On use of complex data in the Middleware

Kris Kostro, 10 Feb 2000

I will discuss the various aspects of this problem and propose a strategy.
Most of the discussion is related to publishing of complex data but some arguments apply to set/get interface and remote interfaces in general.

Previous contributions:

1. Franck: The "conservative" approach (use Data class of CDEV to pack simple properties together)
2. Vito: The "progressive" approach (use Java objects as properties)

Three players

When transporting complex data the data access and packaging/marshalling problems have to be solved at the client side, at the server side and at the protocol handler. The client and server side do not necessarily have to employ the same method (e.g. Java client and C server, compile time and run-time data description). The protocol handler has to know the data types and packing to marshal the data correctly in and out between different architectures and languages.

Four different approaches to data packing and accessing

In general the best comfort and performance can be achieved when all interfaces are strongly typed and known at compile time This is the approach used in NC RPC, Sun RPC, CORBA and Java RMI. This introduces close coupling between the server and the client and a high dependability between various components of a potentially complex system. Because of this there have often been 'good ideas' to relax this approach such as run-time type description (CORBA Any, SL Equip) or map descriptions (CDEV Data object, MapMessage in JMS). Recently approaches based on XML relax coupling even more by packing identity, type and value of data objects into a structured text. Integrating data description with data itself hampers performance (e.g. use of CORBA Any increases response time by factor of 2 as reported by Boeing). For simple data types (and collections of these) special assessors and transfer functions, one for each simple type, can be used to alleviate the performance problem.
In the following we make a short discussion on utility of each approach for Middleware:

1. **Strongly typed interfaces** provide the best comfort and performance. For complex data this means that objects can be sent "as they are". This is possible in Java, in CORBA only from version 3.0 on. In CORBA (before 3.0) "structures" can be sent instead. Strong typing is also interesting for simple data types, especially on the client side because it allows compile-time type checking and more programming comfort. This can be provided on the client side by encapsulation. Strong typing for all three parts: client, protocol and server should only be used for very stable interfaces of the MW.
2. **Map type interfaces** provide good flexibility at the cost of type safety and complicated interface. One example of the map interface is the one used in JMS for the header or for the map body. Another example is the CDEV Data object. In the map interface elements of the message can be retrieved via the member names and the therefore the type must be described explicitly in the message content (JMS does not define how this is done - it is left to implementation). Member types are normally restricted to simple types (one level of complexity). Conversion rules allow to insert and extract data without knowing the exact type. The main problem with this type of data is the lack of an established standard at all the three levels (client, server, transport).

3. **Data description of arbitrary data type** can be sent with the data itself so that this information can be used for marshalling. CORBA defines Any and DynAny interfaces for this purpose. Simple description methods were used in PS and SL previously which allow the user to describe the structure being sent. The biggest problem with this type of interfaces is that the client and server interfaces are very complex (e.g. CORBA) and therefore error-prone. There is no language support for this type of interfaces. One interesting variant is to use XML to describe the data. The XML schema can be used to describe the data type. Another possibility is to describe the type in the XML message itself (like in SOAP). Since XML is very important for the Internet, one can expect tools and libraries, which allow serialisation of complex types to XML becoming commonplace. The disadvantage of XML is the low performance due to the heavy encoding overhead.
4. **Special interfaces for simple data types** (one for each representative type) do not solve the problem of complex data types but provide the best performance and should be used whenever possible. In fact one could be tempted to use a single method to publish both complex and simple data types. But then simple data has to be encapsulated in something like CORBA Any. This means performance loss by factor of two (as shown in the recent Boeing study) and this can be easily avoided.

Proposed strategy

The strategy which I propose is a mixture of approaches, each one having it's own merits. What is important is that from the user point of view the different approaches should appear as a homogenous or at worst evolutionary interface.

I propose to use the mapped approach for composite data for the time being. CDEV Data approach is the most straightforward but JMS is an equivalent and possible alternative. This will cover the (not so frequent) need of setting, getting and subscribing to complex properties. The implementation of this approach would follow more or less Franck's proposal.

At the same time this approach is not "clean" and does not follow the OO principles (as correctly pointed out by Vito). But it is difficult to envisage sending objects as properties within the current CDEV API framework. The next step would rather follow the bean approach and use the naming and interface repository infrastructure with server device objects and use the normal CORBA/Java programming methods with device classes defined in IDL. The MW project should offer such infrastructure to be used by specific projects (i.e. SPS2001, EA).

Since the simple data types are the ones, which are used most frequently it is worth the effort to define the reply interface such that packing and unpacking of unknown data types can be avoided.