The new CERN Controls Middleware

A Dworak[1], F Ehm[1], P Charrue[1] and W Sliwinski[1]
[1] CERN, CH-1211 Geneva 23, Switzerland

E-mail: andrzej.dworak@cern.ch

Abstract. The Controls Middleware (CMW) project was launched over ten years ago. Its main goal was to unify middleware solutions used to operate CERN accelerator complex. A key part of the project, the equipment access library RDA, was based on CORBA, an unquestionable middleware standard at the time. RDA became an operational and critical part of the infrastructure, yet the demanding run-time environment revealed shortcomings of the system. Accumulation of fixes and workarounds led to unnecessary complexity. RDA became difficult to maintain and to extend. CORBA proved to be rather a cumbersome product than a panacea. Fortunately, many new transport frameworks appeared since then. They boasted a better design and supported concepts that made them easier to use. Willing to profit from the coming long LHC shutdown which will make it possible to update the operational software, the CMW team reviewed user requirements and in their terms investigated eventual CORBA substitutes. Evaluation of several market recognized products helped to identify the most-suitable middleware solution: ZeroMQ. This article presents the results of the evaluation process, the proposed design and functionality of the new system as well as the plan of its integration with the currently deployed system.

1. CERN Controls Middleware

The Controls Middleware (CMW) project was launched at CERN over ten years ago. Its main goal was to unify middleware solutions used to operate CERN accelerators. Many software components were developed, among them the Remote Device Access (RDA) [1] library. The main responsibility of the library was to allow communication with servers that operate hardware sensors and actuators. As presented in figure 1, the RDA design corresponds to the Accelerator Device Model [1] in which devices, named entities in the control system, can be controlled via properties. RDA implements this model in a distributed environment with devices residing in front-end servers that can run anywhere in the controls network. It provides a location-independent and reliable access to devices from control programs. By invoking the device access methods clients can read, write, and subscribe to device property values. Currently over 4000 servers (processes) are deployed, which contain altogether almost 80,000 devices. In total the system gives access to more than 2,000,000 properties/IO points, on which clients may perform read/write operations or monitor their values [2].

1.1. Present implementation

From the beginning there were certain requirements [3] imposed on RDA that drove its implementation: relying only on standards; interoperability with the already existing communication infrastructure at CERN; portable on LynxOS with an old gcc v.2.95 compiler, Linux, Windows, HP-

[1] To whom any correspondence should be addressed.

UX and AIX (only the first three are still supported; LynxOS is being eradicated); C/C++ and Java bindings for client/server libraries; request-reply and publish-subscribe operations on device data. Each call type should provide timeout settings and handling of communication errors. Moreover, complementary, centrally managed services like naming service, reservation service and access control should be supplied.

To facilitate development of the new library it was decided to base it on an already existing, mature product. CORBA [4] was a very popular middleware at that time and fulfilled all the requirements. Thus, it was chosen as the communication layer. The C++ implementation was based on omniORB (currently 4.1.2,) and the Java implementation on JacORB (currently 2.2.4.) RDA library wrapped CORBA, hiding all its complexities and providing a simple to use API. The proposed solution was widely accepted and became an operational and critical part of the infrastructure.

