Proposal for an application

Lisa Cattalani and Andrea Pari
Alma Mater Studiorum - University of Bologna, via Venezia 52
47023 Cesena, Italy
lisa.cattalani@studio.unibo.it, andrea.pari6@studio.unibo.it

June 5, 2017

Contents

1 Introduction

The current report describes the entire software development process adopted to analyze, design and implement a specific software system. The process has been divided into two phases: the first phase concerns the definition of a model of the software system, while the second phase concerns the implementation of the software system carried out after that the product owner has viewed the model.

2 Vision

We want to show how to manage the development of a distributed, heterogeneous and embedded software system by focusing on the analysis and design of the system. The software system at issue will be designed and implemented as an application in the field of *Internet of Things*.

3 Goals

The goal is to build a software system able to evolve from an initial prototype (defined as result of a problem analysis phase) to a final and testable product. This goal will be achieved by working in a team and by "mixing" in a proper (pragmatically useful) way an agile (SCRUM) software development with modeling.

4 Requirements

4.1 The context

A differential drive robot (called from now on robot) must reach an area (B) starting from a given point A. To reach the area B, the robot must cross an area equipped with N ($N \geq 1$) distance sensors (sonars). The signal emitted by each sonar is reflected by a wall put in front of it at a distance of approximately 90 cm. Moreover:

- The section of the wall in front of each sonar is painted with a different illustration.
- The robot is equipped with a distance sensor (sonar) and (optionally) with a Web Cam both positioned in its front. It owns also a Led.
- The robot should move from A to B by travelling along a straight line, at a distance of approximately 40-50 cm from the base-line of the sonars.

4.2 The work to do

Design and build a (prototype of a) software system such that:

- Shows the sonar data on the GUI associated to a console running on a conventional PC.
- Evaluates the expression

  $(s_k + s_{k+1} + ... s_N) / (N - k + 1)$

  where k is the number of the first sensor not reached by the robot and s_k is the value of the distance currently measured by that sensor. If the value of the expression is less than a prefixed value DMIN (e.g. DMIN = 70), play an alarm sound.
- When the robot reaches the area in front of a sonar, it:
  - first stops;
  - then rotates to its left of approximately 90 degrees;
  - starts blinking a led put on the robot;

  - takes a photo of the wall (in a simulated way only, if no WebCam is available) and sends the photo to console by using the MQTT protocol;
  - rotates to its right of approximately 90 degrees to compensate the previous rotation;
  - stops the blinking of the led and continues its movement towards the area B;
- When the robot leaves the area in front to the last sonar, it continues until it arrives at the area B.
- Stops the robot movement as soon as possible:
  - when an obstacle is detected by the sonar in front of the robot;
  - when an alarm sound is played;
  - the user sends to the robot a proper command (e.g. STOP).
- Makes it possible to restart the system (by manually repositioning the robot at point A) without restarting the software.

5 Requirements analysis

The first step to analyze correctly the set of requirements is to read the requirements text in order to understand the applicative domain.

After a careful reading, we have considered appropriate to do a short interview with the product owner in order to clear up any doubt come to light during the requirements text reading. The interview is reported below.

Q: How the robot starts its movement when placed on the given point A?
A: The user must send a START command from the console.

Q: Which must be the robot behaviour when it reaches the area B?
A: It must stops his movement, perhaps because an obstacle is in front of it.

Q: Which sonars are responsible to detect the obstacles along the path of the robot?
A: The sonars along the path of the robot can detect any type of object in front of them (the robot in particular). The sonar on the robot can detect obstacles in front of it, too.

Then, we have proceeded to create a glossary of terms in order to avoid any ambiguity or misunderstanding during the next analysis phases. The terms have been reported in **[Tab. 1]**. For each term, the related definition and eventual synonims are shown.

Term	Definition	Synonyms
Prototype	A prototype is an incomplete version of a software system used to demonstrate concepts, to try some project options and to find out problems and possible solutions. A prototype can present all or a few characteristics of the final system.	Model
Differential drive robot	A differential drive robot is a two-wheeled drive robot system with independent actuators for each wheel. The name refers to the fact that the motion vector of the robot is the sum of the independent wheel motions.	Robot
Distance sensor	A distance sensor is a device able to detect the presence of an object (i.e. an obstacle) in front of itself.	Sonar Sensor
Console	In the current domain, the console represents the means by which the user and the robot interact with each other. Through the console, the user sends commands to the robot and receives the data of the photos taken by it.	Command panel
GUI	This term refers to the *Graphical User Interface* associated to the console. Through this interface, the user can view the data sent to the console by the sonar of the robot.	Interface User interface
Alarm sound	The sound emitted by the console when the value calculated through the expression of the distance to the next sonar is under the prefixed threshold.	Alarm
MQTT protocol	*MQTT (MQ Telemetry Transport)* is a communication protocol based on the publish/subscribe pattern. MQTT represents the means by which the robot sends to the console the data of the photos.	
Obstacle	Any type of object located along the path of the robot that can potentially prevent it to reach the area B.	
User	The user is the one who will use the software system, giving commands to the robot through the console and receiving the data of the photos taken by it.	End user
Command	A command is an order given by the user to the robot using the console on the PC.	
Restart the system	The process through which the robot is manually repositioned on the given starting point A without the need to restart the software.	Restart Reboot Reboot the system

