**Using the qa (domain specific) language**

Home

VISION

* A (domain specific) language can be tailored to the main concepts needed in an application domain, by allowing the specification of *executable models*. Such a language should help software developers in writing *executable specifications* during the early stages of software development with particular regard to *requirement and problem analysis*.

GOALS

Show (by means of concrete examples/applications) how the logical architecture and the project architecture of a (distributed) software system can be specified by a custom (domain specific) language (**qa**) and how a proper software factory can reduce the costs and the risks of software development.

**Workflow**

The main aim of the custom (domain specific) language **qa** is to give more expressive power (than UML) for the definition of system models, both during the problem analysis phase and during the project phase.

The language **qa** allows us to **specify**

* the main components (**QActor**) of a software system
* how the components are mapped into computational nodes (**Context**)
* how the components exchange information (using **Messages** and **Events**)
* how each component works (**behavior**)

The point is that we have to specify a **model**, i.e. we must carefully understand the detail level of the specification.

A general *workflow* can be set:

1. find the main subsystems and dene the system **Contexts**;
2. define the structure of the **Events** that can occur in the system;
3. define the structure of the **Messages** exchanged by the actors;
4. define the main **Actors** working in each Context;
5. define the type of the **logical interaction** among the actors;
6. define the **logical behaviour** of each actor according to the interaction constraints.

The starting point should be always the model that express the result of the problem analysis phase, i.e. the logical architecture of the system. In a second step, this specifications can be refined in the project phase, hopefully by simply 'injecting' in the model some application-specifications. Since we follow an interactive, incremental software development process, the logical architecture of the system could be changed according to better understanding of user requirements during the project phase. The general idea however is:

* to work always at model level, by considering the model as a **new kind of source code**
* to privilege a **delcarative style** rather than an imperative one. For example the behavior is expressed by **Plans** rather than procedures or methods
* to reduce the **global costs** of software development, at least for the realization iof a first working prototype
* to have a clear understandings of the **costs and risks** related to the development of critical parts of the system

About actions and zooming

The general idea is that a **qa** model should focus the reader's attention of **WHAT** rather than on **HOW**.

Thus, the main goal of a model should to express **WHAT and WHEN a component has to perform an action**, under the constraints imposed by the interaction with the other components.

The **concrete specification** of an action should be left to another (more detailed) level of model specification (**zooming**) or even left to the application designer, once the effects of the action are clearly understood.

Very often, an action can be better written in Java or in Prolog and simply referred (via **actorOp** or **solve**) at model level.

**QActor (qa DSL) examples**

For an introduction to the **qa** lanague, please read Qactors.pdf

In this lab we will present examples and demo related to the main features of QActors.

1. Logical architecture and fast prototyping

The specification of a model of the **logical architecture** of a system is a fundamental precondition for each type of software production process.

Automatic code generation

With a proper MDSD (Model-driven software development) approach, it is possible to build a software factory with we can trasnform - in a short time - the specification of a logical architecture into a working prototype of the software system. Please look at the code generated in the referred examples.

Graphical interfaces

Fast prototyping should not exclude completely the construcion of working systens in which actors (or at least contexts) can be associated to a GUI. See the **-g** context / actor flag.   
Let us summarize here the events generated by the buil-in GUI interfaces:

Event local\_inputcmd : usercmd(X) //generated by cmd actor gui-interface

Event inputcmd : usercmd(X) //generated by the [Web cmd user-interface](https://137.204.107.21/syskb/it.unibo.iss2015intro/docs/Lab/Lab2016/qaIntroAndExamples.html#webGui)

Event alarm : alarm(X) //generated by [Web cmd user-interface](https://137.204.107.21/syskb/it.unibo.iss2015intro/docs/Lab/Lab2016/qaIntroAndExamples.html#webGui)

Event obstacle : obstacle(X) //generated by [Web cmd user-interface](https://137.204.107.21/syskb/it.unibo.iss2015intro/docs/Lab/Lab2016/qaIntroAndExamples.html#webGui)

1. Structure

Distributed systems are composed of a set of active entities (**Qactor**) each working in a (different) computational environment (**Context**).

1. Behavior

A QActor is an active (autonomous) entity that runs in parallel with the other actors defined in the same or in other contexts. The behavor of a Qactor can be expressed as a Moore Finite State Machine (**M-FSM**). Each state of the **M-FSM** can be expressed by a **Plan** is composed of a sequence of **Moves**. See Plans

1. Interaction

A QActor can interact with others (local or remote) QActor by sending/receiving messages.   
A QActor can also emit or sense **events**.

1. Message based rather than message-driven

A QActor is able to call message **receive actions** in explicit way, with the possibility to specify the structure of the messages expected.

1. Event based and event-driven

A QActor system can express both some event-driven behavior while allowing actors to call event **sense actions** in explicit way, with the possibility to specify the structure of the events expected.

1. Actions

The high level specification of the behavior of a Qactor is only a **partial** specification that aims at capturing the behavior logic. The details related to specific moves (in particular GUI interfaces) can be better expressed in the implementation language directly as application **Action**.

1. Declarative programming

Several task (including the evaluation of guards) can be better expressed by means of declarative rules. A QActor can exploit logic programming to express in synthetic way algorithms and policies.

1. World theory

Each Qactor works in a (real or simulated) world. The state of the world can be represented by a set of lcal variable or objects. A higher level representation can be based on a symbolic representation of the world expressed in declarative way as a set of Prolog facts and rules (**World theory**).   
  
The **World theory** can be used as a local knowledge base whose current content can drive the execution of moves (including a state transition) by a Qactor by means of **Guards**. See **WorldTheory.pl**.

1. Reactive Actions

A Qactor should behave not only in **proactive** but also in **reactive** way. Thus any move of the Qactor should be 'interrupted' before its normal end.   
The availability of predefined reactive moves (e.g. like [ActionSoundTimed](https://137.204.107.21/syskb/it.unibo.qactors/src/it/unibo/qactors/action/ActionSoundTimed.java)) can be useful for fast prototyping and for better human interactions.

1. Web interface

Nowadays the interaction between an user and a software system is based not only on some GUI working on the system node, but can be also a Web-based interaction. A Web-based interface should be automatically provided by a system. See **-http** context flag

1. Standalone
2. See [One button many leds](https://137.204.107.21/syskb/it.unibo.iss2015intro/docs/Lab/Lab2016/qaIntroAndExamples.html#stdAloneBls)
3. Interpreted behaviour

A code generator (provided by an IDE) can produce intermediate code to be executed by an interpreter. If this intermediate code can be understood by the application designer, the behavior of actors can be modified (for debugging/setup purposed) without the usage of the IDE.

