



IEEE 1516 Compliance – Will the Real C++ API Please Stand Up?

By Len Granowetter
MÄK Technologies

Given that the HLA has been an approved IEEE Standard for nearly four years now, it seems reasonable to assume that the term “HLA-compliant” should be unambiguous. Either an RTI complies with the HLA Interface Specification defined by document known as IEEE 1516.1-2000, or it doesn’t. Unfortunately, the reality is more complicated than that. This white paper tries to answer some of the common questions people have about the various incarnations of the C++ API for the IEEE 1516 version of the HLA, and attempts to clear up some of the confusion that has been generated.

Does IEEE 1516.1-2000 actually define header files that make up the C++ API for HLA?

Yes. The *body* of the HLA Interface Specification document describes the semantics of the various RTI service calls in a language independent way. It dictates the functionality that an RTI must implement without specifying precise syntax. However, the document also contains three Appendices that describe exactly how the text descriptions of the services are to be mapped to the syntax of various programming languages (C++, Java, and Ada). In the case of C++, the Appendix consists of a set of header files that are to be adhered to by all RTI implementations if they wish to be considered compliant with the IEEE 1516 version of HLA.

If a standard set of header files were *not* included in the IEEE 1516.1-2000 document, then different RTI vendors might choose different syntactic interfaces to the same set of functionality. For example, the body of the Standard dictates the names of the various RTI service calls, but the C++ API Appendix specifies that these should be provided as member functions of a class (as opposed to global functions), and specifies what the name of that class should be. The body of the Standard dictates a set of enumerated types that the RTI must provide (TransportationType, OrderType, etc.), but it’s the C++ API that specifies that these should be provided as static instances of C++ classes, as opposed to C++ “enums”, or preprocessor macros (#defines).

So, if the IEEE 1516 Standard precisely defines a C++ API, why isn’t that the end of the story?

Unfortunately, the C++ API that was published as an Appendix to IEEE 1516-1.2000 was “broken” in two significant ways:

First, it contained a variety of bugs or omissions that made it unimplementable. That is, it is simply not possible to implement an RTI that actually adheres to the C++ API dictated by the

MÄK Technologies

www.mak.com • info@mak.com • 1-617-876-8085 x2

IEEE 1516.1-2000. (How did this happen? Partly because IEEE balloting and approval occurred before anyone had actually tried to implement an RTI against it.)

Second, the C++ API was not complete. Because the API designers wanted to allow RTI developers the flexibility to choose different implementations for types like Handles, the API called for RTI vendors to “fill in the blanks” with some implementation-specific header files. This means that even if the “bugs” in the API were corrected, a federate developer could not successfully compile a federate against the set of header files dictated by the IEEE 1516 standard without the API being “completed” by an RTI vendor. Because there is no standard way in which RTI vendors are supposed to “fill in the blanks”, different RTI implementations will, in effect, be implementing different C++ APIs.

The fact that the original IEEE 1516 API chose not to *fully* specify a C++ API means that different RTI implementations are not Dynamic-Link-Compatible. That is, when a federate is compiled, it is locked into the choice of a particular RTI implementation. In order to switch to a different vendor’s RTI, the application must be recompiled against the new vendor’s API, and re-linked. While this may not sound like a significant burden, it means that a federate developer or tool vendor must maintain a separate version of each federate for each RTI implementation that they wish to support. The end-user is not free to choose an RTI implementation that is appropriate for a federation simply by swapping in a different RTI’s DLL. Instead, he must go back to the federate developer or tool vendor to ask that the federate be rebuilt specifically for the chosen RTI.

What is the DoD Interpretations Document?

When various RTI vendors first started implementing RTIs to the IEEE 1516 set of HLA Standards, it became obvious that there were a variety of problems with the Specifications. The list of issues to be addressed included not only the bugs in the language-specific APIs, as described above, but also a variety of ambiguities in the body of the Interface Specification.

DMSO convened a series of meetings among representatives of government, RTI vendors, and federate developers to address various problems with the IEEE HLA Specifications. The result of these meetings was a DoD Interpretations Document that amended the IEEE 1516 Standard. The document has gone through a couple of revisions, and is maintained by DMSO.

The effect of the DoD Interpretations document on the C++ API went beyond merely fixing the bona-fide bugs in the API that was originally published as part of IEEE 1516. The document also dictated various other ease-of-use improvements to the API. For example, all of the RTI classes were moved inside of a namespace, and the name of every class in the API was changed, to remove the prefix “RTI_”.

What is the Dynamic-Link-Compatible C++ API for IEEE 1516?

The amendments described by the DoD Interpretations Document created an API that was implementable, but not in a way that avoided the “vendor-lock” problem. In order to address this second problem with the original IEEE 1516 C++ API, the Dynamic-Link-Compatible HLA API Product Development Group (PDG) was formed within SISO.