[Tab. 1 - Glossary]

5.1 User Stories

A **user story** is an informal description of one or more features required by a software system in order to satisfy a need of the stakeholders. They are expressed in natural language, in order to be understandable by anyone.

A user story is then described from the end user's point of view, and not from the system's point of view. This means that the description is made without expressing any technical details about the analysis, the design and the implementation of the software system.

Each user story will be defined using the following format:

*"As a **user**, I want <GOAL> [so that <BENEFIT>]"*

The stories collected are shown below.

1. As a **user**, **I want** to use a console running on a PC **so that** I can interact with the robot and send it specific commands to start and stop its movements.
2. As a **user**, **I want** that both the sonar on the robot and the sonars along the path can detect objects so that collisions with them can be avoided.
3. As a **user**, **I want** that an alarm sound was played when the value calculated by the expression of the distance is under a prefixed threshold so that I can be alerted.
4. As a **user**, **I want** that the robot stops its movement whenever it reaches the area in front of each sonar along the path. Moreover, I want that the robot rotates to its left **so that** it can take a photo of the illustration painted on the wall and send it to the console.
5. As a **user**, **I want** a Graphical User Interface associated with the console so that I can see the data sent by the robot, including the photos.
6. As a **user**, **I want** that the robot stops its movements when it reaches the area B.
7. As a **user**, **I want** to restart the system by manually repositioning the robot at the given point A (i.e. without restarting the software).

5.2 Sum up of functional requirements

Now that the applicative domain has become more clear and definite, we proceed to list in broad terms the requirements that the final system will must possess in order to satisfy the needs of the product owner. The three groups identify the main macro-components of the system.

1. **Robot**
   (a) Movements
      i. Move forward
      ii. Rotate on its left
      iii. Stop
   (b) Detect obstacles
   (c) Blink a led
   (d) Photos
      i. Take photos
      ii. Send photos to the console
2. **Console and GUI**
   (a) Send commands to the robot
   (b) Evaluate distance expression
   (c) Play an alarm sound
   (d) Show sonar data
3. **Sonars along the path**
   (a) Detect obstacles
   (b) Send distance data to the console

5.3 User interaction and scenarios

In order to illustrate in a comprehensive and unambiguous manner the functional interaction between the (external) users and the system, we will use the formalism given by the UML meta-model. In **[Fig. 1]** an use case diagram is shown in order to describe the interaction between an user and the system through a set of "actions", which represent the use case of the system.

Note that only what an user can directly do with the system is shown.

[QUI IL DIAGRAMMA]

[Fig. 1 - Use case diagram]

Now we will describe the scenarios according to the described use cases.

ID / Name	**#1 - Robot (re-)positioning**
Description	Positioning the robot on the given point A.
Actor	User
Pre-conditions	The user must have the robot available.
Main scenario	The user places the robot on the given point A in such a way that: 1. the robot can move straight towards the area B; 2. the distance of the robot from the base-line sonars along the path was of approximately 40-50 cm. Then, the robot can move.
Alternative scenarios	—
Post-conditions	The robot must reboot itself every time it is placed on the point A.

[Tab. 2 - Scenario 1]

ID / Name	**#2 - Sending commands**
Description	Sending commands to the robot through a console running on a conventional PC.
Actor	User
Pre-conditions	1. The console is running on the PC. 2. The system is ready to accept commands from the console.
Main scenario	The user interacts with the robot in the following way: 1. the user sends a command to the robot through the console on the PC; 2. the system receives the command, evaluates it and executes it.
Alternative scenarios	—
Post-conditions	The robot must: 1. move forward if the system receives a START command; 2. stop itself immediately if the system receives a STOP command.

