

# Personality Model for a Companion AIBO

Project Report



Author: **Iulia Dobai**  
Date: 25 July 2005

Project Supervisors:  
Charles van der Mast  
Leon Rothkrantz

*“It is better to know some of the questions than all of the answers.”*

**James Thurber**

## Abstract

In this report the author describes a complex set of models and architectures that allow the embodiment of an emotionally intelligent robot in interacting with other humans or robots. The robot used for deploying these ideas and concepts is AIBO (a quadruped autonomous dog-like robot) made available by Sony.

The work that has been done has the somewhat difficult mission to propose a new cognitive model that allows reasoning with emotions for robots and in the same time to offer a prototype implementation.

This report has two focus-points: one on the personality model that has been developed in order to execute emotional reasoning and the second focus-point on the software that renders valuable the model by including it in a running prototype.

Taking into consideration the realities concerning an emotionally intelligent AIBO, that acts in an unpredictable and changing environment, the existing models of personality need improvement, modifications and adaptations to the current situation. Most existing personality models that are used in virtual humans and agents take into consideration three layers: *personality*, *mood* and *emotions* or just two: *personality* and *emotions*. We, on the other hand will try to add a new set of parameters: *needs* that will not constitute a new layer but together with the personality layer, a pillar for the top two layers of *mood* and *emotions*.

From a software point of view the entire system has been developed using Java, Jess and URBI over a wireless TCP/IP architecture with the server resident on AIBO and the client (that incorporates the cognitive component) running on a computer. The artificial intelligence technique used to apply the reasoning component in practice is made up of two highly coupled rule-based systems running in parallel. While the complex ideas of robot - *learning*, *memory* and *instinct* have not been tackled by this project they are not excluded by the existing model and architecture and are considered subjects for further investigation.

**Keywords:** AIBO, robot, human-robot interaction, human-AIBO interaction, JESS, URBI, emotions, reasoning.

## **Preface**

This report is in order to conclude my 9 month assignment in the Man Machine Interaction group of the faculty Electrical Engineering, Mathematics and Computer Science at the **Delft University of Technology** in The Netherlands. This project was done as part of the AIBO Team in the MMI group that is conducting a series of AIBO related projects.

The work that I have conducted at TU Delft was possible due to a **Huygens Grant** that I received from **Nuffic** (Nuffic is the Netherlands organization for international cooperation in higher education). I am deeply grateful to the people in the Huygens Program for awarding the grant to me and therefore opening in front of me a world of opportunities and challenges.

## Acknowledgement

First of all I would like to thank my supervisors **Leon Rothkrantz** and **Charles van der Mast** for inspiring me and offering guidance in the crucial moments of the project when I most needed it. I deeply appreciate the way Mr. Rothkrantz always offered me new perspectives, different approaches or simply lead me to the right track. Our meetings were an incubator of innovative ideas that became solutions for my work.

I will always be grateful for Mr. van der Mast's support and inputs regarding my work and achievements. He always brought a different light in the work being done and helped shape up the project in the way it is now. I deeply appreciate that he believed in me and he issued the invitation to come here and gave me the opportunity to work on such a challenging project.

Second of all I would like to show my great appreciation to the members of the **AIBO Team** in the MMI group at TU Delft. Our weekly meetings prevented me a lot of times from slipping out of the correct track as well as provided me with a lot of valuable information. Special thanks to: Zhenke Yang, Siska Fitrianie, Stijn Oomes and Dragos Datcu, they offered advice, inspiration, guidance and very valuable solutions.

Third but not last I would like to show my appreciation to my **family and friends** most of who are in Romania and despite the distance they continued to encourage and support me.

Last but not least I would like to thank **Nuffic** for offering me a **Huygens Grant** that made this experience possible, an experience that I will never forget and which definitely represents the starting point in my professional career.

