ADASS XIII
ASP Conference Series, Vol. XXX, 2004
F. Ochsenbein, M. Allen and D. Egret eds.

Transparent XML Binding using the ALMA Common Software (ACS) Container/Component Framework

Heiko Sommer and Gianluca Chiozzi

European Southern Observatory, Karl-Schwarzschild-Straße 2, D-85748 Garching b. München, Germany, Email: hsommer@eso.org

David Fugate

National Radio Astronomy Observatory, 1003 Lopezville Rd., NM 87801, USA

Matej Sekoranja

Cosylab Ltd, Teslova 30, SI-1000 Ljubljana, Slovenia

Abstract. ALMA software, from high-level data flow applications down to instrument control, is built using the ACS framework. The common architecture and infrastructure used for the whole ALMA software is presented at this conference in (Schwarz, Farris, & Sommer 2003).

ACS offers a CORBA-based container/component model and supports the exchange and persistence of XML data. For the Java programming language, the container integrates transparently the use of type-safe Java binding classes to let applications conveniently work with XML transfer objects without having to parse or serialize them.

This paper will show how the ACS container/component architecture serves to pass complex data structures, such as observation meta-data, between heterogeneous applications.

1. XML Data by Value

In any distributed software system, it is vital to reduce the number of fine-grained calls accessing remote data. The usual strategy is to pack together some cohesive set of data items and transmit them over the network at once. This has been described in the J2EE Transfer Object Pattern[1], or in (Fowler 2002). The gained performance has to be traded in for a somewhat compromised OO design though.

Transporting groups of data by value not only helps avoid network congestions and improve overall system performance; migrating such data from one machine to another also decouples the two computers, which adds to the robustness of the system. If the computer which originally supplied the data goes

[1] `http://java.sun.com/blueprints/corej2eepatterns/Patterns/TransferObject.html`

down, the other machine will not be affected if it has retrieved all required data by value before.

Since ALMA software will be implemented in various languages (currently C++, Java, Python), CORBA was chosen as the middleware the ACS framework is built upon. There are several possibilities to transport data by value using CORBA. In the past, for the sake of language independence, CORBA architecture focused on remote service invocations or by-value transport of *simple* data types (primitive data or structs of primitive data); recently it has started to offer a means for transporting more complex data structures by-value, using CORBA value types[2].

For several reasons we decided to use XML as the format for ALMA transfer objects:

- XML avoids difficulties with CORBA's built-in types described in (Schmidt & Vinoski 2001);
- XML as a serialization format can be used not only to send data by value between applications, but also to store that data persistently in a file or a database;
- XML schema allows for data declaration (constraints etc.) more powerful than those available using CORBA value types;
- XML data can be logged directly, and can easily be injected manually into the software system, for example for running unit tests, or to mimic an application that has not yet been built.

ALMA subsystems specify their interfaces in CORBA IDL. To send simple data by value, the built-in data types of CORBA can be used. For more complex, usually hierarchical data, the data definition can be provided outside of the IDL in an XML schema file, and has to be referenced in the IDL. Thus the architecture lets ALMA developers choose between sending data through efficient binary CORBA transport, or using somewhat slower, but more powerful XML plain-text transport. The latter option is expected to be chosen for nested structures such as an observing project and its scheduling blocks, where the size is less than 100 kB.

XML transport is realized in IDL with a CORBA `struct` containing a `string` for the serialized XML, plus complementary administration meta data, e.g. a unique ID.

Very large data structures should be broken up into smaller groups, each described by its own XML schema. For example, the observing project, the proposal, and the scheduling blocks are each modeled separately. A balance must be found between quickly accessing large parts of the data tree in one call, and not transporting too much data at a time when only a part of it is needed. The resulting separate pieces of XML data reference each other using their unique IDs.