[Tab. 3 - Scenario 2]

ID / Name	**#3 - Displaying data**
Description	Viewing the data on the GUI associated to the console.
Actor	User
Pre-conditions	The console is running on the PC.
Main scenario	The user sees and checks on the GUI the data sent by the robot, including the photos taken by it.
Alternative scenarios	—
Post-conditions	—

[Tab. 4 - Scenario 3]

5.4 (Domain) Model

We will now describe the (domain) model of the system according to the requirements expressed by the product owner. So, for each entity of the system, the model will describe them on the three dimensions of structure, interaction and behaviour.

Robot

Structure. The robot is a non-atomic reactive entity. It is composed by the following parts.

1. *Wheels:* circular tire that is intended to rotate on an axle bearing.
2. *Eletric motors:* electrical machines able to move and rotate the wheels. There is a motor for each wheel.
3. *Sensor:* electronic component able to detect events or changes in its environment and send the information to other electronics. In particular, the sensor must be able to detect objects.
4. *H-bridge:* electronic circuit that allows the motors to run forward.
5. *Led:* passive output device that emits a blinking light when activated.
6. *Raspberry Pi:* computational hardware unit able to control and handle motors, sensors, led and generally any I/O operation with other computational entities.
7. *Frame/Platform:* plan in which all the components described above are placed and installed.
8. *Powerbank:* portable device that can supply power from its built-in batteries.
9. *WiFi USB:* device that allows Raspberry to connect to the network and communicate remotely with other computational entities.
10. *Various hardware accessories:* wires and other minor hardware components.

Interaction. The robot interacts only with the console, sending it the photos taken by it.

Behaviour. The general behaviour of the robot can be summed up as follows:

1. to move along a linear path starting from a given point A and moving towards an area B (standard behaviour);
2. to react to particular events that can change its standard behaviour;
3. to receive specific commands able to start and stop its movements.

So, generally speaking, its behaviour follows both an event-driven model and a message-based model.

Console and GUI

Structure. The console is a reactive computational entity running on a PC connected to the system through the network. He is associated with a GUI.

Interaction. The console interacts with the robot and with the N sonars placed along the path of the robot. In particular, the console receives the data of the photos taken by the robot and gets the data emitted by the sonar in order to calculate the value of the distance expression. Moreover, the console interacts with the robot by sending it commands to start and stop its movements.

Behaviour. The console simply display through the GUI associated with it the data acquired and sent by the robot. Moreover, it plays an alarm sound if the value of the expression calculated by it is less than the prefixed threshold. So, its behaviour is reactive with respect to the data sent by the sonar and active in the way it evaluates them.

Sonar subsystem

Structure. With the term *sonar subsystem* we refer to the set of sonars placed along the path crossed by the robot. The sonar subsystem is obviously a composed entity consisting of N sonars. Each sonar is an independent input entity connected to a Raspberry.

Interaction. Each sonar in the sonar subsystem interacts with the console by sending to it the value of the

distance from an eventual object placed in the sonar area.

Behaviour. The sonar subsystem has an active behaviour. Its job is to sense the environment around it in order to detect eventual object in its area, to get the distance from it and to send the data to the console.

5.5 Test plan

We conclude the phase of requirement analysis by reporting a first plan of tests that will be performed in a simulated way (even with dummy data) once the system has been implemented. We will perform first the unit testing for each entity, then the integration testing in order to check that the system behaves as expected.

Unit testing

The unit testing verifies that each entity of the system works correctly when isolated from the rest of the units.

1. Robot
   (a) The robot moves and rotates without problems (this can be done manually)
   (b) The sonar on the robot is able to detect correctly obstacles
   (c) The led on the robot blinks correctly
   (d) The robot takes photos
2. Console and GUI
   (a) The distance expression is evaluated correctly
   (b) The alarm sound is played
   (c) The GUI is able to show the data
3. Sonar subsystem
   (a) Each sonar is able to detect correctly an object in its area

Integration testing

The integration testing verifies that units created and tested independently can coexist and interact among themselves.

1. The console sends a START command to the robot, the robot starts to move.
2. The console sends a STOP command to the robot, the robot stops itself.
3. A sonar in the sonar subsystem detects an object, sends the distance value to the console and the consoles evaluates the distance expression correctly.
4. The console plays a sound alarm if the value calculated by the distance expression is less than the prefixed threshold.
5. A sonar in the sonar subsystem detects the robot, the robot turns on its left, the led starts to blink, the robot takes a photo and sends it to the console.
6. After taking the photo, the robot turns on its right to compensate the previous move, the led stops to blink and the robot starts to move again towards the area B.
7. The robot stops itself when it reaches the area B.