**Projects**

 [it.unibo.test.sf](https://137.204.107.21/syskb/it.unibo.test.sf/)

 [it.unibo.bls2016.qa](https://137.204.107.21/syskb/it.unibo.bls2016.qa/)

 [it.unibo.radar](https://137.204.107.21/syskb/it.unibo.radar/)

**Applications**

Led controlled via web

Build a sofware system in which a Led connected to a RaspberryPi is controlled by a remote console implmented as a web page   
See

 ledWeb.qa

|  |
| --- |
| /\*  \* ==============================================================  \* ledweb.qa in project it.unibo.test.sf  \* ==============================================================  \*/  System ledWeb  Event inputcmd : usercmd(X)    Context ctxWebLed ip [host="localhost" port=8010] -httpserver  QActor ledonweb context ctxWebLed -g yellow {  Plan init normal  println("ledonweb STARTS " ) ;  actorOp createLed ; //configure  switchToPlan work  Plan work  sense time(3000000) inputcmd -> continue ;  printCurrentEvent ;  onEvent inputcmd : usercmd(executeInput(switch)) -> actorOp switchTheLed ;  repeatPlan  } |

Many leds that perceive command events (standalone console)

A console must control a dynamic set of remote leds, each connected to a RaspberryPi.   
The console is a stand alone context including a QActor with abutton that emits a **usercmd** event.   
Each Led works within a systen including the console as a **-standalone** context.   
See  ledStdAlone1.qa

|  |
| --- |
| /\*  \* ==============================================================  \* ledStdAlone1.qa in project it.unibo.test.sf  \* ==============================================================  \*/  System ledStdAlone1  Event inputcmd : usercmd(X)    Context ctxLedStdAolone1 ip [host="localhost" port=8011] -g white  Context ctxConsoleStandalone ip [ host="localhost" port=8088 ] -standalone  QActor ledstdalone1 context ctxLedStdAolone1 {  Plan init normal  println("ledForConsoleStandalone STARTS " ) ;  actorOp createLed ; //configure  switchToPlan work  Plan work  sense time(3000000) inputcmd -> continue ;  printCurrentEvent ;  onEvent inputcmd : usercmd(executeInput(switch)) -> actorOp switchTheLed ;  repeatPlan  } |

 ledStdAlone2.qa

|  |
| --- |
| /\*  \* ==============================================================  \* ledStdAlone2.qa in project it.unibo.test.sf  \* ==============================================================  \*/  System ledStdAlone2  Event inputcmd : usercmd(X)    Context ctxLedStdAolone2 ip [host="localhost" port=8012] -g white  Context ctxConsoleStandalone ip [ host="localhost" port=8088 ] -standalone  QActor ledstdalone2 context ctxLedStdAolone2 {  Plan init normal  println("ledForConsoleStandalone STARTS " ) ;  actorOp createLed ; //configure  switchToPlan work  Plan work  sense time(3000000) inputcmd -> continue ;  printCurrentEvent ;  onEvent inputcmd : usercmd(executeInput(switch)) -> actorOp switchTheLed ;  repeatPlan  } |

 consoleStandalone.qa

|  |
| --- |
| System consoleStandalone  /\*  \* ==============================================================  \* consoleStandalone.qa in project it.unibo.test.sf  \* ==============================================================  \*/  Event local\_inputcmd : usercmd(X) //generated by cmd actor gui-interface  Event inputcmd : usercmd(X) //generated by the Web cmd user-interface  Event alarm : alarm(X) //generated by Web cmd user-interface  Event obstacle : obstacle(X) //generated by Web cmd user-interface    /\*  \* -----------------------------------------------  \* WARNING: compile for last system component  \* -----------------------------------------------  \*/  Context ctxConsoleStandalone ip [ host="192.168.43.229" port=8088 ] -g cyan  EventHandler evh for inputcmd -print ;    QActor consolestandalone context ctxConsoleStandalone {  Plan init normal  println("consolestandalone STARTS");  actorOp createConsoleEmittingEvent  } |

Many leds that register themselves to a console

A console must control a dynamic set of remote leds, each connected to a RaspberryPi. Each led must 'register' to the console in order to be known by the console. The console sends a message to each registered Led

Ping pong with arbiter

Two players exchange information until an **alarm** event is raised by an arbiter.  
See  pingpong.qa

