

Jari OS ORB design for RPC and IDL (draft 1.0)

Ing. Alfeiks Kaanoken

Contents

1 Introduction	1
2 Basic concepts	1
2.1 Protocol interfaces nesting	2
2.2 QoS	2
2.3 Distributed message processing	3

List of Figures

1 Abstract flow and basic diagram of the ORB design	1
2 Abstract of interfaces nesting	2
3 Linear mode of distributed message processing	3

List of Tables

1 Introduction

This paper aims to describe the implementation of the ORB for objects in RPC of the Jari OS, as well as to describe the IDL used within the project.

Since Jari OS platform is a distributed system, it requires a more flexible and extensible methods for remote procedure calls and interactions in general.

CORBA standard was taken as the basis, since it the most versalite and suitable for the target platform.

To conform project targets specification was extended, but CORBA/ORB specification might be extended in future with required additional features as QoS. In this case Jari OS implementation doesn't extend it architectically.

2 Basic concepts

From the basic view there are three major parts of design are exists:

- IDL (specialized DSL conforming to CORBA)
- IDL compiler (basically it will support C)
- Environment libraries: one for server, other for client

IDL will describe the interface for the remote procedure calls, format and some additional logic for request processing (QoS for example).

IDL compiler will process an output for client and server parts that will assemble requests, parse it and so on. Primary goal is a C support (since all libraries and servers written on C).

Libraries will support a low level support for CORBA/ORB environments, there libraries aims to be a layer between platform specific and interface implementation.

The basic flow might be shown on the figure 1. To support identification and role recognizing of the

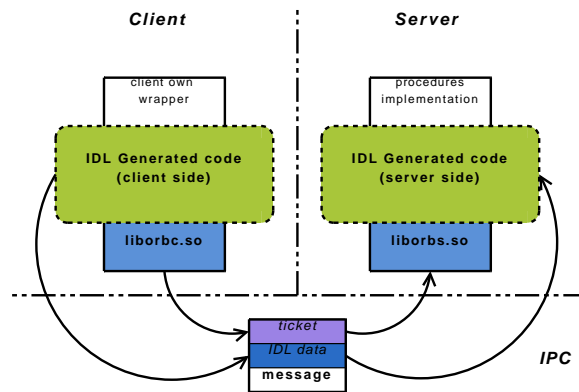


Figure 1: Abstract flow and basic diagram of the ORB design

caller and to be able to support additional features - new abstraction are added: **“ticket”**.

Ticket is the abstraction aims to describe client/caller identification and role recognizing. Actually is the abstraction structure for the low level implementation of role and security policy. By the way, it also used for internals of the ORB implementation.

Communications goes throw messages, and there messages is the abstraction too. Message itself is complete from the two parts:

- ticket
- message data

Message data may be used or maybe not by the IDL generated code, thus depends on the interface.

ORB environment is divided between library supporting code and IDL processed code. Library provide low level functionality and tickets workaround, interface protocol works provided via generated code.

IDL generated code fot the specified interface might be called by the server code if it necessary.

The main objectives of such kind design are:

- protocol interfaces nesting
- flexible mechanism for the transfer of protocol messages
- messages QoS and water marks support
- messages forwarding and distributed processing support

This features are described below.

2.1 Protocol interfaces nesting

Protocol interfaces nesting is required by the varios subsystems implementations such as filesystems, some networking subsystems (MODBUS, DeviceNet and so on). Client request goes to the one server usually, but it can be forwarded with some additions to other and so on. Typical flow can be shown on the figure 2. There are three stages shown on the diagram above:

1. Client sends the message to the server “srv” with its ticket and data
2. Server “srv” adds its own ticket and additional data to process on another server and forward the message
3. Server “srv1” process the message and reply to the client

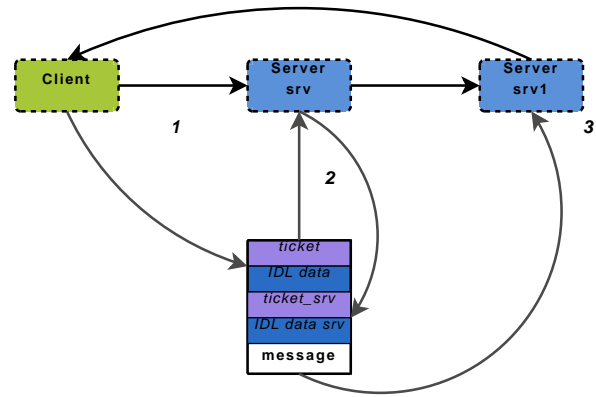


Figure 2: Abstact of interfaces nesting

The other common method of interface nesting is the mechanism for the message transparent transfer. It allows combine distributed and local message processing. CORBA environment implementation allows to call environment from the server code when it required. This allow server developer to implement an abstract design of the system overally. For example, the scheme shown on the figure 2 might be changed – server “srv” and “srv1” may become libraries and works locally within one address space. In this case developer will not required to change the code base, since IDL might works in different cases.