6 Problem analysis

```
/*
 * ============================================================
 * robot.ddr Robot
 * ============================================================
 */

RobotSystem robot

Event obstacle : obstacle(X) /* From the console and from the sonar on the robot */
Event alarm    : alarm       /* Emitted when distance from sonar is less than the threshold DMIN */
Event usercmd  : usercmd(CMD) /* From the user GUI */
Event sonarArea : sonarArea(N) /* From the console (which sensed it by the sonar subsystem) */

Context ctxRobot ip [ host="localhost" port=8070 ] -httpserver
Context ctxConsole ip [ host="192.168.251.1" port=8033 ] -httpserver

EventHandler evh for obstacle, alarm -print ;

/* ROBOT */
Robot robot0 QActor robot context ctxRobot { /* "robot0" will be the name of the configuration file */

   /* First plan */
      Plan main normal
      println( "Ready to start!") ;
      switchToPlan connectToMqttServer ;
      switchToPlan waiting ;

   /* Connects to MQTT server */
   Plan connectToMqtt resumeLastPlan
      println( "Connecting to MQTT server..." ) ;
      actorOp connectToMqttServer( "mqtt_robot_system", "tcp://m2m.eclipse.org:1883", "photo" ) ;
      [ ?? tout(X,Y) ] switchToPlan timeoutExpired

   /* Waiting state of the robot */
   Plan waiting
      println( "Waiting for a command..." ) ;
      sense time(30000) usercmd -> continue ; /* Blocks the execution of the robot until the required
          event occurs or timeout happens */
      [ !? tout(30000,Y) ] switchToPlan timeoutExpired ;
      printCurrentEvent ;
      onEvent usercmd : usercmd(robotgui(w(S))) -> switchToPlan moveForward ; /* START/FORWARD
          command */
      onEvent usercmd : usercmd(robotgui(h(S))) -> switchToPlan stop ; /* STOP command */
      repeatPlan

   /* (Req. 1.a.i) Moves forward the robot and handles the event that the robot can perceive while
       moving */
   Plan moveForward
      println( "Moving forward..." ) ;
      robotForward speed(40) time(20000) react event usercmd -> checkUserCommand /* A command from
          console arrives */
                                or event alarm   -> stop             /* The consoles has played an
                                    alarm */
                                or event obstacle -> stop            /* There is an obstacle */
                                or event sonarArea -> handlePhotoShoot; /* The robot has reached the
                                    area of a sonar */
      repeatPlan

   /* Checks the command received by the robot while it was moving */
   Plan checkUserCommand
```

```
      println( "Checking user command..." ) ;
      onEvent usercmd : usercmd(robotgui(h(S))) -> switchToPlan stop ; /* STOP command */

   /* (Req. 1.a.iii) Stops the robot movements */
   Plan stop
      robotStop speed(0) time(0) ;
      println( "Robot stopped!" ) ;
      switchToPlan waiting

   /* Handles the photo shoot routine */
   Plan handlePhotoShoot
      robotStop speed(0) time(0) ; /* Stop the robot */
      println( "The robot is going to take a photo..." ) ;
      robotLeft speed(70) time(2000) ; /* Rotate to the left */
      actorOp startLedBlink ; /* (Req. 1.c) Start led blinking */
      println( "Led started to blink" ) ;
      switchToPlan takeAndSendPhoto ;
      robotRight speed(70) time(2000) ; /* Turn back on its original position */
      actorOp stopLedBlink ; /* Stop led blinking */
      println( "Led stopped blinking" ) ;
      switchToPlan checkReachedArea ;
      switchToPlan moveForward

   /* (Req. 1.d) Takes a photo and sends it to the console */
   Plan takeAndSendPhoto resumeLastPlan
      [ !? msg( reachedSonarArea, event, console, none, sonarArea(K), N) ] actorOp
          takeAndSendPhoto(K) ;
      println( "Photo shooted and sent to the user!" ) ;

   /* Checks which area the robot has reached */
   Plan checkReachedArea resumeLastPlan
      [ ?? msg( reachedSonarArea, event, console, none, sonarArea(0), N) ] switchToPlan
          moveTowardsAreaB ;

   /* The robot is going to reach the area B */
   Plan moveTowardsAreaB
      robotForward speed(40) time(30000) react event usercmd -> checkUserCommand /* A command from
          console arrives */
                               or event obstacle -> stop          /* There is an obstacle */
      robotStop speed(0) time(0) ;