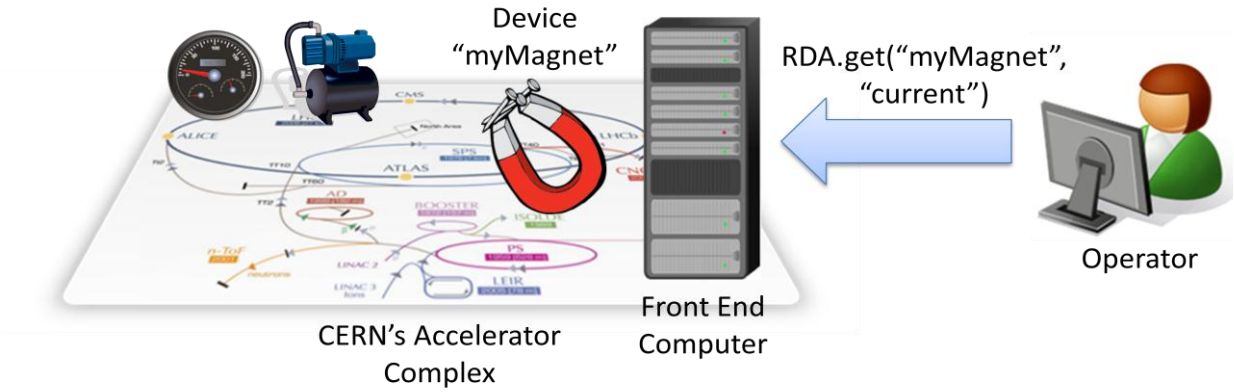

Figure 1. RDA implements the Accelerator Device Model which gives remote access from control programs to servers that operate hardware sensors and actuators.

1.2. Shortcomings of the system

Unfortunately, the demanding run-time environment revealed a few shortcomings of RDA. Accumulation of fixes and workarounds led to unnecessary complexity. Desire to deliver a better, more user-friendly solution led to a general review of the system. Discussions with library clients helped to identify several major issues, of which the most troublesome are the ones directly correlated with CORBA [5]. First, the CORBA standard is inherently huge and complex. Libraries that try to fully implement it have a major memory footprint. This is an issue especially for older front-end computers. It is well understood that RDA as a communication framework uses only a small fraction of the CORBA platform, but users still have to pay the full run-time price. On the other hand, libraries such as JacORB do not implement the full functionality. This leads to mismatches in behaviour of Java and C++ bindings. The struggle to support "asynchronous" operations on top of the synchronous calls leads to unnecessary complexity in the library code and design. Second, the way CORBA is used in RDA leads to multiple data conversions between different representations. This is both time consuming and leads to higher memory usage. Third, CORBA is based on the static Interface Definition Language (IDL), which is difficult to manage and evolve in large, complex environments such as CERN. Finally, the community supporting open-source implementations is shrinking. There is a significant lack of new releases from the major implementations like JacORB, even if the major bugs have been identified and fixed a long time ago.

2. Update of requirements

In view of a 1-year accelerator shutdown at CERN, starting end of 2012, there is a unique opportunity for introducing a major new version of RDA, which should solve all the limitations experienced with

CORBA. On top of that, the period of over 10 years of RDA utilization provided valuable experience on both common and uncommon usage patterns. Discussions with the library users helped to identify a few valid, yet previously not foreseen use-cases. In some cases the missing functionality constituted a serious limitation because of which tricky workarounds had to be applied. Those, in turn, were often the major source of not easy to find bugs.

Collecting the knowledge of the library developers and experience of the users led to an extension of the RDA requirements. For example, on the client API level, support for new data types was requested, as well as an easy to set-up communication policy model, or direct access to communication statistics. A more detailed list of the library features is provided in the fourth chapter.

In context of the new implementation the main concern was to address problems and limitations introduced by the usage of CORBA. Therefore, the CMW team launched the middleware review process, aiming at choosing a new, modern middleware library, to be used for the future version of RDA. In addition to previously specified general requirements the team expected to find a transport library that would provide:

- Consistent implementation for C++ and Java.
- Easy to trace peer-to-peer communication with reliable request/reply and publish/subscribe messaging patterns.
- Synchronous and asynchronous/non-blocking communication.
- Quality of Service (QoS): timeout management, message queues and priorities, various thread management policies.
- Small library size, low memory and resource usage.
- Certain performance characteristics. Analysing the current usage statistics, it was estimated that communication between a typical server on a new front-end computer (Inter Core 2 Duo, 1.5GHz, 1GB RAM, GbE) and a client running on a similar machine should handle:
    - 4000msg/sec req-rep calls, payload = 4Bytes
    - 5msg/sec req-rep calls, payload = 10MBytes
    - publish 400 x 8B to 10 clients, in less than 100 msec
    - publish 30 x 8B to 10 clients, in less than 20 msec
- No, or only a few, external dependencies that can be linked with an application, preferably no need for additional services (e.g. brokers, global servers, daemons.)
- Open source, with a license allowing further redistribution; good documentation, and support from a large active community.
- Simple, easy to learn and use API.

