EmotionBot: Architecture Description

26 de junio de 2014

Índice general

1. General View **3**
- 1.0.1. Simple Action . . . 3
- 1.0.2. Abstract Actions . . . 5
- 1.1. Description . . . 5
  - 1.1.1. Simple Actions . . . 6
  - 1.1.2. Abstract Action . . . 6
  - 1.1.3. Action Profile . . . 6
  - 1.1.4. Emotion Profile . . . 6
  - 1.1.5. Script descriptor . . . 6
  - 1.1.6. Character descriptor . . . 6
- 1.2. Action Modulation . . . 6
  - 1.2.1. Action Modulation . . . 7
  - 1.2.2. Action Generation . . . 8
- 1.3. Belief . . . 9
  - 1.3.1. Relational Social Model . . . 9
  - 1.3.2. Social World Model . . . 9
  - 1.3.3. Emotional Model . . . 10
  - 1.3.4. World Model . . . 10
- 1.4. Implementation . . . 10
  - 1.4.1. Action Modulation . . . 10

2. Requirements **11**

3. Description Design **13**

4. Action Modulation Design **14**

Version	Date	Author	Comments
0.1	27/02/2014	Julian Angel	First draft of the system's design
0.2	23/05/2014	Julian Angel	Changes on the description design and addition on how works the decomposition of abstract action to simple action

Capítulo 1

General View

The architecture presented in this chapter was created following the Belief, Desires and Intentions (BDI) approach, as could be seen in the Figure 1.1 !!WRITE WHY IT WAS CHOSEN THIS APPROACH. Additionally to the modules BDI, it was added the modules of action modulation and description. The description module is used to create information that it could be used by the robot as: the script, normal speed, etc. Each of the modules are decomposed in subsystem that working as a whole could be used to accomplish the task. The final result could be seen in the Figure 1.2, where the module Action was divided in two: Action decision and Action modulation. Also was added the sub-module feature. Each of the modules in the architecture will be described in more detail though this chapter. The whole action system is based on two main concepts: simple actions and abstract actions.

1.0.1. Simple Action

The simple actions are actions that have specialized modules that execute them. In order words they are already implemented and are platform dependent, thus it should be a different implementation of the action for the each platform that should be supported. In the first implementation, simple action should be follow the next characteristics:

- Simple action controllers are not aware about other simple action controller. The synchronization is done through other module.
- Actions that are using the same actuators should be additive. For example the action move and oscillatemove are additive because oscillatemove is just going to introduce oscillation given an angle *theta*, thus the *theta* could be modified by move. These actions two actions are controlled by the same controller, but its interface is different.

At the moment the current actions are:

- $Move(finalposition, trajectoryDescription)$ where finalPosition is the vector $< x, y, \theta >$ and velocity is a vector $< velocity\_x, velocity\_y, velocity\_\theta >$. The controller for this action should move the robot from its current position to the desire position. It is important to notice that the controller must also have as an input the world model in order to avoid obstacles.

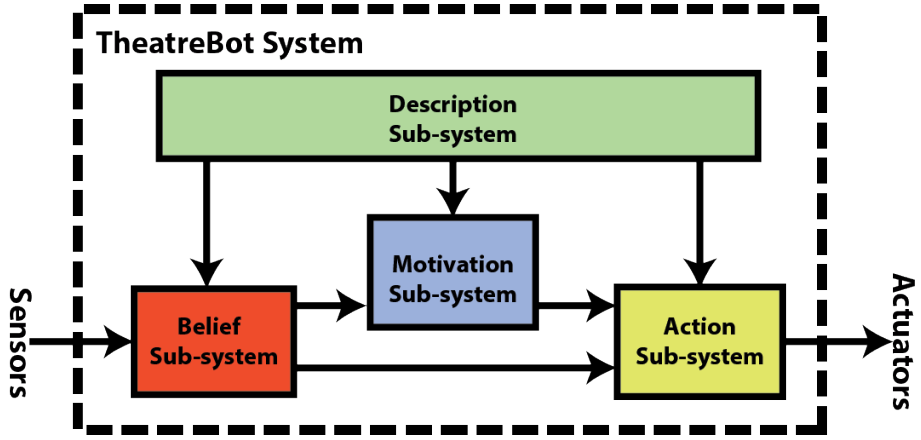

Figura 1.1: General Architecture. The dash line shows the components that are in the system. The concepts of Desires and Intentions are included in the box Motivation.

