Software Systems Engineering
process report template

Roberto Casadei Massimiliano Martella Roberto Reda

February 12, 2015

1 Introduction

This report represents a snapshot of the current status of the "ButtonLed" project (object-based).

2 Vision

We believe that a good software product is the natural result of a **mature, systematic process**.

We believe that there is no code without design, no design without analysis, no analysis without requirements. So, we recognize the causal progression from the *why* (vision/goals/requirements) to the *what* (analysis) and finally to the *how* (design/code).

We recognize the importance of analysis for a mature process, but we don't want to waste time by just producing informal knowledge. Instead, we would like to **capitalize the effort spent in analysis by producing something that has value, that is, working software (prototype) and reusable frameworks**. That is, we believe that *reuse* is prominent in what we do.

We also think that following a **top-down approach**, i.e., from the problem to the technology, is extremely valuable because it can adequately support our goals. In doing so, we would like to constantly **evaluate the abstraction gap** in order to gain a better vision of what we need to *effectively* build the kind of systems we are considering and also as a way to *innovate*.

Moreover, as we are dealing with (complex) software systems, we believe that a **systemic approach** fits more than an algorithmic approach and gives us access to more powerful conceptual tools.

Last but not least, we think that a software development process built upon the principles outlined above can maximize the value of the effort spent during analysis and development, effectively produce software architectures that are resistant to requirements change, and even turn those changes into an opportunity to increase the organizational know-how.

3 Goals

We use the ButtonLed system as an opportunity to actualize the vision. The ButtonLed system, in its simplicity, captures the essential characteristics of all the software systems: input, elaboration, output. In this case study, the technology hypothesis is given by the object-oriented paradigm.

We are aware that the system we are building is just one of the countless applications that could be built in this domain. So, we generalise/abstract from the issues we encounter in a sustainable manner, so that we can **factor domain-knowledge out of the specific system**.

Also, by analyzing the requirements and the problem, we would like to produce a **working prototype by following an incremental (*piecemeal growth*) approach**.

In particular, we would like to evaluate the cost of passing from a homogeneous, concentrated system to a distributed, etherogeneous system.

Moreover, we would like to correlate what we do with the following concepts:

- Models (⇒ Structure, Interaction, Behavior) as a way to capture the essential characteristics of what we talk about
- Software reuse as the key element to deal with bounded resources
  - Design patterns as synthetic analyses and design ideas for recurring problems
  - Framework vs. pattern vs. middleware
- Technologies as specific ways to actualize designs but also as conceptual spaces (i.e., "representing" a paradigm)
- Logic architecture as the main result from analysis and possibly our main specification of the problem at hand
- Traceability (from requirements to architecture to code) as a way to support "informational consistency" within the project
- Specific vs. schematic vs. generic parts of software

4 Requirements

> " Design and build a ButtonLed software system in which a Led is turned on and off each time a Button is pressed. In particular, the system will have the button on an Android device, the controller on a RaspberryPi, and the led on Arduino. "

5 Requirement analysis

From the requirements, it follows that the system is structurally composed of (at least) three **subsystems** (for which we already know the deployment, see the Deployment section 11):

- One subsystem (on Android) with the button
- One subsystem (on Raspberry) with the controller
- One subsystem (on Arduino) with the led

It also follows that **distribution** and **etherogeneity** are essential characteristics of the system.

As noted above, three main entities arise from the requirements: the button, the controller, and the led.

Note that the controller is explicitly considered in the current version of the requirements. However, in the last *iteration*, the notion of a controller resulted from the problem analysis, as we noted that the responsibility of "notiying the led when the button is pressed" could NOT be assigned to the button, nor to the led.

5.1 Use cases

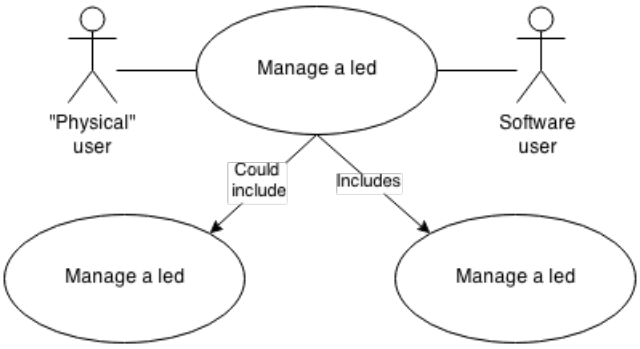

Figure 1: Use case 1

5.2 Scenarios

(Skipped) It's not a primary goal of this case study.

