Changes

Version 1.3.2
June 15, 2006

Abstract

This documents gives a brief overview of the changes between different Mozart versions.

Credits

Mozart logo by Christian Lindig

License Agreement

Contents

1

Mozart 1.3.2

Mozart 1.3.2 is a bug-fix and maintenance release. One of the main fixes is adding compatibility with gcc compilers newer than 3.3.x, including the 4.x series.

1.1 Changes in Oz, Mozart libraries and UI

1.1.1 Dictionaries

Copying of dictionaries was broken. Fixed. (bug 1352)

1.1.2 FSF

Mozart added to the FSF directory. (bug 1207)

1.1.3 Application module

`Application.getGuiArgs` now returns arguments as documented. (bug 1691)

1.1.4 Psql Contribution

The contribution to work with PostgreSQL is working again. To build it use `-enable-contrib-psql` during configuration.

1.2 Changes in the implementation that affect usability and performance

1.2.1 Oz Compiler

- Keyword `native` has been removed from class's properties. This implies a change in the module `Open` in contributions, fixing a runtime error (bug 1277).
- A warning is triggered when a non-valid property is added after keyword `fun` in function definitions. This prevents programmers from creating eager functions when keyword `lazy` is misspelled (bug 1437).
- Added `-warnshadow` option to compiler to emit warning message when variables in an outer scope are shadowed (bug 1707).

1.2.2 Garbage Collection

- Indexes have been fixed to work with RAM memory bigger that 1GB.

1.2.3 Distribution Layer

- `NewPort` does not block anymore when using a distributed variable for the stream. Currently, the builtin receives one input and one output, but it should change to two outputs in a future release (bug 1521).
- Problem with sockets generating 'segmentation fault' with P2PS is fixed (bugs 1304, 1339).
- Failed assertion in the Owner table fixed (bug 1461).

1.3 Changes in the implementation that affect maintainability and portability

- Mozart 1.3.1 compiles with gcc up to version 3.3.x. Mozart 1.3.2 add compatibility to build using gcc newer than 3.3.x and up to 4.1.x
- Succesful build reported on MacOSX/intel after fixing configuration. (bug 1859)

1.4 More Bug Fixes

- Error messages: 1429, 1816
- XML related: 1333
- GCC Compiler: 1636
- RPM building: 1649
- Reflect module: 1839
- FloatToString: 1869
- Emacs OPI related: 1890
- Tools: 1332, 1459, 1775
- Documentation: 590, 991, 1371, 1445, 1374, 1375, 1419, 1466, 1479, 1614, 1660, 1818, 1820, 1832

2

Mozart 1.3.1

Mozart 1.3.1 is a bug-fix and maintenance release, but most importantly it has undergone important modifications to make it work correctly with Mogul again.

2.1 Changes in MOGUL and ozmake

2.1.1 Support for multiple package formats

A serious issue for the 1.3.0 release was that `ozmake` packages produced for 1.2.5 and for 1.3.0 where not compatible. The MOGUL[1] infrastructure was revised and extended to transparently provide support for both. Packages contributed in one format are now automatically also converted and made available in the other format. `ozmake` now uses Mozart version-specific areas to install packages so that you can have both 1.2.* and 1.3.* versions of Mozart installed on your system. Furthermore, `ozmake` has been modified to download packages from the version-specific area of MOGUL appropriate for the version of Mozart that you are running. Unfortunately, this means that an old version of `ozmake` will not know that it needs to access an appropriate area; however, you can easily configure it to do so with the following command:

```
ozmake --config=put --archive=http://www.mozart-oz.org/mogul/pkg/1.2.5/sour
```

After that, `ozmake -install -package mogul:/duchier/select` will once again work as expected, download the named package from the 1.2.5 area in MOGUL and install it on your system.

2.1.2 Bug Fixes

A bug which sometimes caused `ozmake` to hang after downloading a package has been fixed. The compiler bug that was responsible was fixed and a workaround was also added to ozmake in case someone compiles it with a compiler that still has the bug.

syntax errors discovered while scanning functors for automatically extracting dependencies are now reported nicely

[1] `http://www.mozart-oz.org/mogul/`

2.2 Changes in Oz, Mozart libraries and UI

2.2.1 oztool

`oztool version` now prints the version number of the installed Mozart (e.g., 1.3.1).

2.2.2 loop support

Previously, the iterator `X in` *E1*`..`*E2*`;`*E3* only permitted to count from *E1* up to *E2* by increment of *E3*. Now, whether we intend to count up or down to *E2* is determined by the sign of *E3*.

2.2.3 POSIX support

Added `OS.rmDir` to remove a directory. Note that the directory must be empty before it is removed.

Extended `OS.stat` to also return `ino` and `dev` for inode and device information.

2.2.4 Standard Library

The standard library was improved and extended, and documentation was added for all modules. The library currently has sections for abstract datatypes, networking support, operating system support, GUI programming and XML support.

2.3 Changes in the implementation that affect usability and performance

2.3.1 Compiler

- Register allocation (bugs 1291, 1070, 1329)
- Improved static analysis (bugs 87, 299, 654, 655)
- Code generation (bugs 464, 867, 1322): in particular, this fixes the occasional problems with complex andthen/orelse expressions

2.3.2 Unification

Failure in the unification of OFS (open feature structures) did not always untrail properly (bug 1314)

2.3.3 Distribution Subsystem

Distributed resources (distributed entities without a consistency protocol represented by a special resource token on remote sites) handled inproperly during local garbage collection (bug 1304)

2.4 Changes in the implementation that affect maintainability and portability

2.4.1 Compiling C++ native functors on Windows

There is currently a bug in GCC (bugzilla #11005) which results in certain symbols not being exported by the Mozart DLL on Windows. This made it impossible to compile and link certain C++ native functors on that platform. We have now included a workaround which e.g. makes it possible to compile `mogul:/duchier/select` on Windows again.