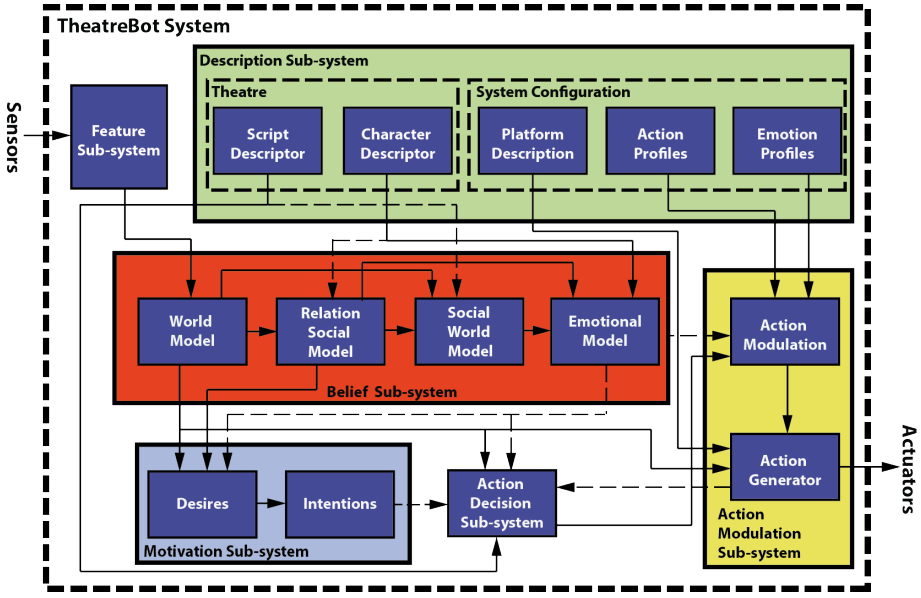

Figura 1.2: Architecture with the subsystems.

The trajectoryDescription is optional, thus if there is not given any information on this, the default values are used.

- $OscillateMove(velocity\theta, maximumAngle, trajectoryDescription)$ the oscillation movement just implies movement in the angular velocity, thus it is just change in the θ component, and $maximumAngel \in [\pi, -\pi)$
- $RotateTorso(angle, trajectoryDescription)$ moves the upper part of the waist, if there is one.
- $OscillateTorso(velocity, maximumAngle)$ which enable the oscillation of the upper part of the waist.
- $Grasp(object)$ try to grasp the desire object
- $RotateHead(angle\theta, angle\beta, angle\omega, velocity\theta, velocity\beta, velocity\omega)$
- $OscillateHead(angelID, velocity, angle)$ where $angleID \in \{angle\theta, angle\beta, angle\omega\}$
- $RorateShoulder(IDShoulder, angle\theta, angle\beta, velocity\theta, velocity\beta)$
- $OscillateShoulder(IDShoulder, angleID, velocity, maximumAngle)$ where $IDShoulder \in (\forall x | x = shoulder \bigwedge x \in robot)$
- $Speak(text)$ where $text$ is the argument that should be said by the robot

1.0.2. Abstract Actions

Abstract actions are actions that could be composed by other abstract or simple actions. For the composition of these kind of actions could be used any kind of synchronization constrains and structure. For example, one abstract action could be composed by other two actions that are executed concurrently or in serially. Right now, the time constrains are not considered.
The abstract action $walk_speak$ which has three parameters: $position$, $trajectory_description$ and $phrase$. This actions is composed by two actions: the abstract action $walk$ and the simple action $speak$. Note that the abstract action $walk$ is also composed by two simple actions: $move_body$ and $move_shoulders$.
The parameters of the abstract actions are explicit attached to the parameters of the actions that composed it. However, in not all the cases the parameters are covered by the abstract actions, thus it should be given an explicit value during the implementation and give a modification through the description system.

1.1. Description

The action description defines all the necessary information that are required to generate emotive action and personalize the system. This system is divided into parts: general, which described the information to generate the emotive actions. And Theatre, which describes the script and the current character that should be portrayed by the robot.

1.1.1. Simple Actions

The description saved for the simple actions are values by default for parameters of each action. If there is not any value for a parameters, it would be used as NULL. These parameters are override each time that the desire action has an explicit value for the parameter.