---

[2] `http://www.omg.org/cgi-bin/doc?formal/02-06-41`

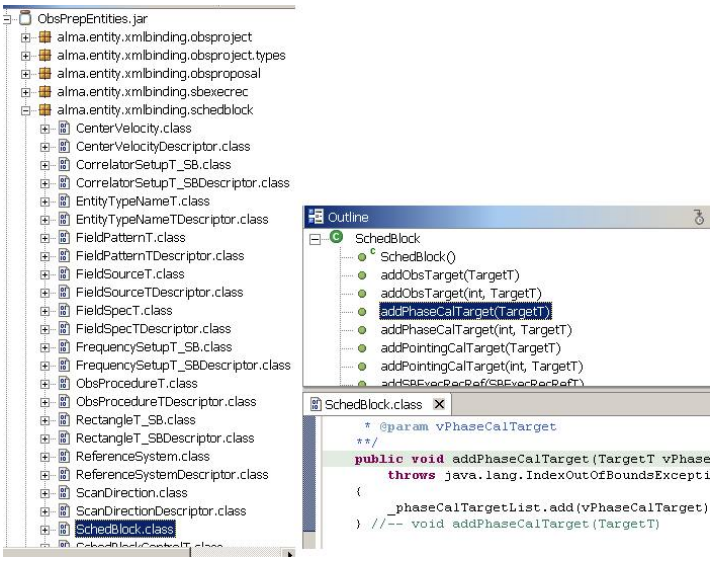

Figure 1. Generated binding classes seen in the Eclipse IDE.

2. XML-Java Binding Classes

XML binding frameworks generate native language binding classes from XML schemas as part of the build process. The binding class instances form in-memory representations of XML documents that belong to the schemas. In Fig. 1 we see the classes compiled from the scheduling block schema.

Applications are written against the type-safe accessor and manipulator methods of these binding classes, thus getting coerced by the compiler to follow any changes in the data model, defined in the schemas.

Note that with the standard representations of XML (such as DOM), application code would contain generic calls like `addChildNode` instead of `addPhaseCalTarget`, thus defying compile time checking.

ACS harnesses Castor[3] for XML binding. Another potential candidate, SUN's JAXB[4], unfortunately lacks the ability to serialize and parse incomplete XML data. For C++ and Python, we have not yet found satisfactory binding frameworks.

3. Transparent Serialization

ALMA software is written as components that run inside ACS containers, as described in (Schwarz et.al. 2003). Each component specifies and implements one CORBA IDL interface; the methods of that interface may use XML data as parameters or return values (see section 1.). However, with a straightforward

[3] `http://www.castor.org/xml-framework.html`

[4] `http://java.sun.com/xml/jaxb/`

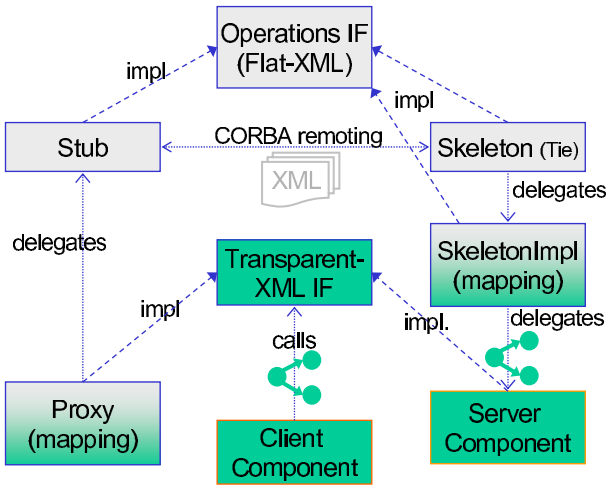

Figure 2. Classes with XML data. To the (green-)shaded classes, XML data appears as binding classes, to others as serialized `strings`.

approach, both the client and the component implementation would encounter XML as a string rather than as a collection of binding class instances.

In Fig. 2, we see the "Transparent-XML" interface, which ACS generates using a custom IDL compiler. It resembles the output of the standard IDL compiler ("Operations-IF"), except that binding classes are substituted for XML-strings. Independently of each other, the component or one of its clients may be programmed against that interface. The container will in this case receive the "flat" XML string from the CORBA ORB, instantiate the appropriate binding classes in its mapping layer, and pass them on to the application.

The tedious task of XML parameter conversion is removed from the application code, and the developers can trust the compiler that at runtime no XML data of unexpected format will be received. The component developer does not even need to know that XML is the underlying interchange data format.

Communication between collocated components can be shortcut by their container, so that binding class instances are passed without going through the serialization-parsing cycle.

The presented ADASS poster is available at `http://www.eso.org/projects/alma/develop/acs/OtherDocs/ACSPapersAndSlides/index.html`. It contains diagrams that will further illustrate the technique.

References

Schwarz, J., Farris, A., & Sommer, H. 2003, The ALMA Software System, this volume, [O8-1]

Fowler, M. 2002, Patterns of Enterprise Application Architecture (Addison-Wesley Pub Co)

Schmidt, D. C. & Vinoski, S. 2001, CORBA and XML, Parts 1 2 3, C/C++ Users Journal