The History of the OMG C++ Mapping

Object Interconnections

Douglas C. Schmidt
Steve Vinoski
September 2000

Introduction

Welcome to the next generation of our Object Interconnections column, which focuses on topics related to distributed object computing and CORBA. As many of you may know by now, the C++ Report magazine no longer exists. Thanks to the efforts of Herb Sutter and Marc Briand, our column will continue under the auspices of the C/C++ Users Journal (CUJ). We look forward to working with the good folks at CUJ and hope that you enjoy our column.

One of the most important original goals for our "Object Interconnections" column when we started writing it over five years ago was to appeal to C++ practioners. As practicing distributed systems programmers ourselves, we realize the importance of lucid examples. Therefore, we've always tried to include working C++ code to illustrate our column's topics.

In looking back over our CORBA examples from past columns (all of which are available on-line at both Steve's home page <`http://www.iona.com/hyplan/vinoski/`> and at Doug's home page <`http://www.ece.uci.edu/~schmidt/`>), we realized that we never took the time to explain the principles of the OMG IDL C++ Language Mapping [1] on which our code examples are based. Of course, we've always included explanations of what our examples do, but we've never stepped back and explained why the mapping is designed the way it is. With the number of C++ programmers, and specifically CORBA C++ programmers, continually growing, we feel that it's time we explain these principles. This is especially important since new C++ programmers tend to learn the Standard C++ library, and they wonder why the OMG IDL C++ Language Mapping doesn't make use of it.

In this column we travel back in time and discover where the OMG IDL C++ Language Mapping came from and learn why it is the way it is. If you understand how the mapping came into being, you'll find it easier to understand the design principles behind it. We don't intend to dive into gory details about the mapping — those are covered elsewhere [2] — but instead will focus on the forces that drove the mapping's design. In our next column, we'll explore some ideas for hypothetical alternative mappings of OMG IDL to C++ that make use of Standard C++ features (but please note that there are currently no efforts within the OMG to define such a mapping, nor do we necessarily condone the creation of any official efforts to do so).

Why a C++ Mapping?

Versions 1.0 - 1.2 of the OMG CORBA Specification [3], which existed during the years of 1991-1995, contained a language mapping for only the C language. Unfortunately, given CORBA's object-oriented (OO) nature, writing CORBA programs in C was tedious and error-prone. However, given that CORBA was strongly influenced by C-based RPC systems such as the Apollo Network Computing System (NCS), standardizing a C language mapping first was easiest for those blazing the CORBA trail.

Even as the first versions of CORBA were being published in 1991, however, the need to create a standard C++ language mapping for OMG IDL was obvious. Most ORB development work in the industry was being done in C++. Companies such as Hewlett-Packard, HyperDesk, IBM, IONA Technologies, and Sun were busy developing not only their own ORBs, but also their own proprietary C++ mappings. Early ORB developers hoped that by providing a natural mapping of CORBA OO semantics onto the OO aspects of C++, they could make CORBA easier to use without sacrificing performance. In hindsight, the focus on

performance was extremely important to CORBA's ultimate success, particularly in performance-driven domains such as embedded and real-time systems. By and large, the network programming community of the day used C, and they would never have moved to the higher levels of abstraction offered by CORBA unless the performance impact was negligible.

Round 1 of the CORBA C++ Mapping Saga

In 1992 the OMG issued a Request For Proposal (RFP) for a standard OMG IDL C++ language mapping, and initial proposals were submitted around mid-1993. The primary submissions consisted of a joint submission from Hewlett-Packard and Sun along with individual submissions from HyperDesk and from IONA Technologies. The HP/Sun submission and the IONA submission were similar in that both were focused on performance, and both required significant attention to memory management from application developers. The HyperDesk mapping was the opposite; its primary goal was to hide low-level memory management complexity from developer.

As the submitting companies worked together into the autumn of 1993 to try to merge their submissions and reach consensus around a single mapping submission, several changes occured. Sun and IONA joined their submissions together, while HP dropped their support for the IONA/Sun submission because the ease-of-use features of the HyperDesk mapping appealed to them. For political reasons, however, HP stopped short of joining the HyperDesk camp (at the time, HP and Sun were involved in a joint ORB development program), and instead assumed the position of negotiator within the submitters' group to help drive consensus.

Unfortunately, by December of 1993, the C++ RFP submitters had still not reached consensus on a single merged submission. This forced the OMG ORB Task Force (ORBTF) to hold its first-ever technology adoption vote. After listening to HyperDesk and IONA/Sun present each of their mapping submissions, and HP present a critique of both mappings, the ORBTF voted to recommend that OMG Technical Committee (TC) adopt the HyperDesk C++ mapping as the OMG standard.

Between the time the ORBTF voted to recommend the HyperDesk mapping (December 1993) and the OMG TC had to vote for adoption (March 1994), each side lobbied heavily for its respective cause. This was a one-sided affair, though, for several reasons:

- Sun had invested heavily in implementing their C++ language mapping in support of their Spring OS [4], and they weren't keen on having to reimplement it all using the HyperDesk mapping, which they felt was not suited for use in a performance-critical area such as an operating system.