|  |
| --- |
| /\*  \* In the ping-pong system, two actors work by exchanging each other a dispatch  \* During their behavior they must be able to react as soon as possible  \* to an event emitted by an arbiter.  \*  \* To test the system in some predictable way,  \* the event could be emitted by one of the two players  \*/  System pingPong  Dispatch hit:hit(X)  Dispatch playerready:playerready(X)    Event usercmd:usercmd(X)  Event arbitercmd:arbitercmd(X)    Context ctxarbiter ip [ host="localhost" port=8030 ] -g white -httpserver //192.168.43.229 //192.168.42.2 Tether  EventHandler evh for arbitercmd -print;    Context ctxping ip [ host="localhost" port=8040 ] -g cyan //192.168.43.229  EventHandler evh for arbitercmd -print { memo currentEvent for ping};    Context ctxpong ip [ host="localhost" port=8050 ] -g yellow //192.168.43.71 /192.168.42.129 Tether  EventHandler evh for arbitercmd -print { memo currentEvent for pong};    /\*  \* ============================================  \* arbiter  \* ============================================  \*/  QActor arbiter context ctxarbiter {  Plan init normal  println("arbiter waits for some player to become ready... ") ;  receiveAndSwitch time(10000) playerready -> waitForOtherPlayer  Plan waitForOtherPlayer  printCurrentMessage ;  println("arbiter waits for the other player to become ready... ") ;  receiveAndSwitch time(10000) playerready -> startToWork  Plan startToWork  printCurrentMessage ;  delay time(2000) ;  println("arbiter starts the game ...") ;  emit arbitercmd:arbitercmd(start) ;  switchToPlan observeTheGame  Plan observeTheGame  println("arbiter emits a stop after some secs from now ") ;  delay time(3000) ;  emit arbitercmd : arbitercmd(stop) ;  sound time(1500) file( "./audio/tada2.wav") ;  println("arbiter ends")  }  /\*  \* ============================================  \* Ping works as sender  \* ============================================  \*/  QActor ping context ctxping {  Plan init normal  println("ping sends the ready message to the arbiter ") ;  forward arbiter -m playerready:playerready("ping");  switchToPlan waitstart  Plan waitstart  println("ping waits for the arbiter start event ... ") ;  sense time(60000) arbitercmd -> continue;  onEvent arbitercmd : arbitercmd(start) -> switchToPlan playTheGame  else repeatPlan  Plan playTheGame  // println("ping sends the ball (if not stopped) ") ;  [ not !? msg( arbitercmd, 'event', arbiter, none, arbitercmd(stop) , N ) ]  forward pong -m hit:hit("ping") ;  // println("ping receiving (if not stopped) ... ") ;  [ not !? msg( arbitercmd, 'event', EMITTER, none, arbitercmd(stop) , N ) ]  receiveMsg time(60000) react event arbitercmd -> reactArbiter or event usercmd -> reactArbiter ;  printCurrentMessage ;  delay time(500) ;  [ !? msg( arbitercmd, 'event', EMITTER, none, arbitercmd(stop) , N ) ]  endPlan "ping ends since arbiter has stopped"  else repeatPlan 15  Plan reactArbiter  println("\*\*\* REACT ping while waiting for a hit")    }  /\*  \* ============================================  \* Ping works as receiver  \* ============================================  \*/  QActor pong context ctxpong {  Plan init normal  println("pong sends the ready message to the arbiter ") ;  forward arbiter -m playerready:playerready("pong");  println("pong waits for the arbiter start ... ") ;  sense time(60000) arbitercmd -> continue; //arbiter -> reactArbiterStart ;  onEvent arbitercmd : arbitercmd(start) -> switchToPlan playTheGame  else repeatPlan  Plan playTheGame  /\*  \* conceptually, all the moves are guarded by [ not !?arbiter(stop) ]  \* but a player can always do some action outside the control of the arbiter ...  \*/  [ !? msg( arbiter, 'event', EMITTER, none, arbiter(stop) , N ) ] endPlan "pong ends since arbiter has stopped" ;  // println("pong waits for the ball ... ") ;  receiveTheMsg m( Hit, "dispatch", Sender, "pong", MSG, N ) time(10000) react event arbitercmd -> reactArbiter ;  printCurrentMessage ;  onMsg hit : hit(ping) -> switchToPlan handleMsg ;  delay time(500) ;  repeatPlan 15  Plan handleMsg resumeLastPlan  println("pong reply to ping (if not stopped) ") ;  [ not !? msg( arbitercmd, 'event', EMITTER, none, arbitercmd(stop) , N ) ] forward ping -m hit:hit("pong")  Plan reactArbiter  println("\*\*\* REACT pong ")  } |

Morse

Design a build a software system that

R1) reads a sentence InputS from a user input device (GUI)

R2) converts the InputS sentence into a sequence of morse symbols

R3) blinks a Led according to a morse-symbols sequence (morse-rendering)

R4) terminates its current morse-rendering if the user sends a STOP command

Game

See  hitTheLight.qa

|  |
| --- |
| /\*  ======== REQUIREMENTS ========================================================  R1)  R1a) Inizialmente tutti i led Li sono spenti  R1b) il led Lflash pulsa, con un certo periodo.  R1c) Il punteggio del giocatore P è inizializzato a zero.  R2)  R2a) Alla pressione del pulsante T1, parte il gioco:  R2b) il led Lflash si spegne e i tre led devono accendersi  a turno (prima L1, poi L2, poi L3, poi L2, poi L1, poi L2..)  simulando il movimento di una luce, avanti e indietro.  R2c) La velocità con cui si “muove” la luce è regolata dal potenziometro Pot  - la regolazione deve avvenire prima che parta il gioco.    R3)  R3a) L’animazione deve continuare per un certo tempo DTstop casuale,  R3b) passato il quale l’animazione si ferma  R3c) e il led corrispondente Lj rimane acceso per un certo DTthreashold.  R3d) A questo punto il giocatore deve reagire,premendo il tasto corrispondente Tj.  R4)  R4a) Se il pulsante non è premuto entro il tempo soglia DTthreashold  R4b) oppure se viene spinto un altro pulsante diverso da Tj,  R4c) il gioco termina, inviando un messaggio sulla seriale  “Game over - Punteggio finale: P”, dove P è il punteggio finale.  R5)  R5a) Se invece il pulsante viene premuto entro il tempo DTthreashold,  R5b) allora Lflash viene acceso per 0.25 secondi  R5c) e il punteggio P del giocatore viene incrementato di (DTthreshold - DTreact),  dove DTreact è il tempo impiegato dal giocatore per reagire (0 <= DTreact <= DTthreshold).  R5d) In questo caso il gioco procede con un nuovo turno, con DTthreashold dimezzato.  \*/  System hitTheLight  Event startTheGame : startTheGame //emitted by pressing button t1  Event userCmd : userCmd(BUTTON) //emitted by pressing any button  Context ctxGame ip [ host="localhost" port=8057 ] -g cyan  QActor playcontrol context ctxGame {  Plan main normal  actorOp createUserConsole //creates a GUI with buttons and leds  onFailSwitchTo failure ;  switchToPlan startSystem  Plan startSystem  actorOp ledsOff; //R1a)  actorOp setParams; //R1c)  actorOp readPot; //R2c)  actorOp printStartGameMsg ;  actorOp ledFlashPulse react event startTheGame -> startGame //R1b) R2a) (t1)    Plan startGame  actorOp ledFlashTurnOff ; //R2b)  switchToPlan doGame ;  switchToPlan continueGame //R5a)    Plan doGame resumeLastPlan  actorOp ledsPulseRandom ; //R2b) R2c)(pot) R3a)(dtstop) TIMED ACTION  actorOp getDTthreashold ; //R5d)  [ ?? actorOpDone( getDTthreashold,DTTH) ] //R3c)  sense time(DTTH) userCmd -> checkUserCmd ; //R3d) REACTIVE ACTION  [ ?? tout(X,Y) ] switchToPlan endTheGame //34a)    Plan checkUserCmd resumeLastPlan  actorOp checkCmd ; //returns true if the plays does the right choice  [ ?? actorOpDone( checkCmd,false) ] switchToPlan endTheGame //34b)  else actorOp updateScore //35c)  //We return (resumeLastPlan) since we do not have tail recursion    Plan continueGame resumeLastPlan  actorOp ledFlashPulse25 ; //R5b)  switchToPlan doGame ;  repeatPlan //R5a) terminates when doGame does not return    Plan endTheGame  actorOp printEndGameMsg ; //R4c)  actorOp ledsOff; //ADDED BY THE DEVELOPER  switchToPlan checkRestart //ADDED BY THE DEVELOPER TO RESTART THE GAME    Plan checkRestart  println("you can restart within 5 sec by pressing again " ) ;  sense time(5000) userCmd -> continue ;  [ ?? tout(X,Y) ] switchToPlan endSystem ;  onEvent userCmd : userCmd(again) ->  switchToPlan startSystem ; //stack grows but it is tolerable  println( "you DID NOT press again" ) ;  switchToPlan endSystem  Plan endSystem  println( "endTheSystem: BYE BYE" )  Plan failure  println("playcontrol has failed " )  } |