3

Mozart 1.3.0

Mozart 1.3.0 is both the development and maintenance release. The work on 1.3.0 began immediately after 1.2.0, while the subsequent 1.2.* releases were intended primarily as maintenance releases. The Mozart system that became 1.3.0 has been extensively (and successfully) used for our own research projects.

The changes in the language, Mozart libraries and the implementation are summarized in the following sections.

3.1 Changes in Oz, Mozart libraries and UI

3.1.1 Redesign of by-need synchronization

The by-need synchronization mechanism has been completely redesigned and reimplemented. It now applies to all kinds of variables, and does not make use of futures. The term 'future' now only refers to read-only views of logic variables. Here is a summary of the changes in the language.

- Each variable has a need status: it can be 'needed' or 'not needed'. A needed variable triggers all by-need computations associated to it (see below). The need status of a variable can be queried with the function `IsNeeded`.
- The status of a variable changes monotonically: it can only go from 'not needed' to 'needed'. This change is triggered by a demanding statement, such as `{Wait X}`. Currently only free variables and futures can be 'not needed'. Kinded variables are needed by default. All values are needed by convention.
- A by-need computation is simply a computation that synchronizes on the need of a variable. This is provided by the statement `{WaitNeeded X}`, which blocks until X is needed. The procedure `ByNeed` has been redefined in terms of `WaitNeeded` and no longer uses futures. Lazy functions use the new `ByNeed`.
- As before, futures are created with the statement `X=!!Y`, which makes X a read-only view of Y. If X becomes needed, Y is automatically made needed. This allows to protect a by-need computation from binding attempts. The procedure `ByNeedFuture` provides this combination, together with failed values (see below) for reporting exceptions.

- Failed values are no longer considered futures, hence the term ‘failed future’ should be avoided. They are created with the primitive `Value.failed`, and have the status `failed` (returned by `Value.status`).

3.1.2 Uniform state operators

Mutable operators (Cells, Object Attributes, Dictionary/Array Entries) can now be updated with the `:=` operator and the current value retrieved with the `@` operator. More details in Chapter *Infix Notations, (The Oz Base Environment)*.

3.1.3 The Oz Base environment

Added `Array.exchange`, `Dictionary.exchange`, `WeakDictionary.exchange`, `Dictionary.condE` and `WeakDictionary.condExchange`.

3.1.4 Ozcar

The Ozcar manual has been largely reworked for improved precision and readability. Ozcar itself has been extended by the ability to debug threads remotely. This feature also supports the debugging of distributed applications.

3.1.5 Ozmake

’ozmake’ became a part of the Mozart’s standard library, and so it it included in the release.

3.1.6 Oztool on Windows

Mozart is now buildable with `gcc` version 3 and upwards. `oztool` now reverts to `gcc` and `g++` as default when invoked with the `-gnu` option. This can be configured as described in Section *Windows Environment Variables, (Oz Shell Utilities)*.

3.1.7 QTk

QTk now transparently supports accentuated characters on all platforms. The `hidden` parameter has been added to the `set` method of the `placeholder` widget. The `onCreation` parameter has been added to all widgets. The `QTk.flush` function has been introduced. Several bugs were fixed, in particular display problems on Mac OS X.

3.2 Changes in the implementation that affect usability and performance

3.2.1 Garbage collection

Entries in thread stacks and closures that are no longer reachable are now garbage collected.

3.2.2 Oz dictionaries

The implementation of Oz dictionaries has been replaced. In particular, the new implementation can efficiently handle lookups for absent entries (when it is necessary to confirm that an entity is not in the dictionary), and can handle subsequent numeric keys space-efficiently.

3.2.3 Performance of the distribution subsystem

The distribution subsystem and the distribution's marshaler have been analyzed and tuned in order to achieve high throughput between Mozart processes.

3.2.4 Pickling format

The pickling format has been incompatibly changed for the sake of space and run-time efficiency.

3.2.5 Bugs..

In general, we tried to keep up with the bug reports. There were too many of them to list them all here..

3.3 Changes in the implementation that affect maintainability and portability

3.3.1 GCC caveats for building from sources

The Mozart 1.3.0 release supports both the GCC 3.2+ with its new multi-vendor C++ ABI, and the old 2.95.*. GCC 3.2+ is the default on most contemporary GNU/Linux platforms, as well as available for most types of unix and the CYGWIN environment for Windows. We use GCC 3.3.2 on Linux and Solaris, and 3.3.1 with CYGWIN.

We would like to emphasize that Mozart binaries are not compatible if compiled with these two versions of GCC! In particular, all custom native libraries have to be (re)compiled with the compiler used for building the Mozart system itself.

Moreover, building a GCC 3.2+ compiler (to be used for compiling Mozart) on an old Linux installation is more complicated than it might appear, and we strongly advise against that. Upgrading Linux is much simpler and more reliable way to obtain the right GCC compiler.

4

Mozart 1.2.5

Mozart 1.2.5 is a bug-fix and maintenance release, but most importantly it has undergone important modifications to make it compatible with both old C++ compilers (e.g. `gcc 2.95.x`) and the new breed of C++ compilers (e.g. `gcc 3.x`) implementing the new multi-vendor standard C++ ABI (Application Binary Interface).

