Software architecture document for project “Robotics”

1. Authors

V. Chernikov

O. Markelov

S. Morozov

I. Shatalov

1. Goals and limitations

1.1. Key functional requirements

* Users must be able to upload their solutions and receive simulation results.
* Teachers must be able to edit levels and create new ones.
* Previous solutions should be stored until they are obsolete.
* Users should have accounts with associated previous solutions and results.

*[Identify use-cases and other functional requirements that significantly affect the architecture]*

1.2. Non-functional requirements

*1.1. Environment*

*● The system should run smoothly on machines with 4 GB RAM and a quad-core x86-64 processor.*

*● The application can be used on any device that has a JVM (Java Virtual Machine) installed and that is connected to the internet.*

*● A student can write scripts for the robot in Groovy programming language.*

*● A teacher can create a new level describing it in Java/Groovy programming language.*

*1.2. Performance*

*● Calculation time in standard situation should be under 2 seconds. Only exceptionally may take up to 10 seconds.*

*● One server should be enough for at least 60 students.*

*1.3. Reliability*

*● The server should work during a week without restart.*

*1.4. Extensibility*

*● The client application not dependent on JVM can be created using the same server API.*

*● The amount of levels calculated at the same time can be increased by adding extra computers to the system.*

*[Identify non-functional requirements that affect the architecture (typically, all of them)]*

1.3. Architectural goals

*[Identify issues that come from 1.1 and 1.2 and will drive the overall architecture.*

*The rest of the section are just examples]*

* There must be multi-user remote access with Web-UI.
* Up to 200 simultaneous users must be supported that is virtually impossible by a single server
* The application must operate on Windows and Linux
* The application must be tolerant to some network messages loss, and handle such losses without lags
* Automatic load testing scripts must be implemented and cover most network-intensive operations.

1.4. Additional goals, restrictions and preferences

* All team members are currently studying Java on the Object-oriented programming course, therefore this language seems to be the best option for server-side part of the project.
* One of the team members is familiar with HTML5 + CSS3 + JS and wants to improve his jQuery and AJAX knowledge, therefore web application seems to be the best option for client-side part of the project.

*[Identify goals and preferences that don’t come from the requirements, but good to be fulfilled. For example, most of the team members know C++ well, so it is better to write on this language if it is possible. Another example: for another ongoing project PostgreSQL DBMS will be used, but the team has little to no experience with that. It is good to gain some during the current project if possible.]*

2. Goals analysis

*[For each goal there must be overview of ways to solve it and analysis of their applicability, pros and cons regarding the system to develop. The overview could mention external libraries, frameworks, algorithms, known architectural solutions (architectural patterns), suggest some modifications of them. For each external element provide reference to corresponded publications/documentation.*

*The rest of this section including subsections are project-specific and provided just as shortened examples.]*

2.1. [ex.] Security

Due to multi-user nature of the system, there must be session support, each session must be authenticated and each operation must be authorized.

There are 3 main user roles in the system:

* Student user, who can do the following: ...
* Teacher, who can manage student accounts and edit levels and inherits abilities from Student.
* Administrator is a teacher, who can add and remove simulator units.

There are two main approaches to support authorization:

* Protection rings [ref], that…
* Access control list [ref], that…

Authentication can be implemented either via login/password pair stored in local DB and appropriate login form, or via external service like Google or Facebook.

It is easier to delegate it to ready-to-use external service, so let’s use Google for authentication. *(In fact, the* *solution is questionable at least. First, there are usually other user-related data besides login/password that must be stored on the application level anyway. Second, integration with an external general-purposes service sometimes could be more complex than writing your own project-specific solution).*

2.2. [ex.]Web-interface

…

2.3. [ex.]OS support

…

…

3. Solution description

*[Provide overall description of the system’s architecture].*

3.1. Modules and subsystems

*[Identify top-level modules/subsystems. Pictures are welcome. For each module provide clear description of its functions/responsibilities and interactions with other modules (contact in other words). Also, for each module identify architectural decisions like key frameworks, libraries, languages, architectural patterns, algorithms. References to the section 4 are OK]*

3.2. Deployment

*[Provide deployment view, i.e. which processes/code parts running on separate devices and how they interact with each other. For distributed software by “devices” separate computers are assumed first of all. Also, this section could show functional distribution between CPU and GPU, interactions with external sensors or other specialized hardware. Pictures are welcome.]*

3.3. …

4. Key architectural elements

*[Identify and define key system-wide (i.e. affecting multiple modules from section 3) behavioral elements like protocols, algorithms, computational mechanisms. Usually such elements are corresponded to at least one goal from the section 1. Each element must be described in terms of modules identified in section 3 and their behavior. Pictures are welcome.*

*This section could be placed before section 3 if necessary.*

*The rest part of this section including subsections are examples]*

4.1. [ex.]Authentication protocol

…

4.2. [ex.]Authorization

…

4.3. [ex.]Fault-tolerant network interaction

…

4.4. [ex.]Modules API

All the modules’ external APIs should follow Command-Query Separation [<reference>] pattern excluding …, where atomic types should be used and, thus, they must violate this pattern.

All the modules excluding … must implement their APIs in two forms: 1) regular Java API in terms of interfaces, and 2) RESTful API using Jetty [<reference>]as embedded servlet container.

Regular Java API must be used in most situations due to better performance and reliability. RESTful APIs should be used mostly for automatic testing purposes.

4.5. [ex.]External plug-in API

For module … the external API for 3rd party video processing plugins must be implemented in reliable way. A plug-in must be run as a separate process restricted by RAM and CPU usage by … So, the main application shall tolerate the situations where plug-in killed due to critical error (segmentation fault or so), overusing memory or CPU, spawning threads.

Interaction with plug-in on lower level must be implemented via STDIN/STDOUT/STDERR. On higher level, the sandbox for plug-ins must be implemented (see 3.NN) and provide the following operations: …

To transmit objects through the raw stream JSON serialization/deserialization must be used. Jackson [<reference>] library is suggested for this. There is the requirement to possess bandwidth of … MB/s for plugins, so Jackson must be checked first to match this.

4.6. …

5. Platform

*[Enumerate all the hardware and external software decisions that come from all the previous section.*

*All the rest is just an example.]*

Server platform: x86-64/amd64, Linux 2.6

Client platform: ARM, Android; Web browser (Edge, Chrome, Firefox)

Language: Java on server and Android client, JavaScript on Web-based client

Server frameworks and libraries: Tomcat as servlet container, Spring (Boot, Security, MVC), PostgreSQL as DBMS, Hibernate for interactions with DB, Jackson for JSON(de-)serialization, JMeter for load testing.

Web client frameworks and libraries: ReactJS