worldTheory.pl

|  |
| --- |
| %==============================================  % WorldTheory.pl for actor soundselector  %==============================================  /\*  For a QActor as a singleton statically degined in the model  \*/  myname(qatusoundselector). %%old version (deprecated)  actorobj(qatusoundselector). %% see registerActorInProlog18 in QActor  /\*  For a QActor instance of name=Name dynamically created  \*/  setActorName( Name ):-  retract( myname(X) ),  retract( actorobj(X) ),  text\_term( TS,Name ),  text\_concat("qatu",TS,NN),  assert( myname(NN) ),  assert( actorobj(NN) ).  createActor(Name, Class, NewActor):-  actorobj(A),  A <- getName returns CurActorName,  A <- getContext returns Ctx,  A <- getOutputEnvView returns View,  Ctx <- addInstance(Name),  java\_object(Class, [Name,Ctx,View], NewActor),  NewActor <- getName returns NewActorName,  actorPrintln( createActor(NewActorName, NewActor) ).  /\*  Name generator  \*/  value(nameCounter,0).  newName( Prot, Name,N1 ) :-  inc(nameCounter,1,N1),  text\_term(N1S,N1),  text\_term(ProtS,Prot),  text\_concat(ProtS,N1S,Name),  assert( instance( Prot, N1, Name ) ),  actorPrintln( newname(Name,N1) ) .    setPrologResult( Res ):-  ( retract( goalResult( \_ ) ),!; true ), %%remove previous goalResult (if any)  assert( goalResult(Res) ).  addRule( Rule ):-  %%output( addRule( Rule ) ),  assert( Rule ).  removeRule( Rule ):-  retract( Rule ),  %%output( removedFact(Rule) ),  !.  removeRule( A ):-  %%output( remove(A) ),  retract( A :- B ),!.  removeRule( \_ ).  setResult( A ):-  ( retract( result( \_ ) ),!; true ), %%remove previous result (if any)  assert( result( A ) ).  evalGuard( not(G) ) :-  G, !, fail .  evalGuard( not(G) ):- !.  evalGuard( true ) :- !.  evalGuard( G ) :-  %stdout <- println( evalGuard( G ) ),  G .  output( M ):-stdout <- println( M ).  %-------------------------------------------------  % TuProlo FEATURES of the QActor soundselector  %-------------------------------------------------  dialog( FileName ) :-  java\_object('javax.swing.JFileChooser', [], Dialog),  Dialog <- showOpenDialog(\_),  Dialog <- getSelectedFile returns File,  File <- getName returns FileName.  %% :- stdout <- println( "hello from world theory of soundselector" ).  %-------------------------------------------------  % UTILITIES for TuProlog computations  %-------------------------------------------------  loop( N,Op ) :-  assign(loopcount,1),  loop(loopcount,N,Op).  loop(I,N,Op) :-  value(I,V),  %actorPrintln( values( I,V,N,Op ) ),  V =< N ,!,  %actorPrintln( loop( I,V ) ),  actorOp( Op ) ,  V1 is V + 1 ,  assign(I,V1) ,  loop(I,N,Op). %%tail recursion  loop(I,N,Op).  getVal( I, V ):-  value(I,V).  assign( I,V ):-  retract( value(I,\_) ),!,  assert( value( I,V )).  assign( I,V ):-  assert( value( I,V )).  inc(I,K,N):-  value( I,V ),  N is V + K,  assign( I,N ).  actorPrintln( X ):- actorobj(A), text\_term(XS,X), A <- println( XS ).  %-------------------------------------------------  % User static rules about soundselector  %-------------------------------------------------  /\*  ------------------------------------------------------------------------  testex :- actorPrintln( testex ),  java\_catch(  java\_object('Counter', ['MyCounter'], c),  %% java\_object('java.util.ArrayList', [], l),  [ (EEE, actorPrintln( a )) ],  %% ('java.lang.Exception'(Cause, Message, \_), actorPrintln( b )),  %% ('java.lang.ClassNotFoundException'(Cause, Message, \_), actorPrintln( c )) ],  actorPrintln( d )  ).  ------------------------------------------------------------------------  \*/  loadTheory(T):-  %% actorPrintln( loadTheory(T) ),  consult(T), !.  loadTheory(T):-  actorPrintln( loadTheory(T, FAILURE) ).    actorOp( Op ) :- actorobj(Actor),  % actorPrintln( actorOp( Op ) ),  java\_catch(  Actor <- Op returns R, %% R unbound if void  [ ( E, setActorOpResult( Op,failure ) )],  setActorOpResult( Op,R ) %%executed in any way  ).  actionResultOp( Op ):-  % actorPrintln( actionResultOp( Op ) ),  java\_catch(  actoropresult <- Op returns R, %% R unbound if void  [ ( E, setActorOpResult( Op,failure ) )],  setActorOpResult( Op,R ) %%executed in any way  ).  setActorOpResult( Op, V ):- unbound(V),!, setActorOpResult( Op,void ).  setActorOpResult( Op, failure ):-setActorOpResult( Op,failure ).  setActorOpResult( Op, Res ):-  ( retract( actorOpDone( \_,\_ ) ),!; true ), %%remove previous actionResult (if any)  cvtToString(Res,ResStr),  %% actorPrintln( actorOpDone( Op,ResStr ) ), %% Res is $obj\_xxx  assert( actorOpDone( Op, ResStr) ).  %% setActionResult : Actor register (in Java) the result under name 'actoropresult'  cvtToString( true , "true" ):- %% true is the result of robot move operation  setResult(true).  cvtToString( false , "false" ):-  setResult(false).  cvtToString( V , "void" ):-  unbound(V),!,  actorobj(Actor),  Actor <- setActionResult( void ).  cvtToString( V , S ):-  number(V),!,  setResult(V),  text\_term( S, V ).  cvtToString( V , S ):-  ( retract( result(\_) ),!; true ), %eliminate any previous numeric result  V <- toString returns S,  actorobj(Actor),  Actor <- setActionResult( V ),  !.  actorOpResult( V ) :- result(V),!.  actorOpResult( Res ) :- actoropresult <- toString returns Res, !.  %% actorOpResult( "" ).  androidConsult(T) :- actorobj(Actor), Actor <- androidConsult(T), !.  androidConsult(T) :- actorPrintln( failure(androidConsult) ).  /\*  ======================================================================  PLANS  ======================================================================  \*/  %%% --------- runPlan ---------------  runPlan(P):-  actorobj(Actor),  execPlan(Actor,P,1).  execPlan(Actor,P,PC) :-  plan(PC, P, S) ,  %% actorPrintln( execPlan(S) ),  ( runTheSentence(Actor,S),!  ; true ), %%failure should be related to guards  PC1 is PC + 1,  execPlan(Actor,P,PC1).  execPlan(\_,\_,\_).  %%% --------- loadPlan ---------------  loadPlan( FName ):-  actorobj(Actor),  Actor <- consultFromFile( FName ).  %%% --------- Showplan ---------------  executeCmd( Actor, move(showplan,P), Events, Plans ):-  %% actorPrintln( showPlan(P) ),  showPlan(P ).  showPlan( P ):-  showPlan( P,1 ).  showPlan( P, PC ):-  plan(PC, P, S) ,!,  actorPrintln( plan(PC, P, S ) ),  PC1 is PC + 1,  showPlan( P, PC1 ).  showPlan( \_,\_ ).  showPlan :-  curPlan(P),  showPlan(P).  %%% --------- storePlan ---------------  storePlan( FName,P ):-  actorobj( Actor ),  %% actorPrintln( storePlan( Actor, FName, P) ),  bagof( plan(PC, P, S) , plan(PC, pdefault, S) , L ),  %% actorPrintln( storePlan( Actor , FName, L) ),  Actor <- writeListInFile( FName ,L ).  %%% --------- clearPlan ---------------  clearPlan :-  curPlan(P),  %% class("it.unibo.robot.interpreter.RobotInterpreter") <- clearPC ,  retractall( plan(PC, P, S) ),  actorPrintln( clearPlan( P,done ) ).  /\*  ======================================================================  SENTENCES  ======================================================================  \*/  runTheSentence(Actor, sentence( not GUARD, MOVE, EVENTS, PLANS ) ):-  GUARD , !, setAnswer( failure ) .  runTheSentence(Actor, sentence( not GUARD, MOVE, EVENTS, PLANS ) ):-  !, runTheSentence(Actor, sentence( true, MOVE, EVENTS, PLANS ) ).  runTheSentence(Actor, sentence( GUARD, MOVE, EVENTS, PLANS ) ):-  %% actorPrintln( runTheSentence111( GUARD, MOVE, EVENTS, PLANS ) ),  ( GUARD = - G , !, retract(G), ! ; GUARD, ! ),  %% actorPrintln( sentence4( GUARD, MOVE, EVENTS, PLANS ) ),  ( executeCmd(Actor, MOVE, EVENTS, PLANS, RESULT), !,  %% actorPrintln( sentence4(RESULT) ),  setAnswer( RESULT );  %% actorPrintln( sentence4( failure ) ),  setAnswer( done(MOVE,failure) )  ).  runTheSentence(Actor, sentence( GUARD, MOVE, EVENTS, PLANS ) ):-  setAnswer( guard(GUARD,failed) ).    /\*  -----------------------------------  The case of Prolog Goal extended sentences  -----------------------------------  \*/  runTheSentence(Actor, sentence( GUARD, GOAL, DURATION, ANSWEREV, EVENTS, PLANS ) ) :-  ( GUARD = - G , retract(G), ! ; GUARD, ! ),  %% actorPrintln( sentence6( GUARD, GOAL, DURATION, ANSWEREV, EVENTS, PLANS ) ),  solvePrologRule(GOAL).  solvePrologRule(GOAL):-  actorPrintln( "WARNING: reactive prolog actions do not allowed in input" ),  /\*  actorobj(A),  actorPrintln( solvePrologRule( A,GOAL ) ),  A <- solveGoal( GOAL, DURATION, ANSWEREV, EVENTS, PLANS),!, %%tuProlog recursive call blocked  actorPrintln( solvePrologRuleDone( GOAL ) ),  goalResult(RESULT),  actorPrintln( solvePrologRuleDone( GOAL,RESULT ) ),  setAnswer( RESULT ).  \*/  GOAL,  setAnswer( GOAL ).  solvePrologRule(GOAL):-  %% actorPrintln( sentence6sprry( GOAL ) ),  setAnswer( done(GOAL,failure) ).  result("noresultyet").  %% Store answer (singleton) to current command  setAnswer( A ):-  ( retract( answer( \_ ) ),!; true ), %%remove previous answer (if any)  ( retract( result( \_ ) ),!; true ), %%remove previous result (if any)  addRule( result( A ) ),  addRule( answer( result(A) ) ).  /\*  ======================================================================  MOVES  ======================================================================  \*/  %%% --------- Actor move ---------------  executeCmd( Actor, move(robotmove,CMD,SPEED,DURATION,ANGLE), Events, Plans, RES ):-  !,  mapCmdToMove(CMD,MOVE),  %% actorPrintln( executeCmd(Actor,MOVE, SPEED, ANGLE, DURATION, Events, Plans) ),  Actor <- execute(MOVE, SPEED, ANGLE, DURATION, Events, Plans) returns AAR,  AAR <- getResult returns RES.  %%% --------- Play sound ---------------  executeCmd( Actor, move(playsound,FNAME,DURATION), Events, Plans, RES ):-  %% actorPrintln( executeCmd(Actor, playsound1( FNAME,DURATION ), Events, Plans) ),  !,  Actor <- playSound( FNAME, DURATION, Events, Plans ) returns AAR,  AAR <- getResult returns RES.  %%% --------- Photo ---------------  executeCmd( Actor, move( photo(DURATION,FNAME,ANSWEREVENT) ), Events, Plans, done(photo,FNAME) ):-  %% actorPrintln( executeCmd(Actor, photo( DURATION,FNAME,ANSWEREVENT )) ),  !,  Actor <- photo( FNAME, DURATION, ANSWEREVENT, Events, Plans ).  executeCmd( Actor, move( photo(DURATION,FNAME) ) , Events, Plans, done(photo,FNAME) ):-  %% actorPrintln( executeCmd(Actor, photo(DURATION,FNAME) )) ),  Actor <- photo( FNAME, DURATION, '', Events, Plans ).    %%% --------- Solve ---------------  /\*  We cannot solve a goal in asynch way while pengine is engaged  Thus we ignore DURATION and Events, Plans  \*/  executeCmd( Actor, move(solve,GOAL,DURATION, ANSWEREVENT), Events, Plans, RES ):-  %% actorPrintln( executeCmd(Actor, move(solve,GOAL,DURATION, ANSWEREVENT) ) ),  Actor <- solveSentence(sentence(true, GOAL, 0, ANSWEREVENT, '', '')) returns AAR,  AAR <- getResult returns RES.    executeCmd( Actor, move(solve,GOAL,DURATION), Events, Plans, RES ):-  %% actorPrintln( executeCmd(Actor, move(solve,GOAL,DURATION) ) ),  Actor <- solveSentence(sentence(true, GOAL, 0, '', '', '')) returns AAR,  AAR <- getResult returns RES.  %% actorPrintln( result(GOAL,RES) ).  %% The following DOES NOT WORK since pengine is engaged  %% Actor <- solveSentence(sentence(true, GOAL, DURATION, '', Events, Plans)) returns AAR.    %%% --------- Println ---------------  executeCmd( Actor, move(print,ARG), Events, Plans, done(print) ):-  text\_term(ARGS,ARG),  actorPrintln( ARGS ).  %%% --------- Emit ---------------  executeCmd( Actor, move(emit,EVID,CONTENT), Events, Plans, done(emit,EVID) ):-  actorPrintln( move(emit,EVID,CONTENT) ),  Actor <- emit( EVID,CONTENT ).  %%% --------- Forward ---------------  executeCmd( Actor, move(forward, DEST, MSGID, MSGCNT), Events, Plans, done(forward,MSGID,dest(DEST)) ):-  actorPrintln( forward( 1,Actor,MSGID,DEST,MSGCNT ) ),  text\_term(A1,DEST),  text\_concat("''",A1,A2),  text\_concat(A2,"''",DESTSTR),  Actor <- forwardFromProlog( MSGID , DESTSTR , MSGCNT ).    %%% --------- plan ---------------  executeCmd( Actor, move(runplan,P), Events, Plans,done(runplan(P)) ):-  %% actorPrintln( runplan(P) ),  execPlan(Actor,P,0).  executeCmd( Actor, move(resumePlan), Events, Plans,done(resume(P)) ).  /\*  %-------------------------------------------------  % Some predefined code  %-------------------------------------------------  \*/  fibo(0,1).  fibo(1,1).  fibo(2,1).  fibo(3,2).  fibo(4,3).  fibo(I,N) :- V1 is I-1, V2 is I-2,  fibo(V1,N1), fibo(V2,N2),  N is N1 + N2.  %% Fibonacci with cache (to be used in guards)  fib(V):-  fibmemo( V,N ),!,  actorPrintln( fib\_a(V,N) ).  fib(V):-  fibWithCache(V,N),  actorPrintln( fib\_b(V,N) ).  fib( V,R ) :-  fibWithCache(V,R).  fibmemo( 0,1 ).  fibmemo( 1,1 ).  fibmemo( 2,1 ).  fibmemo( 3,2 ).  fibWithCache( V,N ) :-  fibmemo( V,N ),!.  fibWithCache( V,N ) :-  V1 is V-1, V2 is V-2,  fibWithCache(V1,N1), fibWithCache(V2,N2),  N is N1 + N2,  %% actorPrintln( fib( V,N ) ),  assert( fibmemo( V,N ) ). |