4.1 Revision 1

- Fixed a garbage collection bug that showed up on Solaris 9.
- ozdoc now generates PNG graphics rather than GIF.

4.2 Portability

- Mozart 1.2.5 is now compatible with newer C++ compilers that implement the new multi-vendor standard C++ ABI. In other words, it can be properly compiled with `gcc 3.x`. The binary layout of C++ classes as implemented e.g. by `gcc 3.2` has changed from what it was in e.g. `gcc 2.95.x` and this had some fundamental and lethal consequences for earlier versions of Mozart. The design of some datastructures in Mozart was accordingly revised.
- Mozart now runs on hppa aka linux-parisc. While Mozart 1.2.5 still is not ready for full 64 bit architectures, it does well under kernels that provide 32 bit addresses in user-space.
- For users of Gentoo Linux, we now provide ebuilds to install and maintain a Mozart installation using the portage system.
- improved Darwin support now also available in CVS.
- additional logic to fix the `glibc` problem with time ticks introduced when the Linux kernel bumped `HZ` up to 1000.

4.3 Graphics

- fixed the Tk interface to work with Tk 8.4.
- new optional `mozart-gtk` component providing GTK 1.2 bindings (see download page for further information).

4.4 Windows

changes and fixes that concern the Windows platform:

- `oztool` now uses `gcc-2` and `g++-2` as default when invoked with the `-gnu` option. This can be configured as described in Section *Windows Environment Variables, (Oz Shell Utilities)*

4.5 Virtual Machine

changes and fixes that concern the virtual machine:

- several fixes in the support for execution of Alice[1] programs

4.6 Constraint Programming

- fixed `FD.reified.card`
- fixed `FD.int` problem with descriptions with more than 1024 items.

4.7 Miscellaneous

- When building from sources, the default is now to ignore an existing mozart installation (this default can be overriden with the new `configure` option `-with-global-oz`). Previously, the user had to be very careful to remove from his/her environment all traces of `oz` prior to invoking `configure`.
- fixed an issue related to sharing when sending to a superordinated port.

[1] `http://www.ps.uni-sb.de/alice/`

5

Mozart 1.2.4

Mozart 1.2.4 is a bug-fix and maintenance release, with some enhancements.

5.1 Base

changes and fixes that concern the base language:

- bytestrings: space now prints as space instead of being encoded as octal
- fixed insertion sort in `dictionary.cc` (so what?)
- improved static analysis for `Port.new`: `{Port.new s _}` now diagnoses the error due to the typo (atom instead of variable)
- added `Value.isFailed`: true iff the arg is a failed future
- `Int.'div'`, `Int.'mod'`: work consistently and according to the specification for small and big integers (concerns negative numbers only)
- `Value.waitQuiet` no longer blocks on failed futures
- robustly handle the case of a messed up `OZLOAD`
- bug fix for `VirtualString.toAtom`
- Sort and Merge are now stable
- new environment variable `OZ_DOTOZ` to override default `~/.oz`. corresponding property `oz.dotoz`
- `List.take`, `List.drop` and `List.takeDrop` no longer unnecessarily block on nondet list tail that is not looked at

5.2 Windows

changes and fixes that concern the Windows platform:

- fixes to compile again with MSVC

- debug emulator can be built
- Control-C should now work
- configuration: Updated to link correctly under the newest mingw (check for either `-ldirent` or `-lmingwex`)
- fixed other problems due to changes in mingw library
- `OS.getEnv`: now correctly distinguishes between an unset environment variable and an environment variable set to the empty string
- fixed `Application.getCgiArgs` on Windows for newer Apache (same as previous item)
- fixed `Open.pipe` pbs
- `OS.pipe`: When the process at the other end of a pipe terminates, end-of-file is reported instead of an exception
- `OS.read`: When an end-of-file arrives on standard input, it is now correctly reported instead of blocking indefinitely
- fixed weird compilation/linking pbs, actually caused by clashing GIDs (see DP)
- oztool more flexible: supports `-I`, `-l`, `-L` and `-s` for GNU and MSVC
- OPI: fixed encoding (on Windows) for sending data with accented chars to oz process

5.3 Documentation

changes and fixes to the Mozart documentation:

- `install`: fixed obsolete hrefs, added windows/cygwin info
- `notation`: fixed def of "in expression"
- `tutorial`: typos
- `xsl/ozdoc-fontify.el`: replaced obsolescent dot-marker by point-marker
- `xsl/ozdoc.cls`: works again with newer version of float.sty

5.4 Constraint Programming

changes and fixes concerning the constraint programming support:

- documented limitations:
  - free var cannot be constrained in subspace: must first be constrained in home space
  - number of alternatives of a choice point must be less than 2^27-2

- `doc/system/fd`: fixed typos
- fixed `dis`
- fixed `FDBitVector::findMinElemen` to correctly return -1 for empty vector
- fixed corner case that was causing the infamous "change of representation" bug at 64 elements in rare occasions (`FSetValue::operator <=`)
- domain consistent propagators are now properly woken up
- `\=:` now correctly takes variable aliasing and entailment into account
- reified constraints: fixed occasional misbehaviour due to out of bound array access
- `FD.distribute`: bug fixes + improvements. (used to not correctly synchronize on stability on 1st invocation)
- `FS.distribute`: fixed to use stable order, improved, round robin bug fixed (used to drop variables)

5.5 Oz Language

changes and fixes concerning the Oz language

- `for` loop now supports new experimental features `while:E` and `until:E`
- `doc/loop` slightly improved
- `for _ in 1..5 do ... end` now works (it used to block)
- eliminated unnecessary memory allocation for experimental `collect` feature