      println( "The robot has reached the area B!" ) ;
      switchToPlan waiting

   /* Alerts that a timeout expired */
   Plan timeoutExpired
      println( tout(X,Y) ) ;
}


/* SONAR */
QActor robotsonar QActor context ctxRobot {

   /* First plan */
      Plan main normal
         switchToPlan init ;

   /* Activates the sonar */
   Plan init
      demo loadTheory( "./robotSonarKB.pl" ) onFailSwitchTo prologFailure ;
      [ !? simulate ] switchToPlan simulateMode ; /* Simulate the work */
      [ !? physicSonar ] switchToPlan realMode ; /* Real sonar */
      println( "Sonar on robot starts working.") ;
```

```
   /* Simulates the sonar work */
   Plan simulateMode
      println( "Simulate mode" ) ;

   /* Manages the real physic sonar */
   Plan realMode
      println( "Working with physic sonar" ) ;
      actorOp startRobotSonar ;
      switchToPlan working

   Plan working
      [!? obstacle(X) ] emit obstacle : obstacle(X) ;
      actorOp getDistanceFromSonar ;
      delay(500) ;
      repeatPlan

   Plan prologFailure resumeLastPlan
      println( "Prolog goal fails" )
}
```

```
/*
 * ============================================================
 * radarEConsole.qa Console and GUI (radar)
 * ============================================================
 */

System console

Event obstacle : obstacle(X) /* Emitted when the sonar subsystem or the sonar on the robot detect an
    obstacle */
Event alarm    : alarm       /* Emitted when distance from sonar is less than the threshold DMIN */
Event usercmd  : usercmd(CMD) /* Emitted when a human user send a command through the panel */
Event inputcmd : usercmd(CMD) /* Emitted when a human user send a manual command */
Event sonarArea : sonarArea(N) /* Perceived and emitted from the sonar subsystem when the robot
    reaches the area of a sonar */

Dispatch polar      : p(DIS, ANG)
Dispatch robotCommand : rc(Command)
Dispatch mqttmsg    : mqttmsg

Context ctxConsole ip [ host="192.168.251.1" port=8033 ] -httpserver

/* RADAR GUI */
QActor radargui context ctxConsole {

   /* First plan */
   Plan main normal
      println( "Starting radar GUI..." ) ;
      actorOp activateGui ; /* Activate the GUI */
      switchToPlan doWorkMsgs

   Plan doWorkMsgs
      receiveMsg time(30000000) ;
         onMsg polar : p(DIS,THETA) -> actorOp sendDataToGui(DIS,THETA) ;
         printCurrentMessage ;
      repeatPlan
}

/* CONSOLE */
QActor console context ctxConsole {

   Plan init normal
      println("Inizio controller ");
```

```
    switchToPlan inizio

  Plan inizio
    actorOp inizializzazione; //setta tutti i vari parametri valori a 0
    switchToPlan work

  Plan work
    println("Sto aspettando un comando");
    receiveMsg time ( 30000000 ) react event usercmd ->checkCmd ;
    onMsg mqttmsg : mqttmsg -> actorOp retrieveAndSavePhoto ; //per la foto del robot
    onMsg polar : p( Distance , SID ) -> actorOp evaluateExpr; //per i sonar
    printCurrentMessage ;
    repeatPlan

  Plan checkCmd
    [?? actorOpDone (OP ," start ")] switchToPlan inizio ; // se gli ritorna il comando star,
        ricomincia
    [?? actorOpDone (OP ," stop ")] switchToPlan stopTheRobot ;//stoppa il robot
    [?? actorOpDone (OP ," alarm ")] switchToPlan alarmSound ; //allarme
    resumeLastPlan


  Plan stopTheRobot
    println("Stop the robot!!!");
    emit usercmd : usercmd(robotgui(h(S))) ; //manda un messaggio al robot per dire di fermarsi
    resumeLastPlan

  Plan alarmSound
    println("An alarm sound is playing !!!");
    sound time (3000) file ("./audio/illogical_most2 .mp3");
    emit usercmd : usercmd(robotgui(h(S))) ; //Il robot, come da requisiti dopo l'allarme si deve
        fermare
    resumeLastPlan

  Plan movoToTakePhoto
    emit stopAndTakePhoto : stopAndTakePhoto ;
  // forward robot -m robotCommand :rc( left );
  // forward robot -m robotCommand :rc( blinkLedStart );
  // forward robot -m robotCommand :rc( takePhoto );
  // forward robot -m robotCommand :rc( right );
  // forward robot -m robotCommand :rc( blinkLedStop );
  // forward robot -m robotCommand :rc( goForward );
    resumeLastPlan

}
```

6.1 Logic architecture

—TO DO

6.2 Abstraction gap

—TO DO

6.3 Risk analysis

—TO DO

7 Work Plan

—TO DO

8 Project

—TO DO

9 Implementation

—TO DO

10 Testing

—TO DO

11 Deployment

—TO DO

12 Maintenance

—TO DO

13 About authors

—TO DO