Software Systems Engineering
Case Study 2016

Paolo Sarti and Marcello Ballanti

Alma Mater Studiorum – University of Bologna
viale Risorgimento 2, 40136 Bologna, Italy
`paolo.sarti2studio.unibo.it, marcello.ballantistudio.unibo.it`

1 Introduction

The following report describes the software development process for an IoT application. The whole process is divided in two steps: at first, the client will communicate some requirements for the application, then new features will be requested. The report will show the impact of client requirements changes on the project on both the design and implementation phase.

2 Vision

We want to discuss the process of software development in order to overcome the limits of a technology-based approach in heterogeneous distributed system application design. We try to adopt a model-driven software development taking into account the AGILE and SCRUM methods for cooperation and work management. In particular, we want to:

- define a formal, executable model of the application to receive feedback from the client and ensure that requirements are clearly defined as soon as possible
- minimize the abstraction gap between the development tools and the application domain entities
- delay any technological hypotesis as much as possible in order to improve application reusability
- create flexible applications to resist requirements changes and add new features easily

3 Goals

The goal is to solve the given problem following the principles described in the vision and determine if this approach is viable and convenient. We want to build a prototype quickly and test if we can add new features to the application with minimum effort.
We'll use domain specific languages to

4   Requirements

We have to solve the following problem:

The Security Department of an Airport intends to exploit a differential drive robot equipped with a sonar (and some other device) to inspect -in a safe way- unattended bags when they are found in some sensible area of the Airport.

The software working on the inspector-roobot should support the following behavior:

- an operator drives the robot from an initial point (robot base area, RBA) towards the bag. To drive the robot the operator makes use of a remote robot control interface running on a smart device or a PC. The robot must accept commands from a single source only;
- as soon as the robot sonar perceives the bag within a prefixed distance (e.g. d=20cm):
  1. the robot automatically stops
  2. the robot starts blinking a led
  3. the robot starts a first detection phase (e.g. it moves around and performs some action according to its equiment - for example it could take some photo of the bag)
  4. the robot sends the results of its detection phase to the Airport Security Center;
- at the end of its work, the robot turns the led off and automatically returns to its RBA. During this phase the Airport Security Center could emit an 'alarm' signal; in this case the robot must restart to blink.

STEP 1
Design and build a working prototype of this inspector-robot.
Non functional requriments at step1
The goal is to build a software system able to evolve from an initial proptotype (defined as the result of a problem analysis phase) to a final, testable product, by 'mixing' in a proper (pragmatically useful) way agile (SCRUM) software development with modelling.
STEP 2 (Implementation Optional)
Extend the last requirement as follows:

- If the bag is qualifed as "harmful", the Airport Security Center emits an 'alarm' signal and activates another (properly equipped) robot that (starting from the same RBA of the robot inspector) should reach the bag in autonomous way and remove the bag from the area.

5   Requirement analysis

5.1   Use cases

The use cases describe how actors (UML actors i.e. the role played by a user or external system) interact with the system. In the requirements we can identify two external entities:

- **The operator** that drives the robot remotely from the initial point to the bag.
- **The Airport Security Center** that receives the results of the robot's detection phase and then it may emit an 'alarm' signal.

These interactions are shown by the UML below:
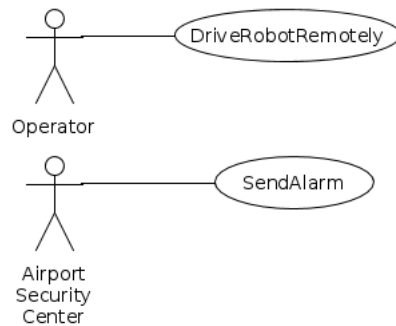
5.2 Scenarios

Title	DriveRobotRemotely
Description	The operator drives the robot to the suspicious bag
Relationships	
Actors	Operator
Preconditions	The robot is in the RBA, waiting for commands from the operator.
Postconditions	The robot starts the detection phase.
Main scenario	The operator uses the remote console to drive the robot.
	When the robot perceives the bag, it starts the detection phase.

Title	SendAlarm
Description	The Airport Security Center sends an alarm signal to the robot if needed
Relationships	
Actors	Airport Security Center
Preconditions	The Airport Security Center received the detection results
Postconditions	The robot blinks its led until it comes back to the RBA.
Main scenario	The Airport Security Center uses its interface to send the alarm to the robot.
	The robot blinks its led.

5.3 (Domain) model

In this phase we try to find an agreement with the client on what the entities mentioned in the requirements are and what they have to do.
The system is composed by three parts:

- **Operator's remote console**
- **Airport Security Center's remote console**

A **console** is a physical or virtual device that allows communication between the system and an external entity. It can get user input data and send them to the system, show some system output data to the user or both. In this case, the operator's console can get input from the operator and the Airport Security Center's console can receive the detection results and emit an alarm signal.

A **differential drive robot** is a composed entity that is able to use some devices to perform actions and receive data from the environment. It can also communicate with other parts of the system. All differential drive robots must have a sonar and are able to move in the environment. In this case, the differential drive robot has DC motors and wheels to move, a sonar and a led. DC motors, wheels, led and sonar are the hardware components mounted on the robot.

A **DC motor** can spin the attached wheel clockwise or counter-clockwise.

A **led** can be turned on or off.