1.1.2. Abstract Action

The abstract actions' description gives default values

1.1.3. Action Profile

The action profiles describes the simple abstract actions that people have created based on the simple actions that are available in the system. It does not include any preconditions

1.1.4. Emotion Profile

Emotions profiles are based on the research done during the first year, where we come up with some basic features that enable the emotion projection. Each abstract action has is own emotional profile and it should be described in terms of simple actions. Example:

```
<emotion id="happiness">
    <simpleAction id="Move">
<parameter id="velocity">fast</parameter>
    </simpleAction>
    <simpleAction id="OscillateMove">
        <parameter id="velocity">fast</parameter>
        <parameter id="maximumAngel">small</parameter>
    </simpleAction>
    <simpleAction id="BalanceArms">
        <parameter id="maximumAngle">medium</parameter>
    </simpleAction>
    <simpleAction id="OscillateTorso">
        <parameter id="velocity">-fast</parameter>
        <parameter id="maximumAngel">small</parameter>
    </simpleAction>
</emotion>
```

1.1.5. Script descriptor

1.1.6. Character descriptor

1.2. Action Modulation

The principal role of this sub-system is to convert abstract actions (e.g. walk, gaze, etc.) and the emotional state to actions that could be performed by the robotic platform that it has been using. These emotions actions are just commands to the platform, but the emotional part is added during the process

due to adding new simple actions, changing the parameters of the simple actions, or both. This could be done thanks that each abstract action should be describe in terms of simple actions, which the system knows how to modify and perform. The main inputs of these module are:

- Emotion profiles, which is the information in how to show an emotion in terms of simple actions.
- Action profiles, which describes the basic actions and the composition of the abstract actions in terms of the simple actions. This is not longer truth because getting the information from the file implies that should be possible to generate the code from these files.
- Emotional state is described by the tuple emotion and intensity. The first says the emotions that is going to convey the robot, and the second says how much is going to be evident the emotion. Still missing the part of the intensity.
- Abstract action, which could be a simple action to a any action described in terms of the simple actions supported by the system.
- Platform descriptions gives the information about which simple actions could be performed by the current platform.
- World model.

The outputs of this module are:

- *Electronic signals:* to control each driver that has the robot.
- *Failure signal:* to inform the action decision sub-system that one of the actions could not be performed.

This sub-system is sub-divided into two blocks: actions modulation and action generation.

1.2.1. Action Modulation

This module gets the emotional state and the action(s) to generate emotive actions, which includes adding new actions and modifying parameters of the current. To do this module add new actions regarding the platform. The inputs to this module are:

- Emotion profiles
- Action profiles
- Emotional state
- Abstract action

The outputs:

- *Emotional actions:* are the set of simple actions with modified parameters to show emotions.

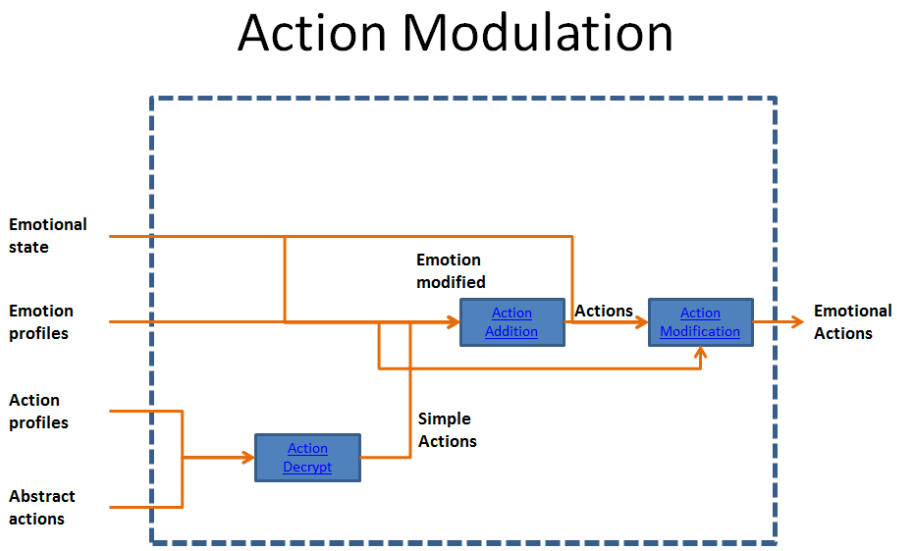

Figura 1.3: Modules of the action modulation sub-system

As well this module is divided in modules with specific objectives, the division could be seen in Figure 1.3. The goal of each module is described below:

- *Action decrypt:* take the abstract actions and converted to simple actions.
- *Action addition:* add the new actions needed to show the emotion.
- *Action modification:* changes the parameters of all the actions.