All of the leading commercial RTI vendors seemed to agree that it was a good idea for the HLA to have a fully defined C++ API that enables Dynamic-Link-Compatibility, and all of them devoted significant resources to the development of this new C++ API for IEEE 1516. In fact, the official Drafting Group consisted of: Len Granowetter from MÄK, Bjorn Moller from Pitch,

and Keith Snively from the RTI NG team. Major contributions also came from Doug Wood of MÄK, Mikael Karlsson of Pitch, and Roger Wuerfel of the RTI NG team, as well as Bill Helfinstine, representing Lockheed Martin's STOW RTI team. In September 2004, the Dynamic-Link-Compatible (DLC) C++ API for IEEE 1516 was successfully balloted by the SISO PDG, and in October, 2004, it was approved by the SAC as an official SISO Standard.

It should be noted that the DLC C++ API for IEEE 1516 was developed in a way that still supports the original API's goals of allowing different RTI vendors to choose different implementations for classes like handle. The new API, however, hides those implementation differences from federate code.

Which version of the IEEE 1516 C++ API is being used for verification testing?

DMSO is currently allowing RTI implementations to be verified against *either* the SISO Standard Dynamic-Link-Compatible C++ API, *or* the C++ API defined by the DoD Interpretations Document (Version 2.0). The IEEE 1516 version of the MÄK RTI implements the SISO Standard DLC C++ API for IEEE 1516, and is now undergoing DMSO verification testing. We hope to be officially verified as IEEE 1516 compliant by early 2005.

So, *none* of the RTI implementations are actually verified against the C++ API contained in the IEEE 1516.1-2000 document?

That is correct. The Pitch RTI's C++ wrapper (their implementation is in Java) implements the C++ API as amended by version 2.0 of the DoD Interpretations document. And again, the MÄK RTI implements the SISO Dynamic-Link-Compatible HLA API Standard. Because the original IEEE 1516 API was "broken", it was impossible to verify RTIs against the original IEEE 1516 C++ API.

What about future of IEEE 1516? What is HLA Evolved?

SISO's HLA Evolved Product Development Group (PDG) is the group that is recognized by IEEE as the Standards Development Organization that is responsible for maintaining, changing, and re-affirming the IEEE 1516 HLA Standards. The HLA Evolved PDG had its kick-off meeting at the Spring SIW in April 2004.

Over the coming year and a half or so, the HLA Evolved Group will be updating the HLA Standards based on lessons learned since HLA first became an IEEE standard in 2000. According to the group's current schedule, an improved version of HLA should become the next official version of the IEEE 1516 Standard by early 2006.

Reed Little of Carnegie Mellon's Software Engineering Institute, who was the lead editor of the original IEEE 1516.1-2000 document, is again the leader of the Drafting Group responsible for updating the Interface Specification. The Drafting Group includes representatives from all of the leading RTI vendors – MÄK, Pitch, and the RTI NG team – including all of the editors of the Dynamic-Link-Compatible HLA API Standard.

In September 2004, the HLA Evolved PDG voted to base the C++ API for HLA Evolved on the SISO Standard Dynamic-Link Compatible C++ API. In addition, they voted to incorporate all of the semantic clarifications contained in the DoD Interpretations Document into the body of the updated HLA Interface Specification.

Why did MÄK choose to implement the Dynamic-Link-Compatible C++ API for IEEE 1516?

When it came time for us to decide whether to implement the C++ API described by the DoD Interpretations Document or the SISO Standard Dynamic-Link-Compatible C++ API, the DLC was the obvious choice for two main reasons:

First, we wanted to encourage Dynamic-Link-Compatibility among different RTI implementations. We think that the HLA is most powerful if end-users are free to choose the RTI implementation that best meets their needs, rather than being forced to use the RTI implementation that the federate was originally compiled against. If various RTI implementations all implement the DLC API, then federates can easily switch among different RTI implementations, and RTIs can compete based on their merits – Performance, Ease of Use, Portability, Robustness, Extensibility, and Support.

It's true that we are not Dynamic-Link-Compatible with RTIs that implement the DoD Interpretations version of the IEEE 1516 HLA. But we could not have been compatible with them, even if we *had* implemented the DoD Interpretations version of the API. (Again, that's because that version of the C++ API requires RTI vendors to “fill-in-the-blanks” with implementation-specific header files – a design that precludes Dynamic-Link-Compatibility).

By implementing the DLC API, we are allowing users to create federates that are not “tied” to our RTI implementation. If and when other RTI vendors choose to also implement the SISO Standard DLC API, end-users will be able to switch to those other implementations just by pointing at the new DLLs.

Part of the reason why we are so against “vendor-lock”, is that in addition to being an RTI vendor, we are also an HLA tools vendor. It would be confusing for our customers, and expensive for us, if we had to maintain and deliver a separate version of each of our tools (VR-Link, Logger, Stealth, PVD, VR-Forces, Gateway), for each RTI implementation we would like to support. We want to be able to deliver a single version of each tool, and allow our customers to decide whether to use it with our RTI or someone else's.

The second reason we chose the DLC API is that it represents the *future* of HLA. The DLC API has already been approved as a SISO Standard, and through the HLA Evolved process, the DLC API has been chosen as the basis for the updated version of the IEEE 1516 Standards next year. Rather than providing support for the DoD Interpretations version of the API, and then shortly switching to HLA Evolved (which will look similar to the DLC API), we thought it was both in our interests, and the interests of our customers, to go directly to the DLC API.