Atacama Large Millimeter Array

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

ADASS XIII

Heiko Sommer (ESO), Gianluca Chiozzi (ESO)
David Fugate (NRAO), Matej Sekoranja (Cosylab)

Overview

ALMA software, from high-level dataflow applications down to instrument control, is built using the ACS framework.

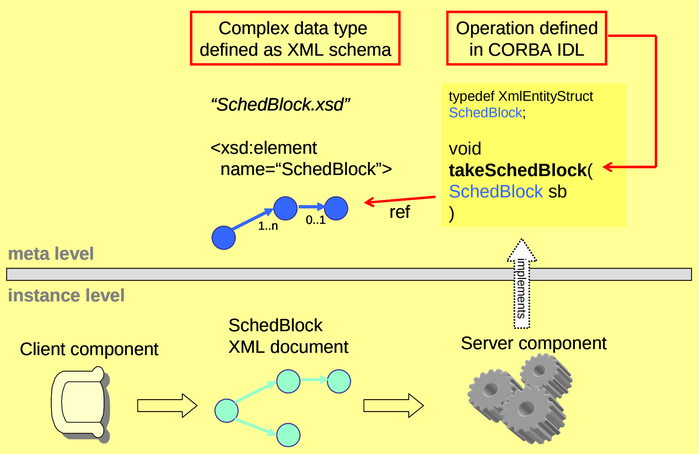
ACS offers a CORBA-based container/component model and supports the exchange and persistence of XML data.

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.

These transfer objects are used to pass by value complex data structures, such as observation meta-data, between heterogeneous applications.

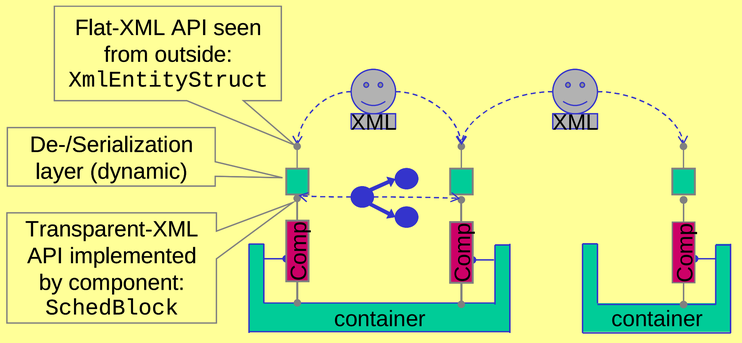
Transparent Serialization

Functional component interfaces are defined in CORBA IDL. XML entities (the transfer objects) are referenced as typedef'd `XmlEntityStructs` (inside: XML as a string + ID, version,...).

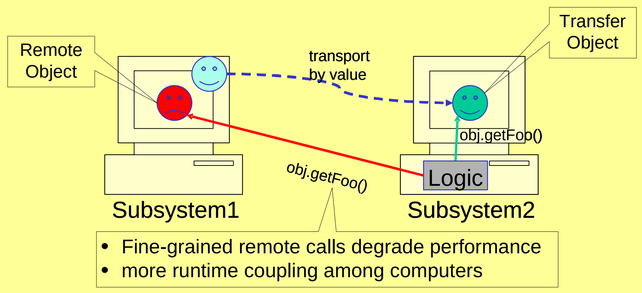


A standard IDL compiler generates Java interfaces that contain XML as a string ("flat"); the custom ACS IDL compiler generates corresponding interfaces that contain the XML binding classes ("transparent" XML).

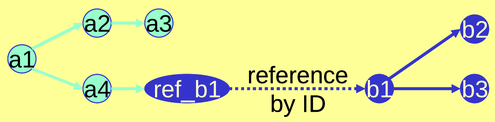
The Java container performs the necessary translations (XML de-/serialization) at runtime. Components only work with binding classes, although standard XML is seen from the outside.