1. **Robot usage**:  
   Build a software system that sends **commands** to move a (nano) [Robot](https://137.204.107.21/syskb/it.unibo.iss2015intro/docs/Projects/Robots/RobotNano0.jpg):  
   [BasicRobotUsageNaive.java](https://137.204.107.21/syskb/it.unibo.lab.baseRobot.example/src/it/unibo/robotUsage/naive/BasicRobotUsageNaive.java) : a basic robot that executes some moves and handles a distance sensor

|  |
| --- |
| /\*  \* ======================================================================================  \* The file ./hardwareConfiguration.properties must contain the robot name:  configuration=mocksimple  \* The file ./configuration/mocksimple/iotRobot.properties must contain the robot configuration  \* ======================================================================================  \*/  package it.unibo.robotUsage.naive;  import it.unibo.iot.baseRobot.hlmodel.BasicRobot;  import it.unibo.iot.baseRobot.hlmodel.IBasicRobot;  import it.unibo.iot.executors.baseRobot.IBaseRobot;  import it.unibo.iot.models.commands.baseRobot.BaseRobotBackward;  import it.unibo.iot.models.commands.baseRobot.BaseRobotForward;  import it.unibo.iot.models.commands.baseRobot.BaseRobotLeft;  import it.unibo.iot.models.commands.baseRobot.BaseRobotRight;  import it.unibo.iot.models.commands.baseRobot.BaseRobotSpeed;  import it.unibo.iot.models.commands.baseRobot.BaseRobotSpeedValue;  import it.unibo.iot.models.commands.baseRobot.BaseRobotStop;  import it.unibo.iot.models.commands.baseRobot.IBaseRobotCommand;  import it.unibo.iot.models.commands.baseRobot.IBaseRobotSpeed;  import it.unibo.iot.sensors.ISensor;  import it.unibo.iot.sensors.ISensorObserver;  import it.unibo.system.SituatedPlainObject;  public class BasicRobotUsageNaive extends SituatedPlainObject{  private IBaseRobot robot ;  private int delay= 1500;  public void doJob() throws Exception{  IBasicRobot basicRobot = BasicRobot.getRobot();  robot = basicRobot.getBaseRobot();  addObserverToSensors(basicRobot);  moveTheRobot();  Thread.sleep(delay\*6);  println( "RobotUsageNaive ENDS " +  robot.getDefStringRep() + " | " + robot.getJsonStringRep() );  System.exit(1);  }  protected void addObserverToSensors(IBasicRobot basicRobot){  ISensorObserver observer = new SensorObserver(outEnvView);  for (ISensor<?> sensor : basicRobot.getSensors()) {  println( "doJob sensor= " + sensor.getDefStringRep() + " class= " + sensor.getClass().getName() );  sensor.addObserver(observer);  }  }  protected void moveTheRobot(){  IBaseRobotSpeed SPEED\_LOW = new BaseRobotSpeed(BaseRobotSpeedValue.ROBOT\_SPEED\_LOW);  IBaseRobotSpeed SPEED\_MEDIUM = new BaseRobotSpeed(BaseRobotSpeedValue.ROBOT\_SPEED\_MEDIUM);  IBaseRobotSpeed SPEED\_HIGH = new BaseRobotSpeed(BaseRobotSpeedValue.ROBOT\_SPEED\_HIGH);  try {  IBaseRobotCommand command = new BaseRobotForward(SPEED\_HIGH);  println( "Executing ... " + command.getDefStringRep() );  robot.execute(new BaseRobotForward(SPEED\_HIGH));  Thread.sleep(delay);  command = new BaseRobotLeft(SPEED\_MEDIUM );  println( "Executing ... " + command.getDefStringRep() );  robot.execute(command);  Thread.sleep(delay);  command = new BaseRobotBackward(SPEED\_HIGH) ;  println( "Executing ... " + command.getDefStringRep() );  robot.execute(command);  Thread.sleep(delay);  command = new BaseRobotRight(SPEED\_MEDIUM ) ;  println( "Executing ... " + command.getDefStringRep() );  robot.execute(command);  Thread.sleep(delay);  command = new BaseRobotStop(SPEED\_LOW) ;  println( "Executing ... " + command.getDefStringRep() );  robot.execute(command);  } catch (InterruptedException e) {  e.printStackTrace();  }  }  public static void main(String[] args) throws Exception{  new BasicRobotUsageNaive().doJob();  }  } |

