C++ and Real-Time Programming

B.E. Glendenning

1999-Apr-13

Outline

- Is Object-anything any good? A tale of two papers.
- C++ as a better C - the non-controversial bits.
- The other bits.
- What about performance?
- RT/Embedded issues.
- Advice.

Does OO Sync with How We Think?

IEEE Software, May/June 1998

- Case study from two ~50kloc systems (program analysis tools), one written in C, the other C++. Detailed maintenance and development statistics were kept.
- Same highly skilled staff, C++ version was “true *ab initio* OO designed parser”
- Initial C++ defect rates 25% higher (although both were good: 2.9/kloc vs. 2.5/kloc).
- 60% of all fixes for C took less than 2 hours, only 30% of C++ fixes did
  - The “time to fix” graph was shifted: all fixes of any complexity took longer in C++
  - Overall, 1341 hours in fixing 94 C++ defects, 375 hours to fix 74 C defects

Does OO Sync with How We Think? (2)

- Other than GUI building, ACM surveys have not shown reuse to be a strong success with C++
- Claim: OO is not a good match for human reasoning - inheritance and polymorphism use long-term memory rather than short-term memory
  - » Encapsulation “at least partially fits how we think”
- Possible problems with the paper
  - » Entanglement of C++ & OO
  - » First C++ project?
  - » Relatively small systems
  - » C++ system did more than the C system in the same size?

Impact of Ada and Object-Oriented Design in the Flight Dynamics Division at Goddard Space Flight Center

SEL-95-001

- Evaluation of ~20 systems implemented in ADA over ~10 years.
- “Use of Ada and OOD in the FDD resulted in
  - » Increased software reuse by 300%
  - » Reduced system cost by 40%
  - » Shortened cycle time by 25%
  - » Reduced error rates by 62%”
- Earliest projects had performance problems and development overruns
- OO FORTRAN had reuse, but not cost, improvements
- Verbatim reuse improves productivity ~5x.
- Political failure - workforce lukewarm to Ada

- Question: Is Ada better than C++, or is experience the difference?

C++ as a Better C

- Some changes have already made it into C (void, prototypes)
- Declare variables where needed, especially temporary variable
  - `for (int i = 0; i<100; i++) {...`
- Inline functions, and enumerations, instead of macros
- `const`
- `bool`

Big Features

- Object-Oriented features
  - » Classes (encapsulation)
    - – construction/destruction
    - – disciplined access to data
  - » Inheritance (extension)
  - » Polymorphism (run-time selection)
- Generic programming
  - » template classes, functions, member-functions, …
- Exception handling
- Standard library
- Missing
  - » Persistence
  - » Garbage collection

Performance

- Pointing aliasing can slow you down ~30% compared to FORTRAN
- Operator overloading can cause severe performance degradation
  - » temporaries creation/destruction, memory allocations, …
  - » inefficient use of registers
  - » Can be overcome by “template metaprogramming” techniques, but requires modern compilers
- Good OO practice usually has more function calls
  - » Generally not an issue - the small function calls can be inlined or don’t take much aggregate time
- Virtual function vs. function calls is a non-issue
  - » it’s very cheap, and if you are using virtual functions you have to do the `if` somewhere anyway

Performance (2)

- Optimizers usually have more trouble with C++
- Exceptions usually have some run-time overhead

Real-Time/Embedded issues

- Temporaries can cause memory fragmentation
- Some care has to be taken to make an object ROMmable.
- WRS estimates that only ~5% of projects implemented with VxWorks use C++
  - » C++ use heavy in telecommunications and banking
- C++ programs tend to use more stack and dynamic memory
  - » Many devices will be <<1MB for years to come
- Embedded C++ subset has been defined:
  - » http://www.dinkumware.com/embed9710.html
  - » No templates, exceptions, multiple inheritance, RTTI
  - » simple library
  - » Much like C++ circa 1990
- Smart pointers/arrays can greatly reduce incidence of memory leaks and problems

Advice

- Stick with C unless
  - you have a lot of complexity to manage; or
  - you already have experienced C++ developers; or
  - there is a large code base you know you can reuse
- Consider using the embedded C++ subset and style guide
- avoid idioms that create temporaries
- In any event, be wary of using exceptions in multi-threaded/multi-tasking code