## Table of Contents:

Abstract.....	3
Keywords:.....	3
Preface.....	4
Acknowledgement .....	5
Table of Contents:.....	6
Table of Figures .....	8
1 Introduction.....	9
1.1 Project Motivation .....	10
1.2 Project Overview .....	11
1.3 Problem Definition.....	12
1.4 Extended Example .....	12
1.4.1 Context description:.....	12
1.4.2 Description of the interaction.....	13
1.4.3 The power of a picture .....	13
2 Background and Theory.....	15
2.1 What or who is AIBO? .....	16
2.2 AIBO – Sony’s Entertainment Robot in brief.....	16
2.3 From Computers to Actors.....	18
2.4 Related Research.....	20
3 Modeling a “brain” .....	21
3.1 Concepts Used .....	22
3.1.1 Personality.....	22
3.1.2 Mood.....	26
3.1.3 Goals, Preferences, Standards.....	26
3.1.4 Needs.....	27
3.1.5 Emotional States .....	28
3.1.6 Emotional Expressions.....	29
3.2 Personality Model .....	29
3.2.1 Transforming concepts into modeled pieces.....	29
3.2.2 Overview of the model.....	30
4 A Software Implementation Perspective .....	33
4.1 Software premises.....	34
4.1.1 Programming AIBO.....	34
4.1.2 Why use URBI? .....	37
4.1.3 Architecture.....	38
4.1.4 Java .....	38
4.1.5 Rule-based systems.....	40
4.1.6 Jess.....	41
4.2 UML Diagrams .....	41
4.2.1 The user model view .....	42
4.2.2 The structural model view .....	44
4.2.3 The behavioral model view.....	50
4.2.4 The implementation model view .....	52
4.2.5 The environment model view .....	55

4.3	Design Patterns .....	56
4.3.1	Singleton Pattern .....	57
4.3.2	Other Patterns.....	58
4.4	Implementation .....	58
4.4.1	Threads and Groups of Threads .....	58
4.4.2	Java Beans and Jess .....	60
4.4.3	AIBOExceptions .....	61
4.4.4	Functionality of the Rule-Based Systems .....	61
4.4.5	URBI Scripts .....	65
5	Evaluation and Tests .....	67
5.1	Results.....	68
5.1.1	Emotions and Actions .....	68
5.1.2	System flexibility .....	69
5.2	Emotion Perception on AIBO.....	69
5.2.1	Test Description .....	69
5.2.2	Test Results .....	70
5.2.3	Conclusions.....	72
5.3	Recommended User Test .....	73
5.3.1	Psychological Validity .....	73
5.3.2	Rule-Base Correctness .....	74
6	Conclusion .....	75
6.1	Conclusions.....	76
6.2	Recommendations.....	77
6.3	Possible applications.....	78
7	Bibliography .....	81
	Appendix A: AIBO from a hardware perspective .....	83
	Appendix B: Implementation.....	85
	Appendix C: expert system files .....	90
	persRules.clp.....	90
	gpsRules.clp.....	100
	Appendix D: RobotOutput Package.....	110
	Appendix E: ACE Paper .....	111
8	INTRODUCTION .....	111
9	The nPME Model.....	112
9.1	Concepts.....	112
9.1.1	Personality.....	112
9.1.2	Needs.....	112
9.1.3	Mood.....	113
9.1.4	Emotional States and Expressions .....	113
9.1.5	Goals, Preferences and Standards .....	114
9.2	Personality model overview.....	114
9.2.1	A schematic approach .....	114
10	DESIGN CONCEPT FOR COMPANION AIBO .....	115
10.1	A modular architecture .....	115
10.1.1	Input translation .....	115
10.1.2	Processing .....	115

10.1.3	Mapping .....	115
10.1.4	Output translation.....	115
10.2	Development perspective.....	115
10.2.1	Implementation .....	116
11	CONCLUSIONS.....	116
12	ACKNOWLEDGMENTS .....	116
13	REFERENCES .....	116