5.6 System Modules

changes and fixes concerning the System modules:

- Application:
  - added `Application.processCgiString`
  - fixed `Application.getCgiArgs` on Windows for newer Apache
  - bug fix for `Application.getCmdArgs`
- Finalize:
  - `doc/system/finalize`: improved doc about stateless values etc...;
- Open:
  - documented `text` and `binary` flags

  - `Open.text`: raises a `closed` exception instead of a `type` exception when operating on a closed stream

- System:
  - documented `System.onTopLevel`
- Property:
  - synchronize on undet property values instead of raising an error
- DP:
  - 40% speed up for small distributed messages
  - speedup for pickling and 10% decrease in size
  - fixed memory leaks in libdp
  - global IDs are now robustly/provably unique
- Tk:
  - iso8859-1 chars properly handled
  - image resolver initialized using methods of pickle resolver

5.7 Tools

changes and fixes concerning Mozart related tools:

- Browser:
  - it is now ok to browse a port in a deep guard
- Inspector:
  - can now inspect array in deep guard
- Gump:
  - use relative import for generated native functor
- Compiler:
  - `\insert` with very long filenames now works
  - misc fixes
  - fixed `dynamicvarnames` pb
- ozdoc:
  - cross-referenced exercises now use sensible text
  - beginning of support for XML
- OPI / Emacs Mode:
  - fixed encoding (on Windows) for sending data with accented chars to oz process

5.8 Miscellaneous

- Configure/Build:
  - version checking improved
  - fixed pbs with new version of Bison
  - fixed pbs with new version of GMP
  - added switch `-enable-compile-elisp` (default=yes) as requested by Debian
  - added `-with-emacs-options=`*OPTIONS* (default=`-q -no-init-file` for GNU `-q -no-init-file` for XEmacs)
  - more robust creation of native functor interface in `contrib/ri`
  - support OpenBSD
  - configures and compiles again on recent Linux distros using new versions of libraries
- Examples:
  - `share/examples/grammar` updated
- Test Suite:
  - fixes + new tests

6

Mozart 1.2.3

Mozart 1.2.3 is primarily a bug-fix release addressing recently discovered memory leaks. It also contains enhancements and is fully backward compatible with 1.2.2

- Fixed a memory leak in the regex contrib: memory occupied by a compiled regex was not properly released during finalization.
- Fixed a memory leak in the unpickler. The leak affected particularly applications that performed:
  - numerous loading of compiled functors
  - numerous retrieval of values from a GDBM database
- Windows: setting environment variable `OZ_TRACE_LOAD` caused the system to misbehave. This was actually due to forgetting to copy the value returned by the environment variable lookup out of a static buffer.
- Many holes have been filled in the compiler documentation. In particular, the main chapters are all complete; only the appendices lack explanation.
- Constraint Programming: `FD.atLeast` and `FD.atMost` performed incorrect propagation when their first argument was a FD variable. The implementation was completely overhauled. All of `FD.exactly`, `FD.atLeast` and `FD.atMost` are now implemented as instantiations of the same template class.
- The `Application` module now exports `Application.processArgv` which permits to invoke argument processing on a list of strings explicitly provided as a parameter.
- The compiler's Gump support was modified so that generated native functor and parser state description are placed in by default in the directory of the source file. This can be explicitly overridden by option `gumpDirectory`, or by command-line option `-gumpdirectory`.
- Standard Library: Mozart 1.2.3 is the first release to include the Mozart Standard Library. At present the latter contains only `QTk` available at URI `x-oz://system/wp/QTk.o`

7

Mozart 1.2.2

Mozart 1.2.2 is an improvement release and is fully backward compatible with Mozart 1.2.1

- Improved debugging support for `for` loops: they now provide meaningful debug info and can be stepped in
- Reference documentation added to TkTools
- Windows:
  - Release 1.2.1 for Windows omitted file `cache/x-oz/contrib/os/mode.ozf` which is required by the GDBM contrib. This is now included.
  - `OS.uName` now correctly fills `machine`, `release` and `version` fields; Before, they always contained `"unknown"`, and provides more information in the `sysname` field. Before, the `sysname` field always contained `"WIN32"`; it is now `"win32s"`, `"win32_windows"` or `"win32_nt"`.

8

Changes between Mozart 1.2.0 and Mozart 1.2.1

The release of Mozart 1.2.1 is an important bug-fix release, with some enhancements, and is fully backward compatible with Mozart 1.2.0.

8.1 Important Issues

- A severe bug in record unification that has been introduced in 1.2.0 is fixed.
- Fixed incorrect variable aliasing detection on Windows (resulted in weaker propagation, therefore in different and larger search trees).
- Many improvements in the Windows port, in particular for subprocesses, finally correctly enabling oztool and ozmake.
- Both domain- and bounds-consistent variants of the alldifferent constraint for finite domains are available.

8.2 Changes

- Inspector replaces Browser as default viewer in Explorer and Ozcar
- The verbosity of printing variables can be controlled by the property `'print.verbose'`
- Mozart uses sorting in many important places (record construction, dictionaries, finite domains, many finite domain propagators). All uses of sorting now share a single, efficient and robust implementation.
- Documented `FD.distinctD` (domain-consistent alldifferent) and added `FD.distinctB` (bounds-consistent alldifferent, naive quadratic version)

8.3 Bug Fixes in Detail

record unification an invalid optimization was introduced in 1.2.0 that affected speculative unification of records

variable aliasing detection (Windows) variable aliasing detection was inoperative. As a result propagation was weaker. This could be observed with the SEND+MORE=MONEY example which produced a larger search tree

`oztool` (Windows)

`oztool ld` now also works under Cygwin (not only Mingw32). Furthermore, options are now accepted in any order, and options `-I`, `-l`, `-L` and `-s` are supported for gcc and Microsoft Visual C++

`OS.system` and `Open.pipe` (Windows)

`OS.system` was broken in 1.2.0 and has now been repaired; bug fixes in inheritance and closing of handles

failed futures

(dis)equality testing now correctly passes up exceptions. `Value.byNeedFail` is now careful to be non-requesting.