3. Middleware evaluation and prototyping

In terms of gathered requirements the CMW team evaluated several market recognized middleware products. Short descriptions of each of them, comparison with CORBA, as well as results of performed tests were presented during ICALEPCS 2011 conference [6]. As an outcome of the evaluation three libraries were selected for further prototyping: Ice [7], ZeroMQ [8] and YAMI4 [9].

The prototyping process extended the previously performed tests by setting-up more complex communication scenarios. In particular: number of communicating peers was extended and the communicating processes were distributed over several machines with different architectures; some of the peers simulated exceptional or bad behaviour; the data exchanged between entities resembled real Controls data; as in the current operational system RDA is responsible not only for the data transport between remote machines but also for assembling the data, its representation and serialization, the prototypes provided similar functionality of assembling and keeping the data in the same way as RDA.

The mechanism that takes care of the data assembly and serialization is called RDA Data and provides functionality of a collection of named unions – a map which keys are parameter names, and entries are unions which type and data may change dynamically. This functionality is available in YAMI4 out of the box and in CORBA it was implemented using “Any” Type. It was not implemented for Ice. In case of ZeroMQ, which does not provide any functionality concerning data representation

or serialization, a review of over 20 serialization libraries were performed. This led to selection of four libraries: Apache Avro, CORBA serialization module (omniORB for C++ and JacORB for Java,) Google Protocol Buffers and MessagePack. They were used to prototype the RDA Data functionality. The performed review and tests led to selection of MessagePack as not only the most efficient one but also because of direct support for dynamic typing, compact binary serialization, support for all needed data types, operating systems and programming languages, good documentation, product maturity and active community behind.

The middleware evaluation was finished with rewriting and re-execution of the performance tests using the prepared RDA Data mechanism. The obtained results for publish-subscribe messaging pattern (publish 400 messages each containing a 64 bit integer and its name of 8 characters) are presented in figure 2. They correspond to the results where raw data had been transported by the tested libraries. As one may see:

- YAMI4 seems to lack some optimization.
- The current implementation of RDA is well optimized.
- In case of Ice an additional hop through a broker (IceStorm) adds a constant value to the communication time.
- Brilliant automatic buffering of ZeroMQ helps it to considerably beat the opponents.

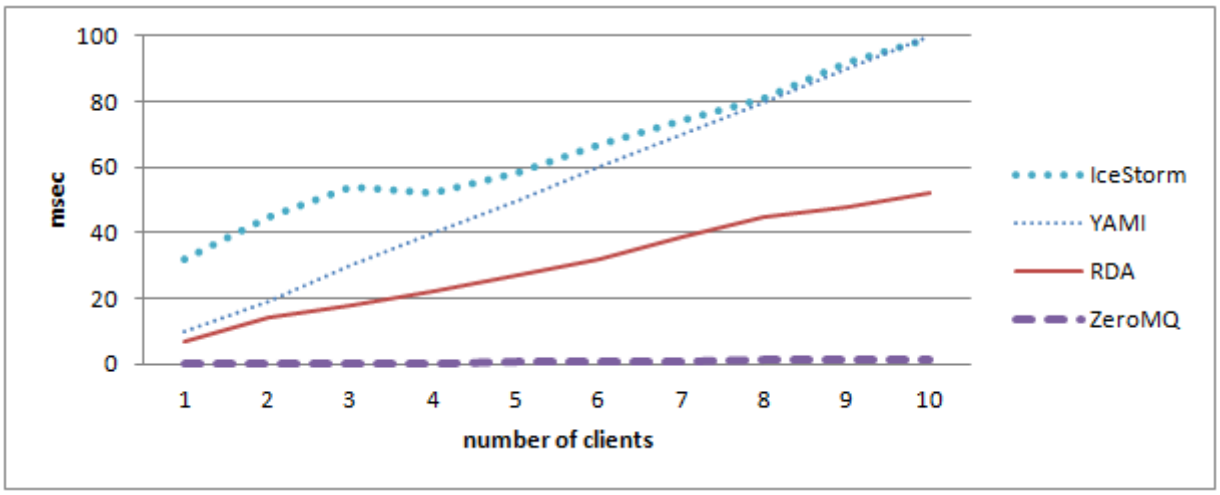

Figure 2. Publish 400 messages each containing a 64 bit integer and its name o 8 characters.