## Proposal for an application

**User story**: **As** a *user*, **I want** to (**R1**) move a *robot* according to a prefixed route. While the robot is moving, **I want** that (**R2**) the *robot* takes a photo of the world in front of it, every PT (e.g PT=3) seconds. After each shoot, the photo (**R3**) must be scanned in the attempt to recognize a conventional *marker*. If the marker is found, **I want** that the *robot* (**R4**) stops, by sending (**R5**) the photo to me, (**R6**) tagged with the current date and time.

## Proposal for a product backlog

* 1. Build a simple **framework** to make a *robot* able to take a photo with reference to some multi-media technology, with the goal to build a prototype in a short time and to change later (it is the case, and without a big effort) the technology.
  2. 'Inject' the framework into our ddr robot
  3. **Define a DSL** to express synchronous/asynchronous reactive actions to take photos (and videos).

# Working with ddr robots lab 13

### VISION

* A (domain specific) language can be tailored to the main concepts needed in an application domain, by allowing the specification of *executable models*. Such a language should help software developers in writing *executable specifications* during the early stages of software development with particular regard to *requirement and problem analysis*.

### GOALS

Show (by means of concrete examples/applications) how the logical architecture and the project architecture of a (distributed) software **IOT** system including physical robots can be specified by a custom (domain specific) language (**ddr**) and how a proper software factory can reduce the costs and the risks of software development.

# Workflow

Please read

 An introduction to the usage of QActors and QRobots (from pg. 35)To start in using the **base robot**, please consult: Project it.unibo.lab.baseRobot.example

To start in using the **ddr** language, please consult: Project it.unibo.qactor.robot.avatar

# The meta models

* A custom language for robot configuration: [Base.xtext](https://137.204.107.21/syskb/it.unibo.xtext.robot.base/src/it/unibo/xtext/robot/Base.xtext)
* A custom language for QRobots: [Robot.xtext](https://137.204.107.21/syskb/it.unibo.xtext.qactor.robot/src/it/unibo/xtext/qactor/Robot.xtext) based on [Qactor.xtext](https://137.204.107.21/syskb/it.unibo.xtext.qactor/src/it/unibo/xtext/Qactor.xtext)

# The BaseRobot model

* The Facade: [IBasicRobot.java](https://137.204.107.21/syskb/it.unibo.lab.baseRobot/src/it/unibo/iot/baseRobot/hlmodel/IBasicRobot.java)
* A useful class: [BasicRobot.java](https://137.204.107.21/syskb/it.unibo.lab.baseRobot/src/it/unibo/iot/baseRobot/hlmodel/BasicRobot.java)

# Sensors and Sensor Data

The current versione implements:

RobotSensorType: Line | Distance | Impact | Color | Magnetometer

|  |  |  |
| --- | --- | --- |
| **SENSOR** | **Data representation in Prolog** | **Data representation Json** |
| COLOR | color(255 255 255, front) | {"p":"f","t":"c","d":{"color":{"r":255,"b":255,"g":255}},"tm":148...} |
| DISTANCE | distance(43,forward, front) | {"p":"f","t":"d","d":{"cm":43},"tm":14...} |
| IMPACT | impact(touch/loss, front) | {"p":"f","t":"i","d":{"detection":"touch"},"tm":14...} |
| LINE | line(lineLeft/lineDetected, bottom) | {"p":"b","t":"l","d":{"detection":"lineDetected"},"tm":14...} |
| MAGNETOMETER | magnetometer(x(50),y(100),z(0), front) | {"p":"f","t":"m","d":{"raw3axes":{"x":50,"y":100,"z":0}},"tm":14...} |

## Position of a sensor

Each sensor is associated to a position that can assume one of the following values:

DONTCARE|

FRONT | RIGHT | LEFT | BACK | TOP | BOTTOM |

FRONT\_RIGHT | FRONT\_LEFT | BACK\_RIGHT | BACK\_LEFT |

TOP\_RIGHT | TOP\_LEFT | BOTTOM\_RIGHT | BOTTOM\_LEFT |

FRONT\_TOP | BACK\_TOP | FRONT\_TOP\_LEFT | FRONT\_TOP\_RIGHT |

FRONT\_RIGHT\_TOP | FRONT\_LEFT\_TOP | BACK\_RIGHT\_TOP | BACK\_LEFT\_TOP

# Experiments

The project [it.unibo.lab.baseRobot.example](https://137.204.107.21/syskb/it.unibo.lab.baseRobot.example" \t "qa) includes:

* An example of a Java program that calls a program written in C to handle the sonar HC-SR04 UltraSonic Distance Measure Module Range Sensor: SonarMain.java SonarAlone.c
* An example of usage of the **BaseRobot** starting from the façade IBasicRobot: BasicRobotUsageNaive.java

# Radar 2016 lab 14

### VISION

The field of IOT (Internet of Things)...

Una applicazione (IOT) dalla analisi al deployment: [Lab14) Radar](https://137.204.107.21/syskb/it.unibo.iss2015intro/docs/Lab/Lab2016/radar2016.html) Espressioni con sensori: dall'espressione a un sistema QActor basato su sensori HC-SR04 (UltraSonic Distance Measure Module Range Sensor). Introduzione a un generatore di codice che utilizza i sonar installati su un singolo QRobot: [GenFirstQa](https://137.204.107.21/syskb/it.unibo.xtext.neon.Exp/src/it/unibo/xtext/mygenerator/GenFirstQa.xtend).

