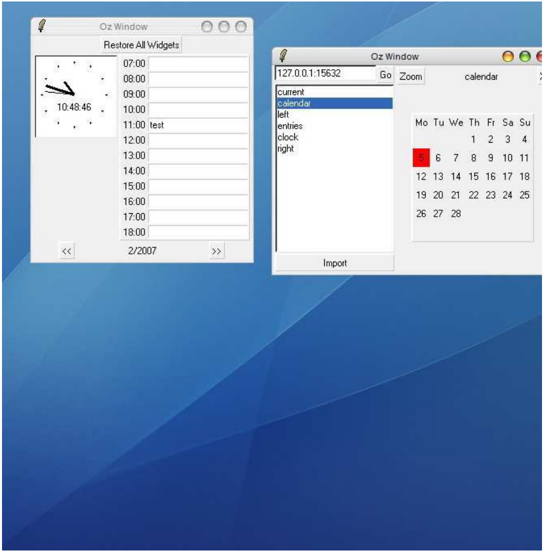
Enhanced Binding Library EBL/tk

by Donatien Grolaux



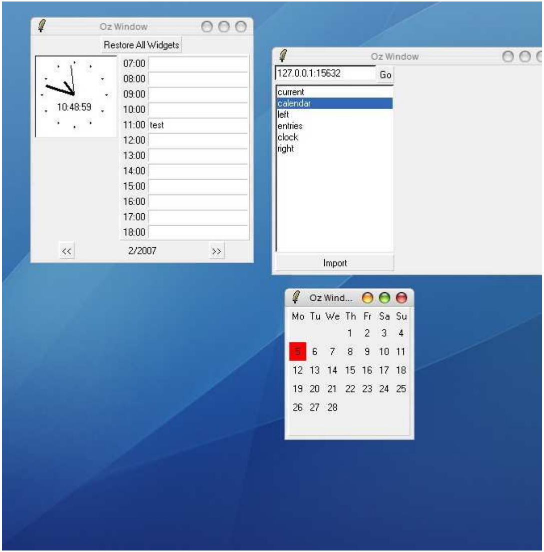
Enhanced Binding Library EBL/tk

by Donatien Grolaux



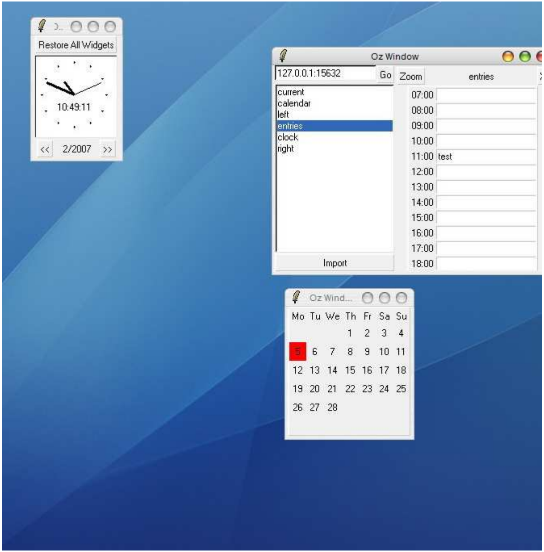
Enhanced Binding Library EBL/tk

by Donatien Grolaux



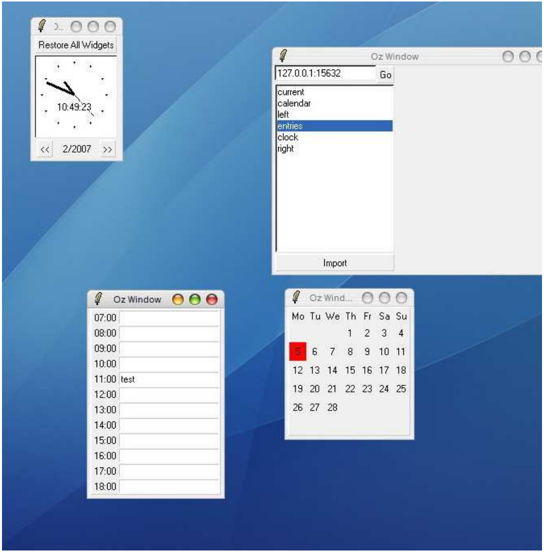
Enhanced Binding Library EBL/tk

by Donatien Grolaux



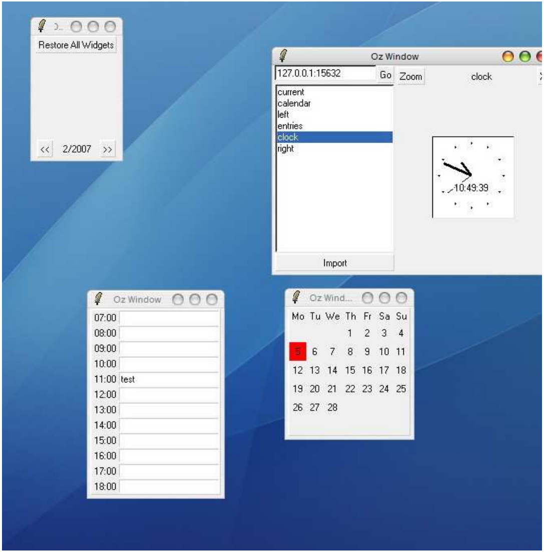
Enhanced Binding Library EBL/tk

by Donatien Grolaux



Enhanced Binding Library EBL/tk

by Donatien Grolaux



Solving package installation problems

by Sébastien Mouthuy

- Check that any package proposed in a distribution could be installed with respect to its dependencies requirements (same as aptitude, yum, etc)
- NP-Hard problem solved with constraint programming
- New search heuristics giving solutions for any package in less than 2 seconds
- It can find an installation solution to all packages of the entire Debian distribution (33200 different packages) in less than 1h50. Much faster than SAT solvers
- Simple implementation using cheap threads and data-flow synchronization

Current Projects

- **MozDSS**: Integration with middleware for transparent distribution support called Distribution SubSystem (Erik Klintskog, Raphaël Collet, Boriss Mejías)
- **GeOz**: Integration with Gecode, a state-of-the-art constraint programming library (Gustavo Gutierrez et al. in Colombia)
- **EVERGROW**: European Project supporting our peer-to-peer development
- **SELFMAN**: European Project to study large self-managing distributed applications based on structured overlay networks

Final Message

- Mozart-Oz is a powerful and mature programming system
  - Since 1995
  - Around 10^6 lines of code
- It supports all major programming paradigms giving you the possibility of choosing the right one for every problem
- Widely used for constraint programming
- It supports cheap concurrency and distributed programming transparently
- Not only for academia. Also professional software development.

Useful links

- **Mozart-Oz**: `www.mozart-oz.org`
- **Strasheela**: `strasheela.sourceforge.net`
- **SCOLL**:
  `www.info.ucl.ac.be/~fsp/scollardocmain.html`
- **LOGIS Caster Scheduler**: `www.logis.cz`
- **P2PS**: `gforge.info.ucl.ac.be/projects/p2ps`
- **P2PKit**: `p2pkit.info.ucl.ac.be`
- **EVERGROW**: `www.evergrow.org`
- **SELFMAN**: `www.ist-selfman.org`