2.2 QoS

The other side is support if the advanced features. Jari OS ORB implementation provide both functionality: **QoS** and **watermarks**. There are an optional features, but it highly required in case of some industry solutions.

For the domestic implementation IDL has support for separation via QoS labels, going throw role identification, watermark, procedure call and other specified features.

CORBA-like environment (actually CORBA/ORB specification doesn’t include QoS support) includes to the ticket QoS label, that determined via specified rules. For example, to determine the QoS level in the figure 2 example Jari OS ORB implementation will take decision in case of two tickets or, if specified, only on one client original ticket. This made to provide support for more complex systems development and it’s required by industry control and quality check systems as well as complex airport automatization systems.

QoS calculation has a predefined options:

- First ticket

- Compound ticket
- Caller ticket

Also, developer may describe it's own calculus of the QoS level within IDL syntax and difinitions. It was made to support custom complex systems and it's semantic. First ticket policy works with the first taken ticket only. Compound ticket works in two modes, firstly ORB environment takes all tickets contains within message and takes more higher leveled, or, instead, more lowest tickets – depends on the mode choosen. Caller ticket policy always take decision on the caller original ticket.

By the way, depends on interface, several policy might be choosen for each interface in one ORB environment.

2.3 Distributed message processing

Distributed message processing is a feature that allows to separate request processing between several servers. That means - ORB implementation can split, forward, organize, substract messages.

There are several cases exist with distributed message processing, starting from the distributed filesystems, finishing with the complex requests in the area if the industry automation. To prevent desintegration of the interfaces Jari OS ORB implementation provide this feature to the domestic areas.

Messages may be processed in two modes:

- Linear mode
- Distributed mode

Linear mode is a simple case of distributed processing. This mode means that several servers works with one message in one by one character. This case shown on the figure 3. This example only for two servers shown, but this processing can be extended to **n** servers (determined in interface IDL description). To make a sense on it, there are three major stages:

1. Client sends the message to the server "srv0" with its ticket and data
2. Server "srv0" write a portion of required data, updates IDL headers and forwards this message to the srv1
3. Server "srv1" process the message, write portion of the data, reply to the client.

The decision of the reply made by the ORB environment and developer should not take care on it in this case. There are some note need to be made: ORB

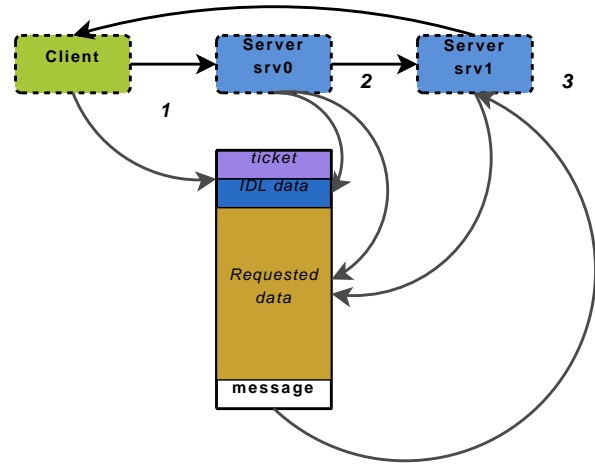


Figure 3: Linear mode of distributed message processing

will calls user defined callback when message processing incomplete and should be forwarded to the next server, user-defined callback should return a server which will get a message next.

This scheme also possible for the centralized systems, where incomplete messages forwarding back to the central server that makes a decision of the next server. This is a typical load-balancing applications with a guaranteed i/o rates (if this will combined with the QoS feature).

Distributed mode is a more complex way to make message processing. This mode allows to divide message to the several ones (in the user-defined manner) and send it to the several servers. When all messages will be delivered ORB environment will complete the message to the original one and reply to the client.

This behavior is under research now, but it may be appear in the next ORB design implementation.