XML Data By Value



- XML good for inter-process communication and persistence
- Simple validation through declarations in XML schema
- Text editor can mimic an application during development



XML Doc 1 schema A imports schema B

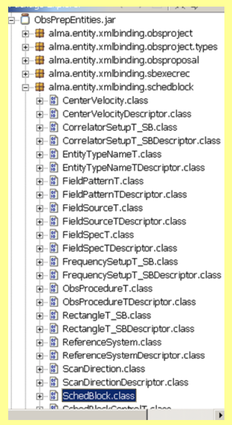
XML Doc 2 schema B

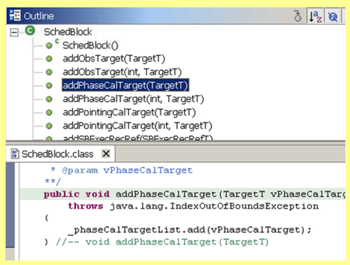
Grouping of data nodes into several schemas. Referencing:
- within a schema: as a child element
- across schemas: by ID (soft reference)
- IDs generated by the archive to ensure system-wide uniqueness; container serves as ID cache

XML Binding

- Java code generated from the XML schemas as part of the software build process
- Type-safe get()/set() methods ensure that version conflicts be caught at compile time
- Binding classes contain code
  - for de-/serialization from and to XML
  - to enforce schema constraints (validation)

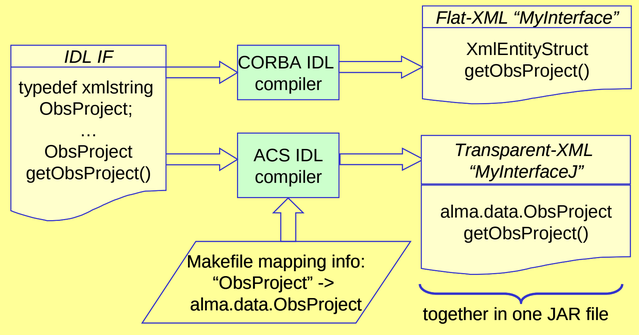
More safety and convenience than with any other XML parsing technique such as DOM, SAX

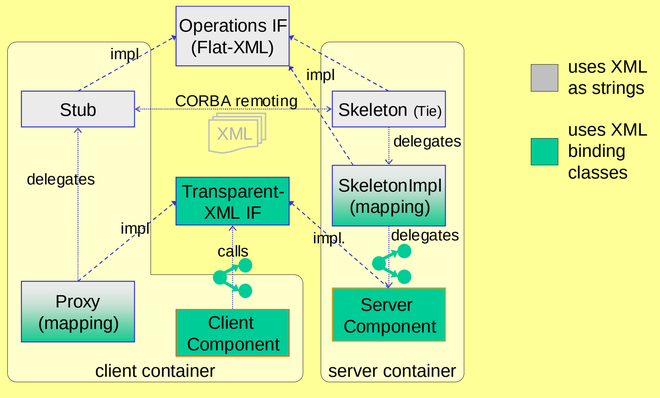




Generated Java packages, classes and methods as seen in the Eclipse IDE

Details on Components and CORBA





The two interfaces ("flat" / "transparent" XML) are connected through mapping layers that the container creates dynamically using Java reflection (`java.lang.reflect.Proxy`).

If both components are collocated, the container can shortcut XML serialization and CORBA communication.

ALMA Usage Examples

- The ObsPrep tool creates deeply nested XML documents for Observation Proposals, Projects, SchedBlocks, using XML binding classes.
- These entities are stored in the archive which has a generic interface and therefore does not work with type-safe binding classes.
- The Scheduling subsystem retrieves SchedBlocks from the archive and ranks them based on their data, again using Java binding classes.
- Correlator configuration is handled as an XML document, produced by the ObsPrep subsystem, and read by the Correlator subsystem.

Details on XML Binding

- To create Java binding classes from XML schemas, we currently use the free Castor framework.
- SUN's JAXB specification is not powerful enough (can only deal with schema-valid XML, thus preventing building up the data spread over components or time).
  XMLBeans look like a promising alternative.
- Binding classes encapsulate data, but lack custom behavior. We either keep functional methods outside (anti-OO) or wrap the relevant binding classes with a "business class" that delegates data access to the underlying binding class.
- Hopefully mature binding frameworks for C++ and Python will become available.