5.3 (Domain)model

From the requirements we know that:

- The button has a state (pressed and not pressed) and, as external entities are interested in that state, it follows that it should be an *observable* entity (see the *Observer* pattern)
- The led has a state as well (on, off) and properties such as the color.

The requirements explicitly give information about the structure of the system and the components, and express a cause-effect relationship between the press of the button and the update of the state of the led. Instead, the dimensions of behavior and interaction are only implictly and partially described by an *observational* point of view.

To explain what we mean by "led" and "button", we provide a **model** for them.

By following an internal convention, we express those models using Java **interfaces**. As interfaces are not sufficient, we also provide **test plans** in order to better define our intended semantics.

By analyzing our idea of "button" and "led", we also note that, in the domain of the *Internet of Things*, the led and the button are particular instance of the concept of **device**. Moreover, we know that a particular device can be realized (implemented) with different technologies (see the *Bridge pattern*)

Listing 1: it.unibo.buttonLed.interfaces.IDevice

```
1  package it.unibo.buttonLed.interfaces;
2  /*
3   * ------------------------------------------------------------
4   * This is a first model of a (IOT) Device:
5   * an entity with a name and
6   * a default representation (expressed in Prolog syntax)
7   * ------------------------------------------------------------
8   */
9  public interface IDevice {
10         public String getName();            //property
11         public String getDefaultRep();  //mapping
12 }
```

Listing 2: it.unibo.buttonLed.interfaces.IDeviceInput

```
package it.unibo.buttonLed.interfaces;
import it.unibo.is.interfaces.IObservable;

/*
 * ------------------------------------------------------------
 * This is a first model of a (IOT) Input Device:
 * an observable entity with a name and
 * a default representation (expressed in Prolog syntax)
 * ------------------------------------------------------------
 */
public interface IDeviceInput extends IDevice, IObservable{
}
```

Listing 3: it.unibo.buttonLed.interfaces.ILed

```
package it.unibo.buttonLed.interfaces;
//import it.unibo.buttonLedSystem.BLSSysKb.LedColor;
/*
* ------------------------------------------------------------
* This is a first model of the Led:
* a Led is a Device with a color and a internal binary state
* ------------------------------------------------------------
*/
public interface ILed  extends IDevice{
        public void turnOn();                   // modifier
        public void turnOff();                  // modifier
        public void doSwitch();                 //non-primitive

        public IColor getLedColor();    //property
        public boolean isOn();                  //property
}
```

Listing 4: it.unibo.buttonLed.interfaces.IButton

```
package it.unibo.buttonLed.interfaces;

public interface IButton extends IDeviceInput{
        public boolean isPressed() ;    //property

        public void high();             //modifier
        public void low();              //modifier
}
```

5.3.1 Behavior

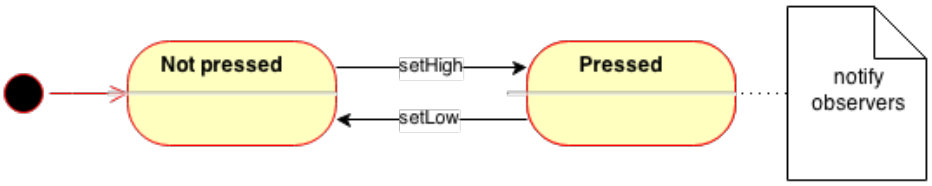

Figure 2: Button: behavior

Figure 3: Led: behavior

5.4 Test plan

See *it.unibo.group08.buttonled.test* project.

6 Problem analysis

6.1 Logic architecture

The system is composed of three main components: the led, the button, and the controller.

From the requirements, we also know that they are on different subsystems (see Requirements Analysis in Section 5).