## Table of Figures

Figure 1:	AIBO ERS 7 .....	16
Figure 3:	Personality Traits in the Big Five Model .....	23
Figure 4:	The big five model and correlations.....	26
Figure 5	Maslow Pyramid of Needs .....	28
Figure 6:	Basic Emotional States.....	29
Figure 7:	Emotional Expressions /Emotional States.....	29
Figure 8:	nPME model at moment $t=i$ .....	31
Figure 9:	nPME Model at moment $t=0$ .....	31
Figure 11:	System Architecture .....	38
Figure 12:	Use Case Diagram .....	43
Figure 13:	Class Diagram for Main .....	45
Figure 14:	Class Diagram for package <initialization>.....	46
Figure 15:	Class Diagram for package <eventGenerators>.....	47
Figure 16:	Class Diagram for package <engines>.....	48
Figure 17:	Class Diagram <engines.gps> (not-detailed) .....	49
Figure 18:	Class Diagram for package <engines.events> (not-detailed).....	50
Figure 19:	Sequence Diagram.....	51
Figure 20:	Activity Diagram.....	52
Figure 21:	Component Diagram .....	53
Figure 22:	Programming Units Diagram: Companion AIBO Overview .....	54
Figure 23:	Package Diagram: Reasoning Component.....	55
Figure 24:	Deployment Diagram .....	56
Figure 25:	Emotion Perception User Test Chart.....	71
Figure 26:	Interpretation of "fear" - Chart .....	72
Figure 27:	Interpretation of "disgust" - Chart.....	72
Figure 28:	AIBO Sensors and Actuators - Front View.....	84
Figure 29:	AIBO Sensors and Actuators - Back View .....	84
Figure 30:	RobotOutput class diagram .....	110



# 1 Introduction

*“There are no secrets to success. It is the result of preparation, hard work, and learning from failure.”*

**Colin Powell**




Somebody once said: *“To create an artificial being has been the dream of man since the birth of science.”* This project is nothing more and nothing less than a mere pebble on the way for accomplishing this dream. Unfortunately there’s infinitely more pebbles needed in order for the road to an “artificial being” to be completely paved.

The project presented in this report aims at developing a new cognitive system that models human personality to be implemented and tested on AIBO (a quadruped robot made by Sony). We start with the existing hardware (the robot itself that is complete and operable) and a limited choice of software development platforms. The purpose of developing such a model for AIBO is to have him realistically respond emotionally to external stimuli. At the end of this project we will see AIBO reason about events and show emotions according to his internal structures rather than on what a human factor would like to see. Another sub-goal of the project is to make AIBO act based on his needs and therefore accomplishing the goal of having it act in changeable and unpredictable environments. The framework developed during this can be seen as a one of the first iterations in a long line of many more iterations to come.

The software developed under this project was made in order to prove the concept of the personality model itself and in order to provide a starting point in future development.

The paper is structured in six chapters with the first chapter entirely dedicated to presenting the problem being solved, the second chapter giving relevant background information for the understanding of the system. The third chapter provides the concepts enclosed in the cognitive model developed and the model itself and is completed by the fourth chapter that presents the software perspective on the system. The fifth chapter is reserved for tests that were conducted while the last chapter concludes the work that has been done.

## ***Chapter Overview:***

-  Project Motivation
-  Project Overview
-  Problem Definition

## 1.1 Project Motivation

*“Ability is what you're capable of doing. Motivation determines what you do. Attitude determines how well you do it.”*

**Lou Holtz**

There's no doubt that robots will become part of our future lives much sooner than we expect. Out of the many possible uses for robots entertainment applications are likely to be here sooner than others. (Current technologies are not sufficiently mature yet to solve the reliability problem that “a useful” robot acting in a home environment requires). Entertainment robots still leave us with a big pool of possibilities including remote controlled robots in various ways and also autonomous robots. The creation of autonomous robots for entertainment seemed like the most challenging job for researchers but also proved to be a major door opener for future technologies and research platforms. Sony was among the first companies to provide the world with an autonomous entertainment robot: AIBO. Before anything else Sony focused on its “lifelike appearance” and tried to answer the major question in the field: “how are humans going to interact with this robot?” In order to answer this question we have to start turning the problem on all its faces and more than anything we have to try and answer the question on how else can we take advantage of the interaction with humans than the pure interaction itself. When Sony developed their first entertainment robot they took care of both the hardware and the software aspects. The AIBO developed by Sony was designed to run around the house and bring a little bit of excitement in a home environment. There is sufficient proof that they managed to accomplish this and mostly everyone that interacts with AIBO in any way will fall in love with it. It is the experience of the author that the first impact with AIBO on almost any new user brought about a big smile. You don't have to be a computer professional or electronics engineer and AIBO will still make you curious.