IO problems (Windows 2000 Service Pack 2)

system would freeze if `ws2_32.dll` was not loaded

Entailment of propagators

propagators are now again included in the suspension count

`Combinator.reify`

fixed bug related to merging

`Schedule.cumulative`

fixed bug related to sorting task intervals

8.4 Miscellaneous

Cross-compilation is no longer necessary to build Mozart for Windows. The entire system can now be natively compiled on Windows under the Cygwin environment.

9

Changes between Mozart 1.1.0 and Mozart 1.2.0

Mozart 1.2.0 is primarily a maintenance release with improved usability, maintainability and performance. The changes in the language, Mozart libraries and the implementation are summarized in the following sections.

9.1 Changes in Oz, Mozart libraries and UI

9.1.1 Loops

*** patterns are now supported:**

```
for X#Y in L do ... end
```

*** iterator expressions with a bit of a C-flavor are supported:**

```
for X in Init;Cond;Next do ... end
```

for example:

```
for I in 1;I<5;I+1 do ... end
```

also iterators of the form `E1..E2`, `E1..E2;E3`, `E1;E3` as before.

*** nullary `break` and `continue` procedures can be obtained using loop features, e.g:**

```
for break:B X in L do ...{B}... end
```

*** EXPERIMENTAL: loops can be used as expressions using a hidden accumulator, e.g.:**

```
{Show
 for collect:C
     L in Ls
 do
    for X in L do
       if {IsOdd X} then {C X} end
    end
 end}
```

This experimental facility uses loop features `collect`, `append`, `prepend`, `minimize`, `maximize`, `count`, `sum`, `multiply`, `return`, `default`. For more information check the documentation (*"Loop Support"*).

9.1.2 'Failed' Futures and Module Manager

- `{Value.byNeedFail E ?V}` binds `V` to the new notion of a 'failed future'. Any attempt to synchronize on `V` raises `E` as an exception.
- The module manager was updated to use 'failed futures': when a module cannot be successfully linked its value becomes a failed future (instead of a record) which raises the exception which caused linking to fail each time the module is subsequently accessed. Thus, programmers have a chance of catching and recovering from linking errors.

9.1.3 Spaces

- `Space.askVerbose` returns `suspended` rather than `blocked`
- An exception is raised, if the argument to `Space.commit` refers to a non-existing alternative
- The control condition for application of space operations have been unified and extended, see the documentation
- `Space.kill` kills a space by injecting fail into it

A full treatment of spaces is available in the Christian Schulte's doctoral dissertation "Programming Constraint Services", from the Mozart publications page.

9.1.4 Distribution Subsystem

- The Oz programmer interface to the distributed subsystem has been extended. Parameters like buffer size and timeouts can now be specified at runtime.
- The family of tools for monitoring the behavior of the Mozart system has been extended with a new member, the Distribution Panel. The tool displays information about known remote Mozart sites, amount of communication, measured round-trip and exported/imported entities. Possibilities to remotely monitor other Mozart processes does also exist.

9.1.5 Constraint Systems

Constraint systems (finite domains, finite sets) have undergone, as usual, various improvements and bug fixes. Check the documentation.

9.1.6 Port Improvements

- Ability to send from a subordinated space to a superordinated space (provided that no local variables and names are referred to).
- SendRecv does a send and returns an answer. For details, see doc. This again works across spaces.

9.1.7 Pickling Format

Unfortunately, old pickles cannot be read by this new system. We do plan to have a generic conversion tool, but face a lack of human resources.

9.1.8 'ozl –rewrite'

The Oz linker (ozl) now supports a new command-line option, '–rewrite', which allows to transform the import URLs used by the output functor. Check the documentation for details.

9.2 Changes in the implementation that affect usability and performance

9.2.1 Bugs

Bug fixes, including, but not limited to:

- the core (centralized) system
- distribution subsystem
- constraint solving facilities
- Oz debugger (ozcar)
- the Windows port
- various memory leaks

All in all, a lot of them. Really a lot.. The system is used now for a series of our projects, as well as for projects outside the Mozart consortium, and also for teaching at all our three sites. Have also a look at http://www.mozart-oz.org/cgi-bin/oz-bugs/FIXED

9.2.2 2GB of Live Data

Oz programs can reference now up to 2GB of live data on a computer with the 32bit address space, compared to .5GB for all the previous releases.

9.2.3 New Supported Platforms

New supported platforms - linux ppc & pentium 4.

9.2.4 Improved Distribution Subsystem

The distribution subsystem has been improved, in particular, as the traffic between Mozart sites increases. In extreme cases the win is up to orders of magnitude.

9.2.5 No Fast Inter-Site Communication

Unfortunately, the inter-site communication over shared memory (property `'distribution.virtuals` see also Chapter *Spawning Computations Remotely:* `Remote`, *(System Modules)* is currently inoperable. We are working on bringing it back in the next release. This is caused by extensive changes in the implementation of the distribution subsystem, as outlined in Section 9.3.