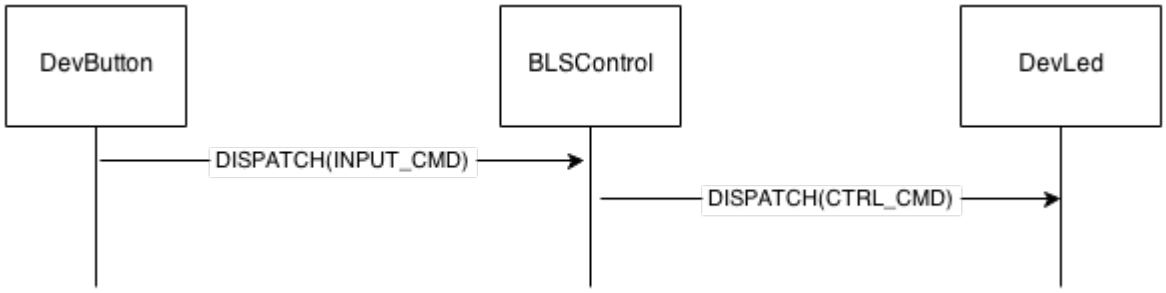

Figure 4: Logic Architecture – Structure/Interaction view

Please consult the software house's internal reference to get a description of the semantics of the "DISPATCH" message.

For simplicity, at the moment we can think at the communication language as described by the following grammar:

```
1 INPUT_CMD = 0 | 1
2 CTRL_CMD = 0 | 1
```

The controller's behavior can be represented by the following FSM:

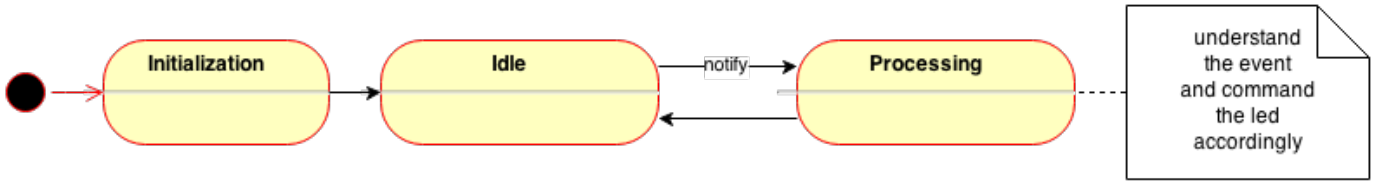

Figure 5: Controller: behavior

6.2 Abstraction gap

Our technology hypothesis is given by the object-oriented paradigm (and, in practice, by Java).

The problem and the practical issues encountered through the analysis phase point out that a gap exists with respect to the following themes/areas:

- **Model specification**: we have specified the models of the system and the components using Java interfaces, test plans, and (custom variants of) UML. However, our need of producing models that are both formal and expressed at the right level of abstraction is not adequately satisfied.
- **"General" system concepts**: OOPLs do not (directly) support our system view (which we use instead of an algorithmic view) through high-level concept such as "environment", "situated entity",

"named entity", ... ⇒ our effort in filling this gap has resulted in the *it.unibo.system* package (within *it.unibo.noawtsupports*)

- **Rapid prototyping**: our goal to have analysis end up with a (graphical) prototype is not effectively supported by Java GUI libraries (AWT, Swing) ⇒ our effort in filling this gap has resulted in *it.unibo.envBaseAwt*
- **Communication infrastructure**: in this technology hypothesis, the communications between the system's components are not adequately supported ⇒ the need to delegate the technology-specific details of communications to the infrastructure brought our software house to develop *it.unibo.noawtsupports*
- **Interaction semantics**: the high-level semantics of the interaction between the system's components is not directly captured by neither the tools we use to express models nor by our platform. We conventionally introduced some names (dispatch, request, signal, ...) to specify a vocabulary for (informally) talking about interaction at the level of abstraction we consider more appropriate. However, this semantics is not directly supported.

In particular, we note that the issue of "distribution" generates forces that urge us to consider a message-passing style for interaction.

6.3 Risk analysis

If we don't adequately fill the abstraction gap, and if we don't abstract from technology-specific and application-specific details, we risk to have significant losses in case of change of (both functional and non-functional) requirements.

7 Work plan

The project team consists of three engineers. So, ideally, the work is subdivided on a subsystem-basis:

- One developer will design/develop the Android subsystem
- One developer will design/develop the RaspberryPi subsystem
- One developer will design/develop the Arduino subsystem

Here, it is important to clearly define the subsystems' interfaces (communication media, communication language, ...). An architectural design will be defined (see Section 8) to better specify the constraints and interactions.

Moreover, certain code/naming conventions should be defined in order to preserve the internal quality (coherence) of the system.