Going further with the interaction with AIBO, scientists started to question whether AIBO and interacting to AIBO can have other benefits also. Therefore a number of scientists and psychologists conducted tests on elderly interacting with AIBO or people with disabilities interacting with AIBO. Masahiro Fujita concluded most of these researches in two elaborate papers: “On Activating Human Communications with Pet-Type Robot AIBO” and “AIBO: Towards the Era of Digital Creatures”. Studies conducted by Masahiro Fujita assess that AIBO seems to be a good partner with users, having a positive effect on their emotional state. Some very recent experiments in the United States and Japan show that AIBO is useful for mental therapy from a medical point of view [12].

This conclusion was enough to trigger our project. If AIBO as it is was developed by Sony with its stimuli-response emotions was able to prove efficient in therapy then we can develop a mental model for AIBO that can provide more complex emotional and action response.

Furthermore according to the same author the current implementation of AIBO software provided by Sony uses behaviors that come from a “manually designed database”. Next steps would be an *open-ended system* or an *ever-evolving system* by which new behaviors are emerged through the interaction with human and environment. While some efforts are being put by others into developing *unknown-word technologies* and *unknown-object*

*learning technologies* we thought about improving the personality model that judges upon the interaction with human or environment.

On the other hand, the researches that conducted experiments with AIBO had no control over the way AIBO will interact with the humans. This was a good reason for us to decide to develop a mental model for AIBO where some aspects of the way AIBO will interact with humans (like traits of personality for example: extraversion) will be subject to change for the researchers. These two observations triggered our two main goals: developing a complex mental model for AIBO to output emotions and actions and make certain aspects of the model adaptable to the context in which AIBO will be used.

A third and very important motivation to run this project is to develop a framework that is flexible and open enough to absorb new components in the future (like learning, motivation, curiosity, etc), a framework that for now can present itself like a testing platform for different research focus-areas.(like complex vision algorithms for face detection, gesture and emotion recognition, etc)

## 1.2 Project Overview

*“The world is moving so fast these days that the man who says it can't be done is generally interrupted by someone doing it.”*

**Harry Emerson Fosdick**

After finding the reasons for conducting this project we started on finding the path to the solution. The difficulty of the project was increased by the fact that the challenges came from a lot of places including: software, hardware, psychology, not enough past experiences and interaction with humans.

The author started the journey by finding out what is behind the name “AIBO” including hardware and software and a lot of the conclusions are to be found in Chapter Two of this report. The findings brought about a few of the limitations that our newly developed systems will have. Therefore the decisions we had to make were:

- Sacrifice the autonomy of the robot in order to develop fast a complex framework. Therefore our system consists of two components: AIBO that receives inputs from humans through his sensors and acts with emotions in real life situations and a PC that does the reasoning behind AIBO’s actions and emotions. There is however huge differences to a remote controlled AIBO application.
- Focus on solely one issue: AIBO portrays emotions of a result of a complex cognitive reasoning. We therefore left out the very important concept of *robot-learning*.

Following the author concentrated on the development of the psychological model that will allow AIBO to reason. Therefore, a lot of study has been conducted on other mental models for game characters or actors that reason and output emotional reactions. We first decided on the concepts that will be included in our model, on the things that our model will bring new to existing models and we finally developed our model that is fully adapted

to AIBO. Most conclusions and decisions regarding our model are presented in Chapter Three of this report as well as explanations regarding the functionality of our system.

Further we tried to decide regarding the place of our cognitive module in the context of the entire system that is functioning. Therefore we designed the software framework that is the basis for our system. Later came the implementation decisions and constraints that lead to our system. A few decisions were also made regarding what will be implemented during the course of this project and what will be left out. As a result the major focus of this project was on the cognitive component leaving uncovered the input component that would do sound and image recognition. Details regarding the system and the implementation are to be found in Chapter Four of this report.