A **sonar** can send an ultrasonic signal (trigger) and generates a corresponding response waveform (echo). The waveform analysis allows to estimate the distance from an obstacle.

The system can be formally defined with a custom language / executable meta-model developed by our software house. It allows us to describe what are the parts of the system, how they interact with each other and their behaviour.

The following is a first description of the system obtained by the requirement analysis:

```
1   RobotSystem testCase2016Analysis
2
3   Dispatch drive : drive(X)
4   Dispatch detectionResults : detectionResults(X)
5   Event alarm : alarm
6   Event obstacle : obstacle(X)
7
8   Context ctxDriveRobot ip[host="localhost" port=8010]
9   Context ctxOperator ip[host="localhost" port=8015]
10  Context ctxASC ip[host="localhost" port=8020]
11
12  QActor operator context ctxOperator
13  {
14    Plan init normal
15      println("Operator starts");
16      switchToPlan sendCommands
17
18    Plan sendCommands
19      println("Waiting for commands");
20      delay time(3000);
21      println("Sending command");
```

```
22        forward driverobot -m drive : drive("driveCmdPayLoad"
             );
23        println("Command sent");
24        delay time(2000);
25        repeatPlan 0
26 }
27
28 QActor asc context ctxASC
29 {
30    Plan init normal
31       println("ASC starts");
32       switchToPlan waitForResults
33
34    Plan waitForResults
35       receiveMsg time(600000);
36       onMsg detectionResults : detectionResults(X) ->
37       println(detectionResults(X));
38       onMsg detectionResults : detectionResults(X) ->
39       switchToPlan riskDecision
40
41    Plan riskDecision
42       println("Evaluating risks");
43       delay time(3000);
44       //It could emit the alarm signal
45       emit alarm : alarm
46 }
47
48 Robot mock QActor driverobot context ctxDriveRobot
49 {
50    Plan init normal
51       println("driverobot starts");
52       switchToPlan drive
53
54    Plan drive
55       receiveMsg time(600000) react event obstacle ->
             detect;
56       onMsg drive : drive(X) -> println(savingmove(X));
57       onMsg drive : drive(X) -> println(driving(X));
58       repeatPlan 0
59
60    Plan detect
61       println("Stopping...");
62       delay time(1000);
63       println("Start blinking the led");
64       println("Starting detection Phase...");
```

```
65        delay time(3000);
66        println("Sending results");
67        forward asc -m detectionResults : detectionResults("
               results");
68        println("Detection Results Sent");
69        println("Stop blinking the led");
70        println("Back to base");
71        switchToPlan backToBase
72
73    Plan backToBase
74        delay time(20000) react event alarm -> alarmReaction;
75        switchToPlan finish
76
77    Plan alarmReaction resumeLastPlan
78        println("Alarm!");
79        println("Start blinking the led")
80
81    Plan finish
82        println("DriveRobot ends")
83 }
```

The operator can only send commands to drive the robot. The Airport Security Center waits for the detection results and can emit the alarm only after the results have been sent.

5.4   Test plan

We can do a test plan even before starting to implement the application, we just need to check if the parts of the system behave and interact with each other as defined in the requirements. We describe these tests in natural language In the initial phase, the operator drives the robot. We have to check the following:

- the operator can send commands to the robot
- the robot executes the commands it receives
- the robot perceives the presence of an obstacle

In the detection phase, the robot inspects the bag. We'll test the following:

- the robot stops and ignores commands from the operator
- the robot starts blinking after it stopped
- the robot can send the results to the Airport Security Center
- the Airport Security Center can receive the results of the inspection
- the robot stops blinking at the end of this phase

In the final phase, the robot comes back to the RBA. These are the tests we'll do:

- the robot actually comes back autonomously

- the Airport Security Center can emit the alarm signal
- the robot blinks the led if it perceives the alarm

At this stage in the development process, we can't define more specific functional or integration test. However, we could test the parts of the system defined with our custom language creating mock entities. We still haven't decided what technology we will use to implement the application, so we can't write executable tests yet.

6 Problem analysis

6.1 Logic architecture

1. Structure:
   - **DriveRobot** differential drive (two motor-wheels) robot equipped with a sonar.
   - **Operator Interface** used by the operator to drive the robot to the suspicious bag.
   - **ASCConsole** allows the Airport Security Center to receive the results of the detection phase and to emit alarm signal
2. Interaction:
3. Behaviour:

6.2 Abstraction gap

6.3 Risk analysis

7 Work plan

8 Project

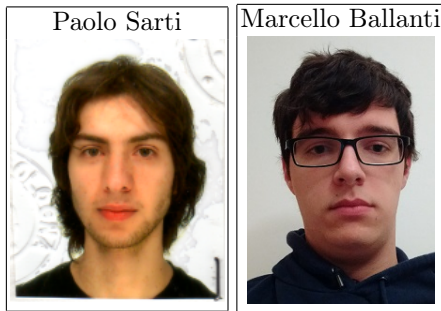## 8.1 Structure

8.2 Interaction

8.3 Behavior

9 Implementation

10 Testing

11 Deployment

12 Maintenance

See [1] until page 11 (`CMM`) and pages 96-105.

13   Information about the author

Paolo Sarti

Marcello Ballanti

References

1. A. Natali and A. Molesini. *Costruire sistemi software: dai modelli al codice.* Esculapio, 2009.