4. User API and functionality of the new system

Interviews with users of the library and direct everyday support provided to them were key inputs that helped to define requirements for the new RDA, called RDA2. It was agreed that the new version should provide similar functionality to the current one, and main changes will consist mainly of exchanging the transport and data representation layers, see figure 3. Thus, the new implementation should fulfill gathered requirements:

- Running on all supported by CERN Controls architectures and operating systems.
- Implementation of the Accelerator Device Model.
- Consistent API and behaviour of C++ and Java implementations.
- Reliable request/reply and publish/subscribe messaging patterns in both blocking and non-blocking modes.
- Number of policies controlling the quality of service (timeouts, priorities, reports on progress/problems.)
- Small library size, low memory and resource usage.
- Performance characteristics as specified for the transport library.

- Support for the current functionality of RDA Data with extensions that would facilitate operations on multidimensional arrays and embedded RDA Data structures. To exchange information between a client and a server, a user does not specify any IDL description of the transferred data. Instead he/she sends and receives the generic RDA Data structure.
- Optimizations and restructuring of the code to minimize data copying both internally and when user calls are executed.
- Internal integration with Controls role based access control (RBAC) service, directory service resolving device names and Controls tracing system.
- Providing functionality of remote management for both client and server processes regardless of programming language used.
- In order to simplify the design, reduce coupling, facilitate testing and future maintenance, codebase should be properly modularized and should not hold any static states.
- Easy to use but not confining API with good documentation which should clearly state how the library should be used in multithreaded environment. In case of the C++ implementation the API itself should advertise who is responsible for resource allocation and deallocation or, if that is impossible or creates constraints, the documentation should clearly explain any ownership ambiguity.