1.2.2. Action Generation

This module is in charge to decide which actions could be performed by the current platform, and execute and control each action that should be performed. The inputs of this module are:

- Platform description
- Emotional actions
- World model

The outputs are:

- Failure signal
- Signal controls

The task decomposition could be seen in Figure 1.4. The goal of each sub-module are described below:

- *Action filter:* filters the possible actions that could be perform due the constrains of the platform, and it makes any necessary change to the action to be correct performed by the platform (e.g. Constrains of movement)

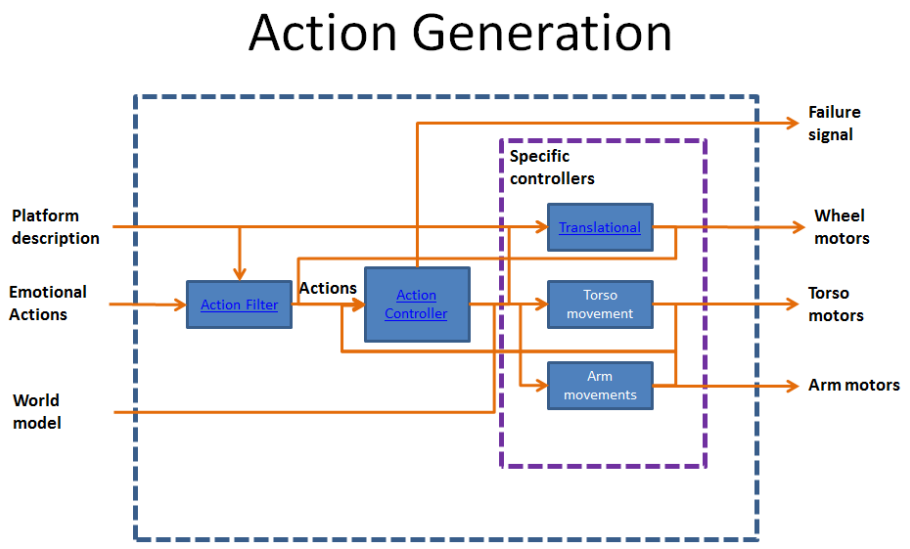

Figura 1.4: Modules of the action generation

- *Action controller:* decides which actions should be executed given the current actions that have performed by the robot. This module communicates with a specific module, which knows how to perform the action. This module knows which actions are executing and solve the problems of precedence and importance.
- *Specific controllers:* are modules developed with the idea to execute an specific simple action.

1.3. Belief

1.3.1. Relational Social Model

This module upgrades the relations with other people and roles, and the emotions that these produce to the character. The inputs of this module are:

- World model
- Character descriptor, which gives the concrete information of the relation during a play. This allow focus in the other modules and test the system.

The output of this module is:

- A list of people and roles that are in the current situation, and the emotions that these produce in the character.

1.3.2. Social World Model

This module is in charge to recognize the current social situation and determine other's emotional state. The inputs of this module are:

- World model
- A list of people and roles that are in the current situation, which is used as a filter.
- Script descriptor, which gives information of the current situation and the emotional state of the other people present in the current scene.

The outputs are:

- Social situation
- List of the people present in the environment and their respective emotional state

1.3.3. Emotional Model

This module calculates the current emotional state based on the people present, their emotional state, and the current situation.

1.3.4. World Model

Capítulo 2

Requirements

- How to change the emotional state without changing the current action?
- How to change the action without the emotional state?
- How to handle when the emotional state changes and the action is the same?
- How to describe the script?
  - How to handle the questions written above?
  - Is it necessary to introduce time?
  - How to add the coordination points?
    - Levels of these points?
- Which is the best way to describe the abstract actions, emotions profiles, etc.? Possibilities:
  - JSON
  - XML
  - YAML

  Comparison:
  - JSON vs XML
  - JSON vs XML
  - YAML
- How to implement the specific controllers in ROS?
  - the controllers depend on the platform?

Capítulo 3

Description Design

Capítulo 4

Action Modulation Design

Capítulo 5

Implementation

5.1. Implementation

The implementation of the system was done in C++ and using ROS, missing reference and more info.

5.1.1. Action Modulation

To model the abstract actions it was used the pattern composite, where the leaf are the simple actions and the composite class are abstract actions. Each abstract should be implemented and it should be specify which action and how the parameters are related in the action, and in further the parameters and the connection could be read from a file. For example the action walk is composed by two simple actions: TODO The name given to each actions are:

- speak
- oscillate_shoulder
- oscillate_body
- move_shoulder
- move_body

These names should be used when it is wanted to execute an action, if a different name is used the system will not execute any action.

The synchronization of the emotion action is done through the topic. This should be done to tell the other actions when they have to change their emotive parameters. If this is not done, it should be add more information to each action. To implement this, it should be tell which is the principal action that coordinates the rest of the actions.