Just a few experiments were conducted to insure the integrity of the system and of the model and the results are presented in Chapter Five of this report.

Chapter Six concludes the work that has been done and presents the authors view regarding possible future approaches of the matter.

### 1.3 Problem Definition

*“A problem is a chance for you to do your best.”*

**Duke Ellington**

Taken into consideration the complexity of the matter assumed by this subject it is understandable that only a few matters were approached in a very serious way. Therefore the problem definition for this project is:

**Design and implement a mental model for AIBO that represents the basis for showing emotions and actions during interaction with humans in some specified contexts. The mental model developed needs to be presented in a software framework that is flexible and open.**

### 1.4 Extended Example

*“The path of precept is long, that of example short and effectual.”*

**Seneca (5 BC - 65 AD)**

In this chapter we will present a self-explanatory example of the expected outcome of this project.

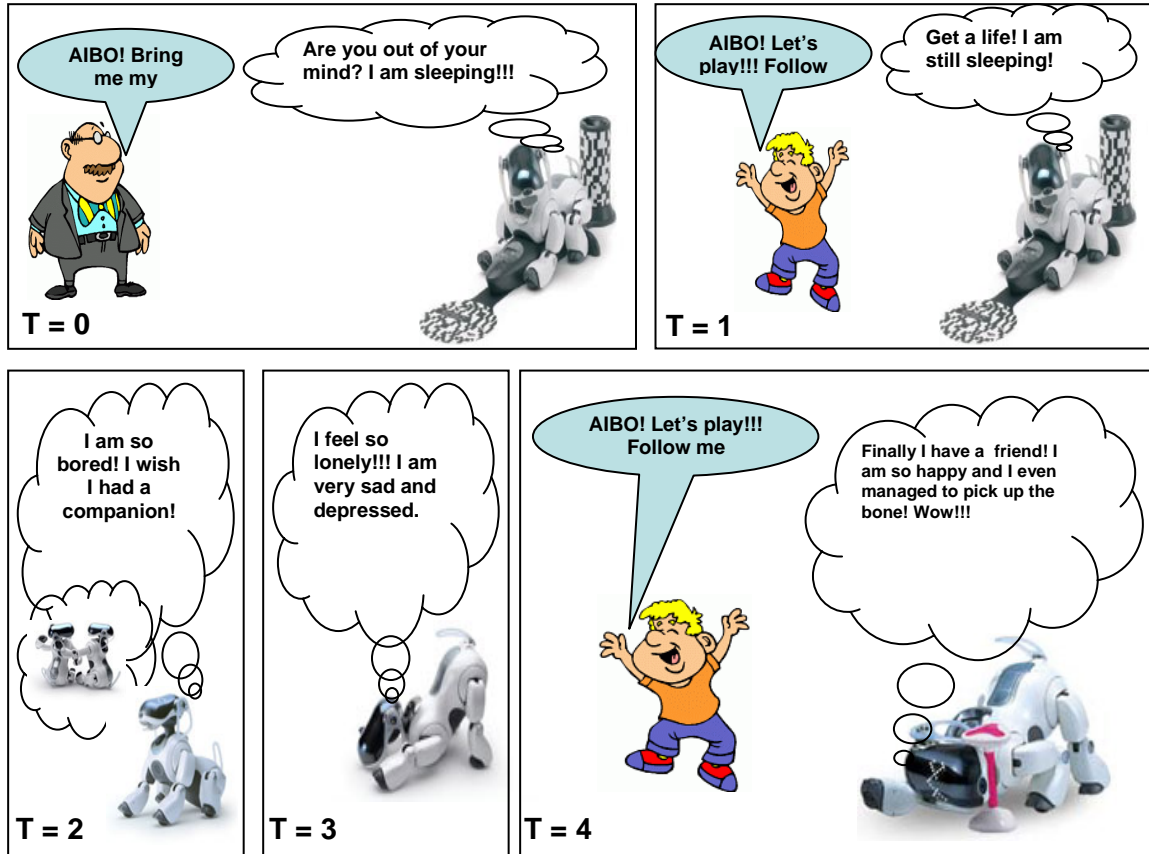
#### 1.4.1 Context description:

AIBO is part of a family of four: Jane and Mathew and their two children Josh (12) and Linda (6). Josh received AIBO as a gift for his birthday and ever since they became very good friends. Occasionally Mathew also likes to play with AIBO.

