**Frontend – Backend Separation**

**Software Design Review**

Written by: Slava Chuhovich

May 2012

Table of Contents

[1. Introduction 3](#_Toc325444551)

[1.1 Background 3](#_Toc325444552)

[1.2 Design Goals 3](#_Toc325444553)

[1.3 Abbreviations and Acronyms 3](#_Toc325444554)

[2. Architecture 4](#_Toc325444555)

[2.1 Overview & Data Flow 4](#_Toc325444556)

[2.2 Implementation 5](#_Toc325444557)

[2.3 Pros & Cons 6](#_Toc325444558)

[3. Additional solutions 6](#_Toc325444559)

# Introduction

## Background

The purpose is to reduce mutual dependency between Frontend (user interface) and Backend (low level and hardware related modules) in embedded application and implement proper MVC architecture.

This process could be done by separating Frontend and Backend into two different applications that will communicate with each other through the RPC mechanism, which runs on TCP/IP protocol. Each application could be located on different computers or platforms. The UI interface side (Frontend) could be done with various technologies, like Java, Web … Also partial UI could be implemented if required.

## Design Goals

* Separate Embedded application into two different applications.
* Implement RPC-Server and RPC-Client on each side.
* Implement Frontend related functionality on one side and Backend related on the other.
* Make both applications run simultaneously on the same PC. (a possibility)
* Implement TCP-Server and TCP-Client on both sides for the communication layer.

## Abbreviations and Acronyms

|  |  |
| --- | --- |
| **Abbreviation** | **Description** |
| GUI | Graphic User Interface |
| UI | User Interface |
| FE | Frontend |
| BE | Backend |
| RPC | Remote Procedure Call |

# Architecture

## Overview & Data Flow

The architecture design of both sides is separated into several main modules:

* TCP Client / Server – TCP/IP communication between applications
* RPC Client – Get the required command from the application, serialize it and send to the other side.
* RPC Server – Receive the serialized command and translate it to Roster method call.
* Roster – Database, that holds all available, registered, callable methods in the application.
* Backend:
  + Low level modules - sequencers, motors, door, filling, UV, …
  + Frontend interface – receives FE messages and sends command through RPC.
* Frontend:
  + GUI – main UI where statuses are updated.
  + Backend interface – receives backend commands (actuators, print start / stop, files loading…) and sends them through RPC.



## Implementation

Initial, concept proof implementation is as follows:

* Both applications will be created from the same code-base separated by conditional compiler defines (#ifdef).
* Base classes will be created for both BackendInterface and FrontInterface with corresponding heritage for **Local** and **Remote**.
* Additional 2 virtual COM port (mapped to TCP) objects will created to connect Client to Server and Server to Client on both sides.
* The project will be split into FE and BE executables, for instance: Objet500FE and Objet500BE.
* Application Mutex will be defined to allow both to run on the same PC.
* On both Remote BE and FE implement all available (required) methods to be serialized and sent through RPC and not to be called locally.
* Register all available (required) methods to Objects Roster.
* Add a parameter to set the IP address of the remote machine (both sides).
* Verify correct Client – Server communication and message path.

More complete functionally implementation to be considered:

* Disk and file access from FE side.
* Modes access and activation from FE side.
* Handle methods that return statuses.
* Strip the BE from any UI completely.
* Strip the FE from any low-level or “mid” level functionality, such as MachineManager.
* Handle all HASP related functionality. What should be “enveloped”? Is it possible to tunnel HASP messages through TCP to FE?
* Package all communication, RPC and Roster related modules into separate unit (DLL ?) to be able to use it in any type of UI, for instance Java or Web based.
* Handle reconnection of FE without reopening the BE.
* Consider handling several different UIs simultaneously with the same BE. Multi-client TCP / RPC server.
* Tunnel all log messages to FE’s QMonitor.
* Handle all error and notification user messaging on the FE side.

## Pros & Cons

Pros

* Correct MVC software architecture design and UI – logic separation.
* Ability to design more modern UI with no development environment dependency. (Java, web …)
* Ability to use several UIs at once or use minimalistic UI for just progress and statuses.
* Ability to develop the BE (stripped of all UI parts) on any development environment or platform.

Cons

* Large scale project that deals with many system’s core components.
* This implementation greatly increases the maintenance in every day development. Each method must be implemented for both Local and Remote FE and BE.
* All interaction between sides becomes network throughput and TCP communication dependent.
* When using different ways for UI implementation (Java, web …) must add additional technologies to interact with our Roster and communication modules.

# Additional solutions

There is an option to implement more modern, HTML / CSS based UI apart from BE – FE separation.

This functionality could be achieved by integrating build-in HTML Layout engine into our application. The engine comes in form of a DLL and could be used and compiled in our development environment.

All the UI events will be implemented in our code in C++ and could be triggered from the HTML implemented GUI. At the other hand, all UI controls could be directly accessed from C++ code by their DOM tree representation.

Such API is freely available at [www.terrainformatica.com](http://www.terrainformatica.com) with many examples and documentation.