C++ Development for OOo: Tricks of the Trade

Thorsten Behrens
StarOffice/OpenOffice.org
Sun Microsystems

This Talk:

- this is about OOo core programming.
- the core: that's where the vast majority of both code and functionality is located, and it's is where to know your way around when things go wrong (i.e. you need to fix a bug or you want to add a feature that affects core functionality).

What's the Matter With OOo?

- OOo is huge: ~ 5.2 MSLOC, 4.6 MSLOC C++ (89%), ~23,000 C/C++ files (early 2.6 kernel: ~4.3 MSLOC)
- at its core, OOo is classic C++: classes & inheritance, spread across a lot of shared libraries.
- that gives us: a highly coupled beast, that takes a day to build from scratch

Coupled - In What Way?

- Separate compilation units ('cxx'-files) are highly dependent on other files:
  - e.g. header file vcl/window.hxx: 2552 (for 7127 of relevance) files that directly or indirectly depend on it
  - about 1/3rd of OOo would need recompilation, when vcl/window.hxx changes.

So, What's the Matter Again?

- nobody seriously wants to wait 3 hours to recompile after a single change
- in contrast to the scholarly focus on encapsulation (which is about logical dependencies), a large C++ project like OOo also has to care about physical dependencies:
  - transitive closure of OOo's dependency graph: 1,950,117 edges (from 7129 active compilation units), i.e.~274 mean dependent files per compilation unit

Break Dependencies, Brute-Force

- OOo is broken down into a bunch of modules, where each module ideally contains a delimited, cohesive area of functionality (e.g. VCL: GUI platform abstraction; SW: Writer)
- each module provides a public interface via "exported headers": during build time, each module "delivers" headers to the global solver directory, which makes those headers visible to other modules.

Break Dependencies (cont.)

- switching off dependencies on headers taken from solver (by undef-ing MKDEPENDSOLVER) leaves only intra-module dependencies: now only ~42 mean dependent files per compilation unit
- this leads to the notion of "compatible" vs. "incompatible" changes
  - “compatible”: one does not need to recompile other modules (by hand)
  - “incompatible”: some, or all of the higher modules need rebuilds

Break Dependencies, the C++ Way

- changing implementation should not require recompilation in other modules
  - i.e. a class should be truly *insulated*
- in a first step, reducing dependencies can be achieved via
  - use forward decls instead of header inclusion wherever possible (ptr or reference to given type, return value)
  - keep enums at the classes that use them (instead of putting them into a central enums.hxx)
  - avoid default arguments - they need full definitions, not only forward declaration
  - ...

Break Dependencies (cont.)

- aforementioned list helps, but still leaves class internals exposed to client code
- now, true insulation can be achieved by
  - pimpl idiom (or handle-body idiom)
  - abstract interface (protocol class) plus factory

What's a Pimpl, Anyway?

```
class MyClass
{
public:
    someMethod();
    someOtherMethod();

private:
    struct MyClassImpl;
    MyClassImpl* mpImpl;
};
```

Pimpl Vs. Abstract Interface

- performance: pimpl is slightly faster than virtual functions calls on a protocol class
- pimpl provides concrete classes, from which one can derive and that can be freely constructed (even on the stack)
- protocol classes also remove link-time dependencies (see UNO)
- but for both:
  - overhead prohibitive, e.g. for low-level, frequently used classes with simple getter/setter methods
  - when passing pimpled objects by value, consider to also COW (Copy-On-Write) them.

Also Bad (When Used Large Scale)

- non-local statics
- passing user-defined types (class, struct, union) by value
- COWed mass objects that need to be thread-safe
- short and float at interfaces
- automatic conversions
- code is not warning-free
- not being const as const can.
- using exception specifications
- ...

What's Out There to Help You?

- boost::scoped_ptr for RAII
- boost::shared_ptr/boost::weak_ptr for ref counting
- comphelper::servicedecl for UNO lib boiler-plate avoidance
- o3tl::cow_wrapper, if you've pimpl already, and need COW on top
- rtl::Static for providing on-demand created static objects

Gdb or When All Else Fails

- use most recent version (CVS)
- hack unxlngi6.mk to define -ggdb instead of -g
- if gdb gives 'incomplete type' on classes that your code uses, try setting 'envcflags=-femit-class-debug-always' and rebuild the file(s) in question.
- exercise the stuff you want to debug once, and only then attach gdb to the running office ('gdb soffice.bin $PID').
  - that way, you work-around gdb's current inability to reliably set deferred breakpoints in demand-loaded libs...

Development Tools

- (X)Emacs
- Vim
- NetBeans/Eclipse
- MSVC
- UML
  - argo
- build
  - for Emacs & vi: wrappers
- Testing
  - /testshl2cppunit
  - delta
- Code analysis
  - cvsstat gcc-xmloink sloccount cpd bonsailxr cscope doxygen
  - gcc dump options
  - sourcenav
- IFace design
  - DialogDump

Recommended reading/links

- OOo's list of literature
- Large-Scale C++ Software Design by John Lakos
- C++ Coding Standards by Herb Sutter and Andrei Alexandrescu
- Watch out for an update to OOo's coding guidelines

Q&A

Thorsten Behrens
thorsten.behrens@sun.com