9.3 Changes in the implementation that affect maintainability and portability

9.3.1 Accessing Oz Data Structures

Mozart 1.2.0 features the new design and implementation of the part of the run-time system (engine) that deals with allocation and accessing Oz data structures. This not only allows 2GB of live data, as mentioned in Section 9.1, but also:

*** Greatly enhanced portability** Mozart does not impose anymore any constraints on where the "C" data regions are mapped, which is different across different flavors of Un*x. This also simplified the Windows port.

*** Is THE prerequisite for clean 64bit ports** while this release still does not support 64 Bit machines such as Alphas, etc, this change is the first – and quite big – step towards that goal.

9.3.2 Redesign of the Distribution Subsystem

Distribution subsystem has undergone a principle overhaul.

- The distribution subsystem is cleanly divided into a protocol layer and a message-passing layer, with interfaces between them explicitly specified.
- Flexible connection establishment. A pair of Mozart sites connect now by execution of dedicated, replaceable Mozart functors. This enables customization of connection protocols for creation of closed subdomains or traversal of fire-walls. Currently there is no user interface to this facility, but we plan to introduce one.
- The message-passing layer has been redesigned and re-implemented. The new design is featured by:
  - better resource utilization, both in terms of memory and system resources (e.g file descriptors).
  - improved balance between the (centralized) engine and the distribution subsystem in terms of run time. In particular, large messages sent between Mozart sites do not lock out the engines on either side.
  - an open architecture enabling introduction of new transportation mediums.
  - higher throughput by better pipelining of messages.
  - caching of TCP channels reworked.

- automatic round trip calculation.
- failure detection on measured round trips rather than by TCP.

A document describing the Distributed Subsystem added to the documentation tree.

10

Changes between Mozart 1.0.1 and Mozart 1.1.0

Mozart 1.1.0 is a maintenance release that features a completely new and improved implementation of pickling and a major improvement of the constraint programming primitives.

10.1 Changes

10.1.1 Pickling

Due to some redesign of the instruction set and the pickling algorithm, the pickle format changed between Mozart 1.0.1 and Mozart 1.1.0. A conversion tool has been made available however, see Chapter *Conversion of Pickles:* `convertTextPickle`, *(Oz Shell Utilities)* for documentation.

10.1.2 Constraint Programming

General improvements One of the main achievements in the 1.1.0 release is a fairly complete overhaul of the constraint programming functionality in Mozart. This makes the system leaner with respect to both code size and memory requirements and several even severe bugs have been fixed. In average, the refurbishment buys you a 20% speedup on constraint applications (up to 40% in rare cases).

Constructive disjunction removed That's basically just for the records: nobody used it. Since it was complicated and a constant source of problems it has been removed. In the rare case that you used constructive disjunction, contact us for help.

`FD` and `FS` synchronization behavior corrected

All `FD` and `FS` propagators now conform to their documentation as it comes to synchronization on their arguments. Mozart 1.0.0 and 1.0.1 were buggy in that execution did not block even though the propagators required their arguments to be finite domain or finite set variables. Watch out! In case your scripts that use finite domain or finite set propagators just block (they show a light (ugly) green color in the Oz Explorer) this is a likely cause! Fixing is easy: just make sure that all variables supplied to propagators are in fact constrained to be finite domains or finite sets.

`Space` and `RecordC` are system modules

To achieve better factorization of constraint programming support in Mozart all constraint programming modules are system modules rather than modules in the base environment. This results in a much smaller memory footprint of the Mozart engine in case the constraint programming facilities are not needed.

10.1.3 Distribution

The distribution layer of Mozart has problems with fire-walls. From our point of view fire-walls defines all sub nets that restrict their traffic in some way. It has been impossible to connect to oz sites through any kind of fire-walls up till now. A naive solution is included in this release. It will only enable connections through the simplest of fire walls, but that is better than nothing. We are working on a more general solution that will enable our sites to work over more complex fire-walls.

There were problems related to the shortcoming of fire-walls. When a Mozart sites needs to communicate it will try to open a connection. If it fails to reach the desired site it will time-out and retry unless it can deduce that the destination site is dead. The site will continue trying to open the connection until it succeeds or finds the site dead. This behavior can disturb fire-walls a lot. The time-out is now growing with a growth factor. There is now a way to alter the start time-out value, the growth factor and the timeout ceiling.

10.1.4 Documentation

Global Index The online documentation has been provided with a new index that encompasses all index entries from the individual documents. It can be reached from the main documentation page, either using the link in the margin or the link under the Getting Started/Documentation header. Caveat: Not all documents have a useful index yet!

Postscript and PDF Is available now.

10.1.5 Support for loops

In order to provide convenient syntax for loops, 2 new keywords have been introduced: `for` and `do`. This is an incompatible change. Check your code: you must now quote every occurrence of ‘for’ and ‘do’. Support for loops is still preliminary. The general syntax is:

```
for Iterators do ... end
```

where *Iterators* is a sequence of 1 or more iterators. Supported iterators are e.g.

```
X in L
```

for iterating over the elements of a list

```
X in I..J
```

for iterating from integer `I` to `J`. The loop terminates as soon as one iterator runs out. The complete documentation is available in *“Loop Support”*

10.1.6 Compiler Macro Names

The Mozart compiler defines macro names to identify the version of the system that is running (see Section *The Compiler's State, (The Mozart Compiler)*). These used to be

```
Oz_1   Oz_1_0   Oz_1_0_1
```

for Mozart 1.0.1, but to avoid clashes with the macro names provided by DFKI Oz, they are now

```
Mozart_1   Mozart_1_1   Mozart_1_1_0
```

10.1.7 URL Support

The format of URL records has changed incompatibly. It is now simpler: feature `absolute` is a boolean indicating whether the url is absolute and feature `path` is now just a list of strings representing the components of the path. An empty component is now simply omitted when converting to a string using cache syntax: thus the bug involving a `//` in the middle of a path has now disappeared. Parsing urls is also faster.