8 Project

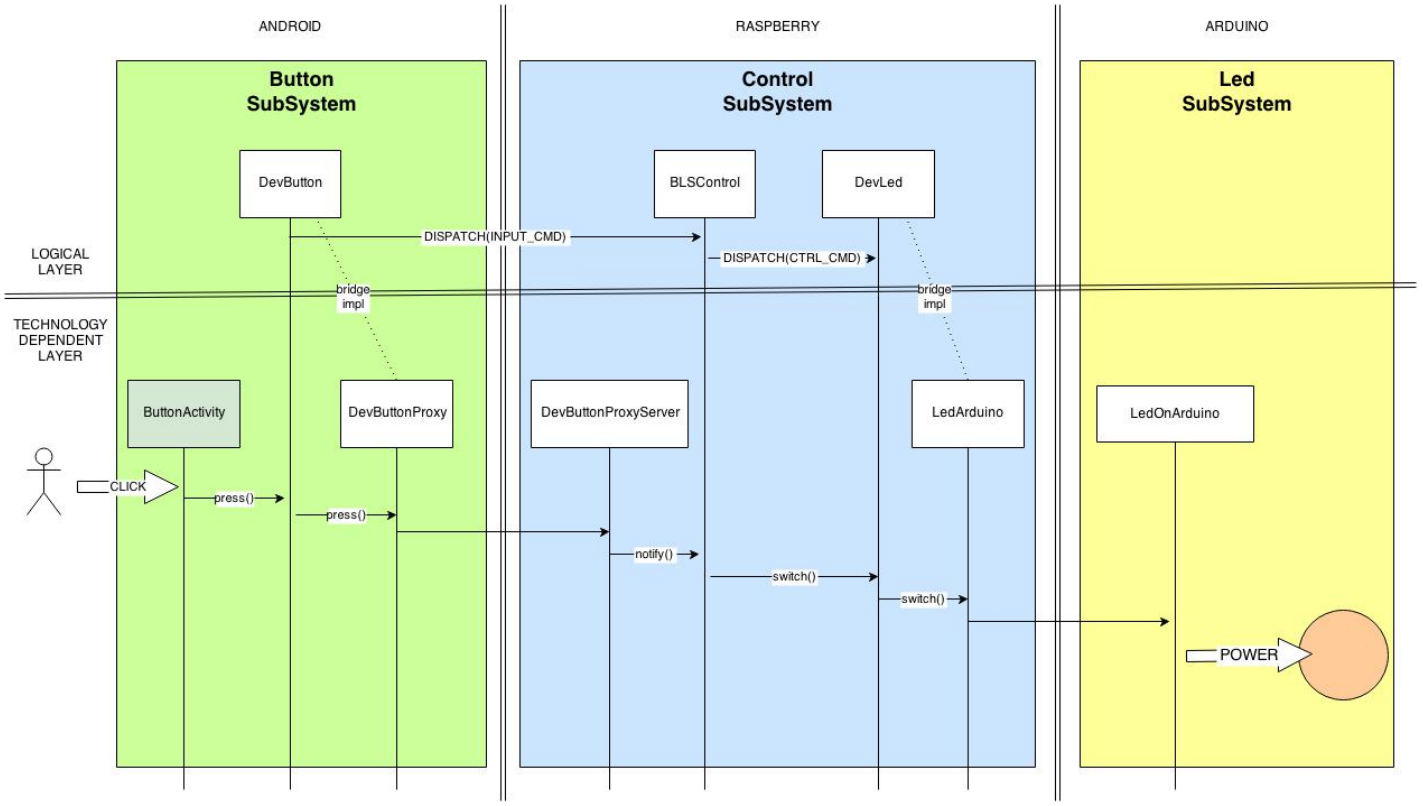

Figure 6: Design architecture

Notes:

- The pattern *Proxy* is used to hide the details of network-based communication when requesting operations to the button/led/controller

9 Implementation

See *it.unibo.group08.buttonled.logical* and *it.unibo.group08.buttonled.system* projects.

10 Testing

See *it.unibo.group08.buttonled.test* project.

11 Deployment

Directly from the requirements, the following deployment configuration follows:

- An APK package for the Android subsystem
- A JAR package for the RaspberryPi subsystem
- A INO file for the Arduino subsystem

12 Maintenance

13 Information about the authors