- IBM started to visibly support the IONA/Sun submission, and both Sun and IBM enjoyed tremendous political clout within the OMG. They were able to lobby many TC members to vote against the proposed HyperDesk mapping adoption. IBM was not involved in the C++ mapping submission process initially, but became involved after they realized how important it was to their System Object Model (SOM) effort [5], which was an early OO component model.

- HyperDesk had begun suffering business problems (and as a result would disappear from the marketplace soon thereafter).

Before the OMG TC had a chance to complete its voting, HyperDesk realized that its mapping had no chance of being adopted because of Sun's and IBM's lobbying. Due to this and their mounting business woes, they withdrew their C++ mapping submission, leaving the OMG back where they started, without a C++ mapping.

Round 2 — Let's Try Again

The failure of the OMG C++ mapping standardization process had wide-reaching effects. Naturally, it set back productization plans for multiple ORB vendors, and affected implementation plans of companies intending to build CORBA-based distributed systems in C++. Unfortunately, OMG member companies

weren't the only ones affected by this failure of the standardization process. At that time, CORBA was growing quickly in popularity, and most of its users were C++ users. Long-time C++ advocates and experts, such as Doug Lea and Mark Linton, were among the first to express concern about the failure of the C++ mapping standardization process, because they too were interested in developing portable CORBA-based applications in C++.

While the dust in the OMG settled, Doug Lea, Mark Linton, and Steve Vinoski started looking again at the details of the two C++ mapping submissions. We knew performance was a critical factor for both Sun and for Mark's Fresco project [6], so we wanted to verify Sun's claims that the HyperDesk mapping did not perform well. This turned out to be relatively easy: the reference counting and memory management that the HyperDesk mapping hid from the programmer disallowed both copy avoidance and mutex lock elimination under common circumstances. Some of our simple test progams showed a 4-5x performance difference between the HyperDesk mapping and the IONA/Sun mapping.

Based on our investigation, it was clear that a C++ mapping that didn't perform at least as well as the IONA/Sun mapping would never be adopted as the standard, but at the same time we didn't want to give up on the ease-of-use-characteristics of the HyperDesk mapping. We decided to take a layered approach: the mapping's bottom layer would not perform any memory management, allowing performance-critical systems to optimize management for themselves, while its top layer would provide memory management assistance. An application's use of the top layer would be optional.

We began writing a new draft C++ mapping while we contacted all the remaining OMG submitters and other interested parties to see if we could get them to come to the table and forge a new agreement based on our new C++ mapping approach. Involved in this new round of talks were (in alphabetical order) Digital, Expersoft, HP, IONA, Novell, and Sun. We met several times during the summer of 1994, and in September 1994 we submitted a new C++ mapping proposal to the OMG, which was adopted as the standard in December 1994 and was published as part of the CORBA 2.0 specification in 1995.

Technical Details

With all the newcomers to the submission process, there were additional technical challenges that we had to address. The most significant of these was IBM's insistence that the C++ mapping be binary compatible with the existing C mapping. They required this because SOM was a binary component model based on the C language, and thus all languages supported by SOM had to support their C-based calling conventions to allow seamless inter-language invocations within a single address space. To appreciate the issues involved with this requirement, let's examine the C++ mapping issues for a few OMG IDL data types:

- Interfaces have two aspects to their mapping: the interface itself, and the reference to an object implementing the interface. For the interface mapping, there was unanimous agreement to map each interface into a separate C++ class, with any operations and attributes in the interface mapping to member functions in the class. However, there was disagreement as to how to map object references — should they map to pointers, or to smart pointer objects? The original IONA/Sun mapping took the pointer approach, while the HyperDesk mapping used the smart pointer approach. Obviously, pointers require manual memory management, but our test programs showed that they could be much more efficient than smart pointers, which need to perform thread-safe reference counting behind the scenes. This issue was also divided among the newcomers to the submission process, as SOM used pointers as object references, but Novell wanted to use smart pointers. Our final decision was to allow both: we mapped object references to a "_ptr" type (e.g., the type of object reference for interface **A** is named **A_ptr**), which the mapping implementer could **typedef** to either a pointer or a smart pointer. To make this work portably, we also had to limit the operations that applications could perform on object references, so that they could be successfully implemented using either pointers or smart pointers. For example, conversion to **void***, arithmetic operations, and relational operations (including testing for equality) are not allowed.

- Strings could be mapped either to a **char*** to match the C mapping, or to a C++ class. While mapping to a class seems the obvious choice, both then and now, we chose to map to **char*** for two reasons.

First, in 1994 there was no standard C++ string class — there wasn't even a front-runner that appeared to be a standardization shoe-in — and it seemed that every C++ software package came with its own separate C++ class. We didn't want to add yet another string class to the existing mess. Second, nobody could come up with a practical way to map strings to a class while still allowing seamless inter-language calls in SOM. Thus, IBM insisted on a mapping to **char***.