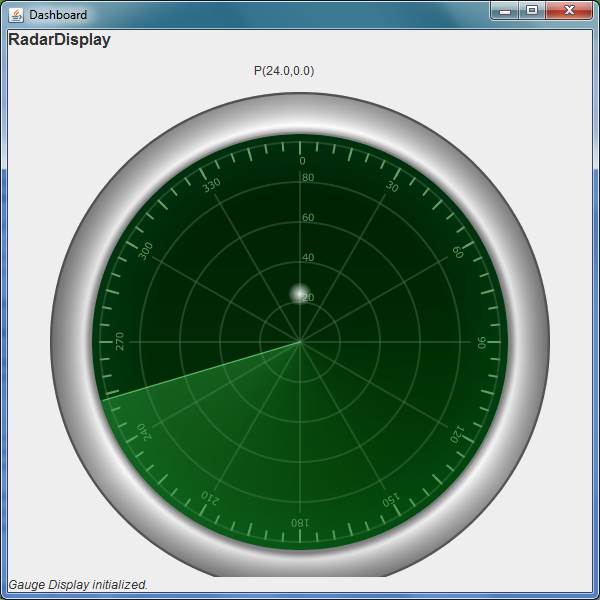
### GOALS

Show how ...

# Requirements

**User story**: **As** a *operator*, **I want** to activate (**R1**) a **RADAR DEVICE** on a RaspberryPi(3) node, and (**R2**) a **RADAR GUI** on a conventional PC **so that**:

* the **RADAR DEVICE** (**R3**) detects obstacles, by (**R4**) exploring the space in front of it, in a range from **0** to **180** degrees.
* the **RADAR DEVICE** (**R5**) sends information of **p(D,A)** (where **D** (0<=D<=80) is a distance and **A** is an angle (0<=A<=180) to the **RADAR GUI**
* the **RADAR GUI** (**R6**) shows the obstacle on a screen like that shown in the picture hereunder and (**R7**) plays a short sound when the distance of the obstacle is lessa than a prexided value.



# Logic architecture

After The system:

* **structure**: is composed of two actors: a **radaguilogic** and an **radardevicelogic**
* **interaction**:
  + The **radardevicelogic** emits *events* of the form **obstacle:p(Distance,Angle)**.
  + The **radaguilogic** expects a *dispatch* of the form **polar:p(Distance,Angle)**.
* **behavior**: the **radardevicelogic** behaves like a source of events, while the **radaguilogic** works as a *message-based* actor that waits for a dispatch **polar:p(Distance,Angle)** and than display a point on the GUI screen.

The problem analysis states that the *radardevicelogic* and the *radaguilogic* *cannot interact in an immediate way*. However, the task to transform an event emitted by the *radardevicelogic* into a message required by the *radaguilogic* can be performed by an **EventHandler** defined in the context of the *radaguilogic*.

## The analysis model

Our first logic prototype defines a single working context and provides some predefined data to facilitate the testing.  **radarSystemLogic.qa**

# Project architecture

## The RADARGUI

The **radagui**cis defined as a standalone system that can be launched on a PC without the need to wait for the data sources.   
The application designer must define the operation *sendEDataToGui(int, int)* to display the data sent from the *radardevice* and from any other data source: **radargui.qa**   
The GUI interface is implemented by an expert application dsigner:

 RadarControl.java

 Radargui.java

## The RADAR DEVICE

The *radar device* is a custom device whose hardware include the components reported in the picture below:

|  |  |
| --- | --- |
| **The physical system** | **The hardware components** |
|  | * A [sonar HC-SR04](https://137.204.107.21/syskb/it.unibo.iss2015intro/docs/Devices/HC-SR04.html) * A [stepper motor 28BYJ-48](https://137.204.107.21/syskb/it.unibo.iss2015intro/docs/Devices/28BYJ-48StepperMotor.html) (alternatively, a [Micro Servo SS099](https://137.204.107.21/syskb/it.unibo.iss2015intro/docs/Devices/SS099.html)) * A [Motor Driver Board L298N](https://137.204.107.21/syskb/it.unibo.iss2015intro/docs/Devices/LN98N.html) * A [RaspberryPi](https://137.204.107.21/syskb/it.unibo.iss2015intro/docs/Raspberry/raspberryEntry.html) |

In particular, the sonar HC-SR04 is put on the top of the stepper motor 28BYJ-48. From the software point of view: 

|  |  |  |
| --- | --- | --- |
| **Component** | **Software** | **Behavior** |
| sonar HC-SR04 | [SonarAlone.c](https://137.204.107.21/syskb/it.unibo.qactor.radar.sensors/src/it/unibo/extra/SonarAlone.c) | Activates the sonar and *writes an***int***(distance) on the standard output*   Application logic: [Qasonar.java](https://137.204.107.21/syskb/it.unibo.qactor.radar.sensors/src/it/unibo/qasonar/Qasonar.java) |
| stepper motor 28BYJ-48 | [stepper.py](https://137.204.107.21/syskb/it.unibo.qactor.radar.sensors/src/it/unibo/extra/stepper.py) [sysKb.py](https://137.204.107.21/syskb/it.unibo.qactor.radar.sensors/src/it/unibo/extra/sysKb.py) | Moves the stepper from 0 to 180 and sends *on the port of the context radardevice, via TCP*the message:  msg(pos,dispatch,python,qastepper,p(VALUE),NUM)    Application logic: [Qastepper.java](https://137.204.107.21/syskb/it.unibo.qactor.radar.sensors/src/it/unibo/qastepper/Qastepper.java) |

With respect to the initial prototype:

* The empty plan *exploreTheSpace* of **radardevicelogic** becomes now a message-receiving activiy that waits for a distance or for a directtion and stores their values within the actor
* The plan *emitTheData*of **radardevicelogic** becomes now an activity that performs *data fusion* and emits an event of form:

obstacle : p(D,A)

* A new actor named **qastepper** is added in order to acquire data from the stepper and forward to the **radardevice** actor a message of the form

pos : p( Angle )

* A new actor named **qasonar**is added in order to acquire data from the sonar and forward to the **radardevice** actor a message of the form

distance : d( Distance )  
**radarDevice.qa**

## TESTING

Ee can add to the system other data sources (without the need to use any placeholder for the **RADARGUI**). For example: **radarOtherSensors.qa**