
# XecliP – The big picture

## 1. Introduction

This document is intended to provide an overview to the way the XecliP server and clients communicate with each other. It will not describe the protocol messages and low level details, since there is a separate document for this, but rather give a high level view.

## 2. Architecture

XecliP is based on classic client-server architecture, or maybe more precise, a hub and spoke architecture. All communication between clients is handled by a central server, which modifies, dispatches or generates the appropriate messages for the clients.

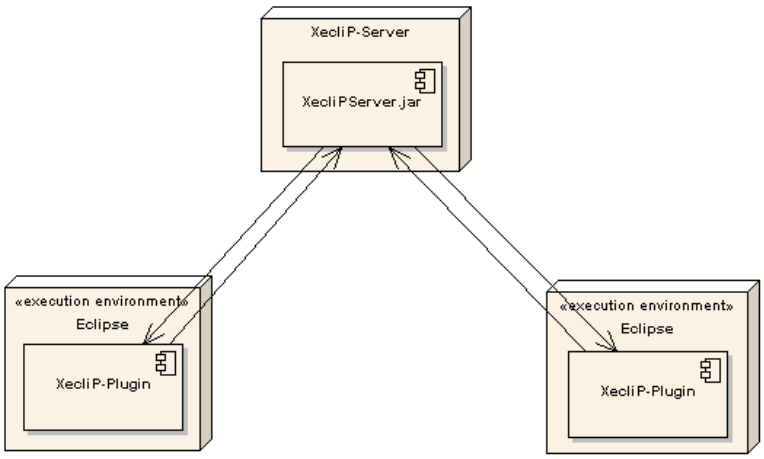


**Figure 1: Communication between clients and server**

The hub and spoke architecture allows the server to easily keep all clients up to date on all relevant events.

The network connection between the clients and the server are persistent TCP connections.

The communication is done partially synchronous, partially asynchronous. Message sending is done synchronous, while messages are received asynchronous, implementing the observer design pattern.

## 3. The XecliP server

The server creates a worker thread for each incoming connection. This way, the server knows (after successful login) which connection belongs to an online user, allowing to route the messages appropriately.

In addition, the state of all online users, sessions and other "shared objects" is available inside the server.

## 4. Communication patterns

The standard communication pattern between a client and the server is a client sending a message to the server, which processes the message and forwards it to the intended receiving client (maybe with a different message type, but matching content). This is a more or less pure client / server communication.

There are several messages for which the server broadcasts events to all users currently online (or all users participating in a given session). This one-to-many communication is basically used to propagate state changes.

Another pattern of communication used often is the classic client/server communication, where a client sends a message to the server, which processes it, and afterwards sends the result back to the client. An example for this type is the login message.

*Figure 2* shows a part of the session setup procedure, which contains all of the patterns described above, as they are used in this scenario.

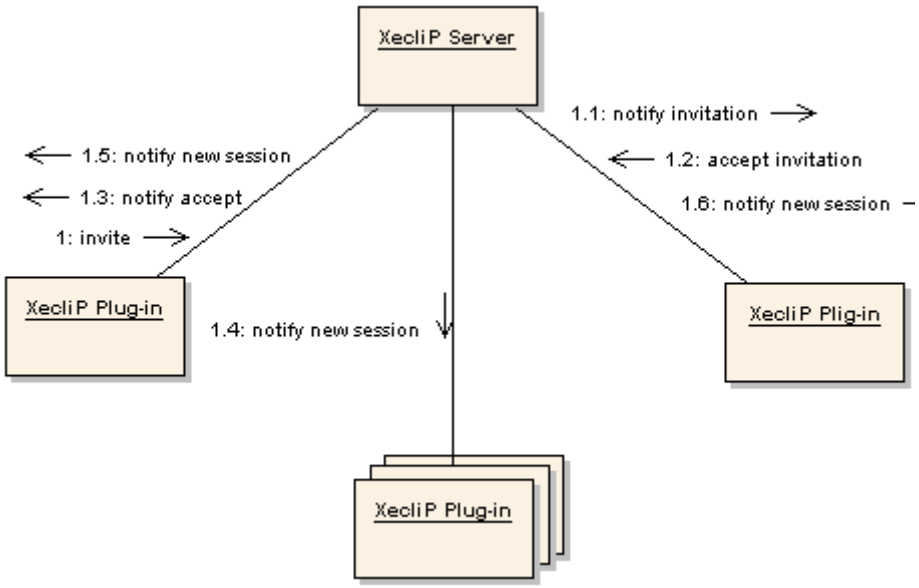


**Figure 2: Session setup messages (inccomplete)**