- There were also two choices for mapping structs: either to a plain C-like struct,i.e., what the C++ standard calls a "plain old data structure" (PODS), or to a C++ class. Unlike strings, the best mapping for a struct really depends on its member data. For example, a simple **Point** struct representing two- or three-dimensional graphical coordinates is best mapped to a PODS with public data members holding x, y, and z coordinate values, especially given that such a mapping enables simple static initialization. From a memory management point of view, on the other hand, a complicated struct holding strings, sequences, and arrays as members might be better off if it were mapped to a C++ class with accessor and mutator functions for the data members. We chose a middle ground: IDL structs map to C++ structs, but all their data members are self-managing. For example, a string member of a struct maps to a class type that manages the string's memory upon struct creation, assignment (including direct assignment of the string member itself), and destruction. Also, structs with a "fixed length" — i.e., their sizes of all their data members are fixed and they do not reference dynamically-allocated memory — are mapped to PODS and do not have user-defined constructors or destructors in order to allow static initialization.

- Arrays map to C arrays rather than to some kind of C++ array class. As in the case for strings, this decision was made because of SOM's C compatibility requirements.

OMG IDL operations and attributes allow applications to pass arguments both in and out of functions, and allows return values as well. For example, if a CORBA client invokes an operation that returns a string, the operation implementation will return a **char***. Obviously, somebody has to take responsibility for eventually freeing the memory pointed to by the **char***, otherwise the program will have a memory leak. It's also obvious that the client, not the server, must take that responsibility because the client can't hold a pointer to memory across the network on the server's machine (CORBA is not, and does not require, a distributed operating system). Unfortunately, saddling the client with the responsibility of remembering to always free the returned string is somewhat of an imposition. It's all too easy for the application developer to forget to free the string, or even to forget to store the return value into a variable to keep it around for later deletion.

To allow the application developer to choose whether to manage memory directly or use more automatic means, the C++ mapping top layer supplies memory management helper classes called "_var" classes. These types are much like the Standard C++ **auto_ptr** type, and they're named by adding the suffix "_var" to the original type name (e.g., the "_var" type for an interface named **A** is named **A_var**, and the "_var" type for the string type is named **CORBA::String_var**). Assigning a value requiring memory management to a "_var" type causes the "_var" type to assume ownership for the memory, and to free it in its own destructor. The use of "_var" types, while completely optional, can greatly ease the memory management responsibilities for applications when compared to using just the bottom layer of the C++ mapping.

Why Not Update the C++ Mapping?

Everything we've discussed so far indicates why the OMG IDL C++ Mapping was designed in a particular way, but that was back in 1994! New CORBA C++ users often ask why the C++ mapping hasn't been revised to take advantage of Standard C++, with the most frequently asked question being, "Why aren't IDL strings mapped to the Standard C++ string class?" They also frequently want to know why IDL container types, such as sequences and arrays, are not mapped to STL container classes.

Unfortunately, modifying the C++ mapping to take advantage of Standard C++ is a business problem, not a technical one. Both ORB suppliers and ORB customers have spent millions of dollars on applications that are written against the current mapping, and few are eager to revise code that is already working and in

production. While making the technical changes to the C++ mapping necessary to use Standard C++ would be fairly trivial, the business case for doing so is rather weak — there's simple too much existing code in place that would either require legacy C++ mapping support, or would require a total rewrite. Neither approach is appealing to existing users or vendors. The only viable alternative is to somehow allow the existing C++ mapping to exist in concert with an extended mapping that took advantage of STL-related features.

One argument for a revised C++ mapping is the claim that the current mapping is hard to use. Some claim that even with the "_var" layer assisting with memory management, the mapping is still too hard to use. We feel that this is an unfair characterization of the C++ mapping, as we believe its memory management rules to be consistent and straightforward once you understand the principles and patterns behind them, which are thoroughly explained in [2].

Concluding Remarks

This column provides an overview of the history of the OMG IDL C++ Language Mapping. It's clear that, like most projects and standards, politics and personalities had a profound influence on the design of the C++ mapping. Still, we believe the mapping provides a nice balance of performance and ease of use, especially given the compatibility requirements imposed on the mapping's design by several OMG members.

In our next column we intend to explore what a revised C++ mapping might look like if it were to take advantage of Standard C++ constructs. And we promise to include some C++ code examples next time!

References

1. Object Management Group. IDL C++ Language Mapping. http://www.omg.org/technology/documents/formal/c++.htm, 1999.
2. Michi Henning and Steve Vinoski. Advanced CORBA Programming with C++. Reading, MA: Addison Wesley, 1999.
3. Object Management Group. CORBA 2.3.1 Specification. http://www.omg.org/technology/documents/formal/corba_2.htm, 1999.
4. J. Mitchell et al. "An Overview of the Spring System". In Proceedings of Compcon Spring 1994. IEEE, February 1994.
5. Danforth, S., P. Koenen, and B. Tate. Objects for OS/2. New York: Van Nostrand Reinhold, 1994.
6. Mark Linton and Chuck Price. Building Distributed User Interfaces with Fresco. In Proceedings of the Seventh X Technical Conference, pages 77-87, Boston, MA, January 1993.