10.2 Fixes

10.2.1 All Platforms

- Return methods for classes in `Tk.menuentry` added.
- Added `Class.getAttr` to straightforwardly resolve multiple inheritance conflicts.
- Fixed printing of '~' for floats in virtual strings (bug 390).
- Code garbage collection bug fixed (bug 389).
- Bug in `Record.dropWhile` fixed, thanks to Benko Tamas (bug 383).
- Bug in `FD.exactly` fixed, thanks again to Benko Tamas (bug 378)
- Bug in documentation of `Append` fixed (bugs 331, 372)
- Bug fix in `ByNeed` returning a variable (bugs 340, 370)
- Several bug fixes for networked file systems (aka interrupted system calls) (bug 360)
- Bug fix in handling `X=X|X` during garbage collection (bug 359)
- Bug fix in `IsDet` for distributed variables (bug 357)
- Added `Pickle.pack` and `Pickle.unpack` for pickling and unpickling from/to byte strings.
- Fixed bug for doing large number of http requests (bug 350)

- Bug fix for binding faulty distributed variables (bug 348)
- Several compiler bug fixes (bugs 304, 305, 306, 344, 339)
- Bug fix in raising distributed programming exception (bug 341)
- Several fixes in unification (bugs 337)
- Some quirks in documentation fixed (bugs 257, 295, 302, 322, 327)

10.2.2 Windows

- Executable functors are now fully supported under Windows.
- More contributions have been made available. In particular the native functors for the regex and gdbm modules have now been built.
- Subprocesses started from Mozart do not open new console windows.
- Mozart is no longer confused by other programs such as `fortify` by a complete redesign of the communication between Mozart and Tk (bug 338).
- Temporary files are put under `C:\TEMP` by default (instead of `C:\`). Creating many files directly under `C:\` made Windows NT freeze.
- `ozd` now looks in the registry to see whether it can figure out where Emacs is installed. Furthermore, `ozd` depended upon `bash` as command interpreter to start Emacs–now it also works with `command.com` and `cmd.exe`.

10.2.3 Linux

- Memory management for Linux 2.2.x fixed (bugs 391, 403).

10.2.4 Other Platforms

- Initial support for Mac OS X.
- Compiles under FreeBSD (Bug 393).

11

Changes between Mozart 1.0.0 and Mozart 1.0.1

This is a minor improvement release to fix some small bugs and offer some improvements. You can judge yourself whether you should upgrade to 1.0.1 by reading the list of fixes and improvements. If you have suffered from any of the problems mentioned, you should definitely download the new version.

General fixes

- Obsolete menu entries in OPI removed (Bug 276)
- Argument parsing made even more POSIX compliant (Bug 278)
- Error messages pop up right buffer in OPI (Bug 243)
- OPI connects correctly to engine during startup
- Module managers resolve user names '~name' in file names (Bug 219) Many small fixes

Windows fixes

- Blanks in URLs work now: Mozart can now be installed into a directory whose path has blanks in it (for example `C:\Program Files\Mozart`) (Bug 255)
- Default contributions available
- Performance improvements for Graphics
- Improved installation under Windows (Increased Mozart awareness for Microsoft Internet Explorer and Netscape Communicator)

Unix fixes

- OZHOME can be adapted in oz startup script
- Linear solver packages excluded by default

Linux RPM fixes

- Version numbering scheme fixed such that upgrades become possible

Other platform fixes

- Configure problems for FreeBSD 3.0 (freebsd*-i486) fixed
- Ports to Irix (irix6-mips), OSF-Alpha (osf1-alpha), HPUX (hpux-700) improved

Improvements

- Remote module managers support arbitrary fork methods (in particular `ssh`)
- Parallel search engines allow specification of fork methods
- Example programs included in all distributions (not only rpms)
- `Open.pipe` allows brute force shutdown via close method

12

Changes between DFKI Oz and Mozart 1.0.0

12.1 General Changes

12.1.1 Functors and Modules

Mozart now comes with a powerful internet-based module system that supports lazy loading, native modules and more. For an introduction see *"Application Programming"*.

To make best use of the new module system, the previous Oz Standard Modules have been split into the *base environment* and the *system modules*. The compiler always provides the base environment, it contains all operations working on data structures like records, lists, and so on. For more information see *"The Oz Base Environment"*.

All remaining modules (including the constraint programming support) are now provided as system modules that are subject to import in functor definitions. The system modules are described in *"System Modules"*.

The Oz Programming Environment however still follows the design to ease explorative development. For that reason all system modules are still available in the Oz Programming Environment. The Environment nicely exemplifies the merits of the new module system: while providing all system modules the Environment starts in a fraction of a second by taking advantage of dynamic linking.

12.1.2 Applications

The rudimentary standalone application support available in DFKI Oz has been replaced by powerful abstractions and command line tools (see *"Oz Shell Utilities"*) to support different aspects of application programming. In fact, a new tutorial (see *"Application Programming"*) is entirely devoted to application programming with Oz and Mozart.

12.2 Syntax Improvements

Mozart implements the language Oz 3, as opposed to DFKI Oz 2, which implemented Oz 2. This chapter summarizes language changes between Oz 2 and Oz 3, of which most are only of syntactical nature.

12.2.1 Conditionals

The `case` keyword used to introduce one of two conditionals: the boolean or the pattern matching conditional. To adapt to common intuitions, the syntax and semantics have been changed.

