Requirements

Content

[1 Overview 1](#_Toc329706643)

[1.1 Target 1](#_Toc329706644)

[1.2 Audience 1](#_Toc329706645)

[2 Software requirements 1](#_Toc329706646)

[2.1 General functional requirements 1](#_Toc329706647)

[2.2 Documentation requirements 3](#_Toc329706648)

[2.3 Performance requirements 3](#_Toc329706649)

[2.4 Portability and interoperability requirements 4](#_Toc329706650)

[3 Notas 4](#_Toc329706651)

# Overview

## Target

The purpose of this document is to draw up the requirements of the project. This project aims to develop a RPC framework over DDS.

## Audience

This document has been written for the developers of this project.

# Software requirements

## General functional requirements

**[REQGEN1] RPC4DDS shall support the RPC paradigm using standard DDS as communication middleware.**RPC4DDS implements mechanisms necessary to communicate a client and a server using RPC. All data communications will be done using the DDS standard.

**[REQGEN2] RPC4DDS shall use only the standard API offered by DDS standard.**RPC4DDS will not use any privative API.

**[REQGEN3] RPC4DDS shall offer an API from a library to create a server application that exposes services.**User will be able to use RPC4DDS’s library API to create a server application that exposes services that user has defined.

**[REQGEN4] RPC4DDS shall offer an API from a library to create a client application that can access to services exposed in a server.**User will be able to use RPC4DDS’s library API to create a client application that can use defined services exposed in a server.

**[REQGEN5] User shall be able to define services using Interface Definition Language (IDL).**User will be able to define its own services in IDL. RPC4DDS will have an IDL compiler that parses IDL files and generates specific source code. This source code offers a simple API to create a server that exposes the defined services and create a client that can access to defined services in a server. Also it has to generate a client’s stub, a server’s skeleton and the files necessary to compile the generated code.

**[REQGEN6] RPC4DDS shall support CORBA 2.x IDL.**RPC4DDS IDL compiler will be able to read a CORBA 2.x IDL. RPC4DDS doesn’t support all types of CORBA .2x. In case user uses a non-supported type, IDL compiler will show a error.

**[REQGEN7] RPC4DDS shall support next basic types: octet, char, wchar, short, unsigned short, long, unsigned long, long long, unsigned long long, float, double, boolean, enumerations, strings and wide-strings.**RPC4DDS IDL compiler will support these basic types.

**[REQGEN8] RPC4DDS shall support next complex types: arrays, sequences, type definitions, unions and structures.**RPC4DDS IDL compiler will support these complex types.

**[REQGEN9] RPC4DDS shall support the definition of one interface and operations inside it.**User will use one interface and operations IDL definitions to create its own services.

**[REQGEN10] RPC4DDS may support exceptions.**User will define using IDL its own exception that will raises by its services.

**[REQGEN11] RPC4DDS shall offer a C++ API.**RPC4DDS will exposes a C++ API, and IDL compiler will generate specific source code in C++.

**[REQGEN12] RPC4DDS shall offer a Java API.**RPC4DDS will exposes a Java API, and IDL compiler will generate specific source code in Java.

**[REQGEN13] RPC4DDS may offer a .NET API.**RPC4DDS will exposes a .NET API, and IDL compiler will generate specific source code in .NET.

**[REQGEN14] RPC4DDS may support synchronous invocations.**RPC4DDS will offer a synchronous mechanism to user where a remote procedure call in client side is blocking. RPC4DDS doesn’t return the execution to user until server’s reply has arrived or there was a timeout exception.

**[REQGEN15] RPC4DDS shall support asynchronous invocations.**RPC4DDS will offer an asynchronous mechanism to user where a remote procedure call in client is non-blocking. User will set a callback when it makes the remote procedure call, and the execution is returned by RPC4DDS after sending the request. When the reply has arrived, the callback is called by RPC4DDS. In asynchronous invocations also there are timeout exceptions.