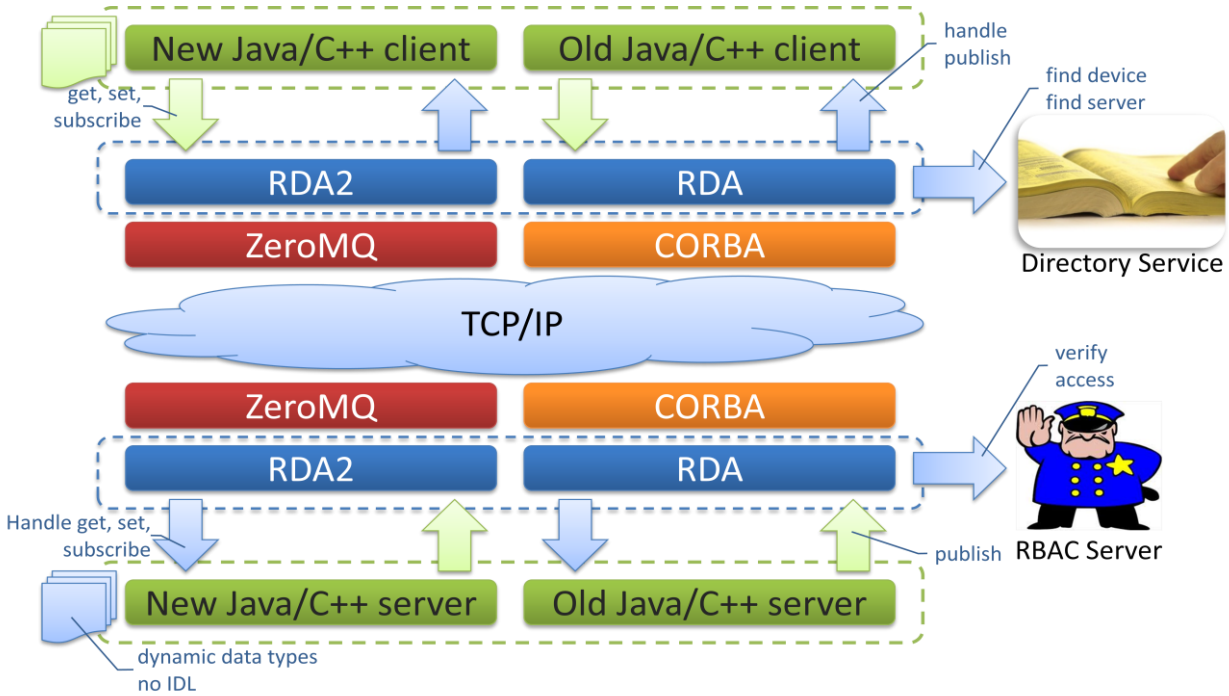

Figure 3. The architecture and functionality of RDA has proved its value, hence the new one resembles the predecessor.

The requirements were used to prepare RDA2 API specification which is currently being verified by the main library users. To facilitate the verification process several code samples were provided which demonstrate the usage of the new API. Along the demo code a series of presentations is prepared to show differences between the old and the new RDA. They will highlight what users gain with the new version and explain how to upgrade software to use the new library.

5. Plans for the future

After acceptance of the new API the implementation of the core system and of the additional services will begin. To follow well established modern software engineering practices, in parallel to the

extension of the library code base a set of unit and integration tests will be developed to ensure proper functioning. The modular design and dependency injection will allow mocking and thus testing of the new functionality in separation from other system parts.

The CMW team plans to finish the implementation phase and start system testing and system integration testing at the end of 2012. Soon after proving the interoperability with the existing operational system, the migration process to the new one will begin.

Even though the long shutdown is planned to last a year and the team will do its best to help the users to move to the new library, probably not all the software using current version will be upgraded during that time. In fact, it is unavoidable that at some point the old and the new systems will have to interact with each other. Hence, to facilitate the migration process and to cover such cases, it is planned to provide bridges between the two systems. Two types of bridges will have to be provided. First type, translating from the old system into the new one, will wrap the new implementation with the old API (AKA the adapter pattern [10].) In such a case an old client communicating with a new server will only have to recompile its sources. Second type, translating from the new system into the old one, will allow using the old protocol through the new API (AKA the bridge pattern [10].) This type of a bridge may be necessary in case of a new client and an old server that was not upgraded.

The team members are optimistic about the presented plans and are eager to start the development phase.

6. Summary

The paper presented the ongoing work of reimplementation of the current CERN Controls Middleware library. With the outline of the future plans this describes a whole system-development life cycle. First, the shortcomings of the current system were identified and listed. This in fact took a few years, but in the end firmly proved that in a long run it is better to rewrite the system from scratch than to patch it. Second, system requirements were updated to match the current user needs. Third, in terms of the new requirements a thorough review of available solutions was performed. This helped to chose a few middleware products on top of which simplified, but providing core functionality, systems were prototyped. Through this process the most suitable middleware solution, namely ZeroMQ, was chosen to handle the transport in the new RDA. Next, an upgraded version of API providing access to the new functionality was designed and presented to the clients for verification. Finally, an outline of testing, implementation and integration processes was presented.

References

[1] N. Trofimov *et al.*, “Remote Device Access in the new CERN Accelerator Controls middleware”, ICALEPCS 2001, San Jose, California, 2001.

[2] Z. Zaharieva *et al.*, “Database Foundation for the Configuration Management of the CERN Accelerator Controls System”, ICALEPCS'11, Grenoble, France, October 2011.

[3] V. Bagiollini *et al.*, “CERN PS/SL Middleware Project, User Requirements Document”, CERN Note SL/99-16(CO), Issue 1 Revision 3, Geneva, Switzerland, August 1999.

[4] OMG CORBA http://www.corba.org/

[5] M. Henning, “The rise and fall of CORBA”, http://queue.acm.org/detail.cfm?id=1142044, 2006.

[6] A. Dworak *et al.*, “Middleware trends and market leaders 2011”, ICALEPCS’11, Grenoble, France, October 2011.

[7] ZeroC Ice: http://www.zeroc.com/

[8] iMatix ZeroMQ : http://www.zeromq.org/

[9] Inspirel YAMI4: http://www.inspirel.com/yami4/

[10] E. Gamma *et al.*, “Design Patterns: Elements of Reusable Object-Oriented Software”