Boolean Conditionals The boolean conditional is now written as

```
if E then SE1 else SE2 end
```

If the construct is statement position, the `else` *SE2* part is optional and defaults to `else skip`.

Since the `if` keyword is now used for boolean conditionals, the former (and seldom used) `if` conditional has been renamed to `cond`. There is no `elsecond` to replace `elseif`.

Pattern-Matching The `case` *E* `of` ... `end` conditional retains its syntax but changes its semantics. Where formerly logic (dis-)entailment was used to match the value against a pattern, now a series of sequential tests is performed. This makes no difference if the match is entailed. Disentailment, however, may remain undiscovered and the thread block, e.g., in:

```
case f(a b) of f(X X) then ... end
```

Furthermore, the box `[]` separating pattern-matching clauses now also has sequential semantics, and is thus equivalent to the now deprecated, though still allowed, `elseof`.

`elseif` and `elsecase` may still be freely intermixed within `if` and `case` conditionals.

12.2.2 Functors

To accomodate modular application development, a module system has been designed. The language itself supports the definition of *functors*, from which modules can be obtained via linking.

12.2.3 Exceptions

The construct `raise` *E1* `with` *E2* `end` has been removed. This was an experimental feature that has been found to be rarely used.

12.2.4 Keywords

New Keywords Due to syntax changes, Oz 3 has the following keywords, which thus cannot be used as unquoted atoms any more:

`at`	`cond`	`define`	`export`
`functor`	`import`	`prepare`	`require`

Removed Keywords The following keywords have been returned atom status and do not count as keywords any more:

```
with
```

12.2.5 Core Expansion

The core expansion of Oz 3 as defined in *"The Oz Notation"* does not give core variables (written without backquotes) normal variable status any more, but considers them variables statically bound within a runtime library environment. This means that the used backquote variables are not part of the Base Environment.

This was necessary because the old design compromised language security.

12.3 Base

This chapter documents the changes that have taken place in the Base Environment (formerly Standard Modules) and base language.

12.3.1 Classes with Multiple Inheritance

Multiple inheritance does not provide for automatic conflict resolution. If a conflicting method definition arises in multiple inheritance, the conflict *must* be resolved by overriding the method. Otherwise, an exception is raised.

A conflicting method definition arises if a method is defined by more than one class. For example,

```
class A meth m skip end end
class B meth m skip end end
class C from A B end
```

raises an exception (the old model would silently pick the method from `B`), since both `A` and `B` define the method `m`. The only way to fix this is by overriding `m` when creating class `C`:

```
class C from A B meth m skip end end
```

Features and attributes are handled identically. For a more thorough discussion see Chapter *Classes and Objects, (Tutorial of Oz)*.

12.3.2 The Modules `Class` and `Object`

The modules `Class` and `Object` underwent a major redesign and re-implementation. The redesign became necessary because the old modules compromised both system and application security. Programming abstractions that support common patterns of object oriented programming are described in the module `ObjectSupport` (see Chapter *Support Classes for Objects:* `ObjectSupport`, *(System Modules)*).

12.3.3 Chunks

The procedures `Chunk.hasFeature` and `Chunk.getFeature` are gone. Just use `HasFeature` and `Value.'.'` (see Chapter *Values, (The Oz Base Environment)*).

12.4 System Modules

12.4.1 Search Engines renamed

The engines that used to be available by `SearchOne`, `SearchAll`, and `SearchBest` are now available under `Search.base.one`, `Search.base.all`, and `Search.base.best` (for more information see Chapter *Search Engines:* `Search`, *(System Modules)*). However in the Oz Programming Interface `SearchOne`, `SearchAll`, and `SearchBest` are still available for convenience.

12.4.2 Scheduling support moved

Scheduling support is now provided by the system module `Schedule` rather than `FD.schedule`. See also Chapter *Scheduling, (Finite Domain Constraint Programming in Oz. A Tutorial.)* and Chapter *Scheduling Support:* `Schedule`, *(System Modules)*.

12.4.3 `System.get` and `System.set`

`System.get` and `System.set` have been replaced by more powerful procedures that are available in the module `Property`, which is described in Chapter *Emulator Properties:* `Property`, *(System Modules)*.

12.4.4 `System.valueToVirtualString` and `System.virtualStringToValue`

`System.valueToVirtualString` and `System.virtualStringToValue` are now available as `Value.toVirtualString` (see Chapter *Values, (The Oz Base Environment)*) and `Compiler.virtualStringToValue` (see Section *The* `Compiler` *Module, (The Mozart Compiler)*). In particular, `Compiler.virtualStringToValue` is a full featured and stable replacement for the ad-hoc `System.virtualStringToValue`.

12.5 Tools

12.5.1 New Tools

Mozart comes with (improved) versions of the tools that came with DFKI Oz. Additionally, it has:

- a profiler, described in *"The Mozart Profiler"*, and
- a source-level debugger called Ozcar, described in *"The Mozart Debugger"*.

12.5.2 Compiler

The compiler has been reimplemented in Oz. This means that an arbitrary number of compiler objects may be instantiated on the same VM. Linguistic reflection is thus fully supported through an API that offers unrestricted access to the compiler's functionality, documented in *"The Mozart Compiler"*.

12.5.3 Gump

Gump, the frontend generator for Oz, is no longer a stand-alone tool that must be invoked on a file, but is closely integrated into the Oz compiler. It is now sufficient to set a switch:

```
\switch +gump
```

and full Gump functionality as described in *"Gump–A Front-End Generator for Oz"* is available within the language. Furthermore, support for Gump under Windows has been greatly improved.