

The C Language Mapping –

Does it meet the needs of Embedded Real-time Applications?

S. Aslam-Mir Ph.D.
Senior Software
Architect



Agenda

- Motivations
- Key highlights of the mapping
- Some possible advantages
- Specific mapping issues/gotchas
- Conclusions

Motivations

1. DSP embedded RT developers are entering the era of OO and avoid C++ because of their severe resource constrained environments.
2. These DSP power users see clear benefits in using CORBA, but in a different way and on their terms
3. Very large proportion of the hard core embedded engineering market is still C and assembler based. – bit and pointer wizards
4. Real-time embedded engineers are vary large C community.
5. Large assembler community wraps its stuff into C/C++ callable units.
6. A comprehensive C mapping therefore has clear benefit

Some other motivations

1. Use with CORBA services and other C++ and Java apps running elsewhere.
2. Use with minimal services in highly embedded DSP envs
3. Use for highly demanding ultra-low footprint real-time environments
4. Run over minimalist low footprint embedded RTOSs
5. Large assembler community wraps its stuff into C/C++ callable units.
6. A comprehensive C mapping therefore has clear benefit

Key highlights of the mapping

1. Though C is not OO in nature the mapping tries to create that illusion to some respect – good and bad
2. Type safety is void *
3. Work was done prior to minimum CORBA and Real-time CORBA – so perhaps a small revision might be in order
4. Very clear and simple to understand – most IDL construct have natural intuitive mappings into C – e.g interfaces map to C structs as do IDL structs. Unions to unions, and so on.
5. Complicated constructs like Sequences of complex types and Anys are more ‘sinister’ and cumbersome to manage
6. Recursive IDL like struct that contains a sequence of itself <type> is intuitively mapped to its struct in C only there are issues with its memory management.

Key highlights of the mapping (contd)

7. Under specified in areas e.g.
 - ❑ valuetypes, box types, and abstract interfaces possibly because these didn't exist at the time the specification was completed
 - ❑ Certain rules w.r.t memory management of Sequences of variable types
 - ❑ Memory management of complex variable types.
8. Server side mapping does cover the POA
9. POA mapping is somewhat complex in that it tries to reflect the C++ mapping to some extent (except that it avoid the multiple inheritance)
10. Server side mapping is ideal for a 'big footprint' ORB server !

Key highlights of the mapping (contd)

11. Writing and configuring Impl functions is simple lining up of function pointers – but is cumbersome for endusers – this is a bad thing for those hardened embedded C programmers when the numbers of modules/interfaces/ops rises in any substantial project

Specific mapping issues/gotchas

1. Non OO nature of C
2. Exception handling – stack unwinding and memory clean-up
3. Memory management (footprint control)
4. Type safety – run-time and compile-time
5. CORBA Anys
6. Dynamic Anys
7. CORBA Sequences – variable types and recursive types
8. CORBA Unions containing complex IDL
9. Server side mapping
10. Inheritance – multiple inheritance vs. low footprint

Conclusions

1. At face value a straightforward mapping with a few major hidden gotchas
2. Exception handling, and memory management require extreme care in implementation – embedded programmers are used to this
3. Ideal language for very low footprint (very very low) – embedded developers like that -
4. High performance is easier to achieve– without dipping into assembler steroids ! – real-time predictability is easier to achieve
5. Unusual optimizations in the marshalling engine are possible using C mapping – again no assembler or even having to switch to highly optimizing C compilers !
6. Awesome portability !
7. Ultra-fine grained memory and thruput control – embedded developers are given some leverage in writing and controlling use of memory themselves – implied in the spec