**[REQGEN16] RPC4DDS shall support one-way invocations.**User will be able to set as service as one-way using the special IDL word *oneway*. RPC4DDS return the execution to user after sending the request and RPC4DDS doesn’t wait any reply from server.

**REQGEN17] RPC4DDS shall support Shared Memory transport, UDP transport and TCP transport.**User will be able to select what transport it wants to use in the client side and the server side.

**[REQGEN18] RPC4DDS shall support the communication across Internet using the TCP transport.**User will be able to deploy a server using a public IP, and any client will be able to use its services. For this purpose, TCP transport has to be used.

**[REQGEN19] RPC4DDS shall support several strategies when requests are arrived.**

* **One-thread strategy**
* **Thread-by-request strategy**
* **Thread pool strategy.**

User will be able to configure the server strategy that will be use when a request arrives.

**[REQGEN20] RPC4DDS shall be thread-safe.**All API functions and remote procedure calls will be thread-safe.

**[REQGEN21] RPC4DDS shall support RTI DDS as communication middleware.**There will be a version of RPC4DDS that works over RTI DDS middleware.

**[REQGEN22] RPC4DDS shall support OpenDDS as communication middleware.**RPC4DDS implementation should be prepared to use OpenDDS middleware.

## Documentation requirements

**[REQDOC1] RPC4DDS shall contain the following documents:**

* **Installation guide.**
* **Getting started guide. This document contains an easy example to learn quickly the basic concepts of RPC4DDS.**
* **User’s manual. This document explains how to use RPC4DDS and its API.**

**[REQDOC2] RPC4DDS shall contain examples.**RPC4DDS will bring examples that user can follow and understand the behavior and use of the product.

**[REQDOC3] RPC4DDS shall offer its documentation online.**The main format of the RPC4DDS documentation shall be online.

## Performance requirements

**[REQPER1] Round-trip latency of RPC4DDS shall be on the same order as the DDS middleware in use.**The round-trip latency in a remote procedure call shall be on the same order as publishing (any other kind of) data and receiving a subscribed-to response. The latter is measured as the latency between sending a sample and the receipt of a sample in return (a Ping-Pong latency test). This requirement applies to all supported transports.

**[REQPER2] Throughput of RPC4DDS shall be on the same order as the middleware in use.**The throughput in a one-way remote procedure call shall be on the same order as publishing (any other kind of) data.

## Portability and interoperability requirements

**[REQPOR1] RPC4DDS shall be supported in following operation systems:**

* **Windows 7 32-bits**
* **Windows 7 64-bits**
* **Fedora 17 32-bits**
* **Fedora 17 64-bits**

**[REQPOR2] RPC4DDS shall offer installers for each supported operation systems.**In Windows case, there will be a setup executable and in Linux case there will be a package supported by Fedora package’s system.

**[REQPOR3] RPC4DDS IDL compiler shall generate project files to compile the generated source code.**In Windows case, IDL compiler will be able to create project files for Visual Studio 10, and in Linux case IDL compiler will be able to create a Makefile system.

**[REQPOR4] RPC4DDS .NET wrapper shall support .NET 4.0**

**[REQPOR5] RPC4DDS Java wrapper shall support Java 1.6**

# Notas

[REQRPC] RPC library debe funcionar a través de internet. El servidor debe estar accesible a través de una dirección pública.

[REQWEB] Ejemplo cool en un Getting Started que el usuario cree un cliente que pueda comunicarse con nuestro servidor.

NOTA: En el escenario de Internet, DDS solo aporta la serialización/deserialización. Podríamos incluir nuestra propia serialización y tener un producto autónomo.

NOTA: ¿Por qué hemos decidido RPC, en contra de RMI?

* En el ámbito de internet suele usarse RPC, por ser protocolos stateless.
* Por el deadline del 15 de Septiembre.
* Porque los middleware de moda son RPC: WebServices y Thrift.
* Por simplicidad en la implementación del producto.