### 1.4.2 Description of the interaction

AIBO is able to give emotional response while interacting to humans according to his own judgment rather than what humans would expect. Therefore AIBO will be careful to satisfy his needs in turns and show emotions according to this.

### 1.4.3 The power of a picture





## 2 Background and Theory





*“The Past: Our cradle, not our prison; there is danger as well as appeal in its glamour.  
The past is for inspiration, not imitation, for continuation, not repetition.”*

**Israel Zangwill**

In this chapter the author will try to present the various elements that made up the study conducted in order to reach conclusions and solutions in this project. First we will introduce the subject of the investigation: AIBO with its hardware and possible programming environments that are available when we want to create new software for AIBO. Secondly we will investigate the characteristics of AIBO and especially those of the software that makes up the Sony Entertainment Robot: AIBO from the point of view of personality-emotions. Thirdly we will present some conclusions and ideas from the field of virtual humans, game characters, etc generally models of ideas that helped shape up our cognitive model for AIBO.

In order to conclude, the following chapters will present a cognitive model that was put in practice on AIBO. This chapter will present ideas and prerequisite that is required in order to understand and set the context for the following chapters.

### *Chapter Overview:*

-  What or who is AIBO?
-  AIBO – Sony’s Entertainment Robot in brief
-  From Computers to Actors
-  Related Research

## 2.1 What or who is AIBO?

AIBO is Sony at its very best, combining its flagship technologies to conceive a fully autonomous companion to accompany and entertain man in day-to-day life. Sometimes a picture says more than a 1000 words therefore take a look at Figure1.



Figure 1: AIBO ERS 7

AIBO receives information from the outside world through his sensors: video camera (a 350,000 CMOS image sensor), stereo microphones in its ears, two distance sensor, and various touch sensors on head, back, chin, paws, acceleration sensor, and a vibration sensor.

In order to show reactions AIBO is equipped with speaker on its chest, LED Lights (on face, ears and back) and a series of movable parts: head (3 DOF<sup>1</sup>), mouth (1 DOF), legs (4\*3 DOF), ears (2\*1 DOF) and tail (2 DOF). Apart from this AIBO is also equipped with a wireless card and a blue LED to show its status. For more information regarding the physical capabilities of AIBO please refer to Appendix A and the references.[29, 30]

## 2.2 AIBO – Sony’s Entertainment Robot in brief

The centerpiece of AIBO’s artificial intelligence is the AIBO Mind 2 software, located on a removable Memory Stick™. This controls AIBO’s behavior and the applications that you can use via PC or a mobile device. [29, 30]

In day-to-day life, this software enables AIBO to entertain and communicate with you. A privileged companionship will flourish with you thanks to AIBO cleverly recognizing your face and voice. When battery life runs low, AIBO will swiftly locate the energy station to replenish its battery. [29, 30]

With AIBO's skills being located on the Memory Stick, its memory and personality traits are saved to the onboard memory, making your AIBO unique. The AIBO Mind 2 software also offers you the choice of nurturing a fresh AIBO from babyhood to adulthood, or having a mature AIBO, ready fully educated with all AIBO skills. Over time, new skills can be added to AIBO by upgrading the AIBO Mind2Memory Stick. [29, 30]

---

<sup>1</sup> DOF = Degrees of Freedom



AIBO is a great companion and a born entertainer. Throw the pink bone, and it will fetch it for you. The same with the pink ball, AIBO will gladly play with it. AIBO's skill level can improve through encouragement from its owner. [29, 30]

With the Navigator interface you can instruct AIBO wirelessly from your PC to even greater heights of performance. With Navigator remote control, obstacle avoidance and recovery time is greatly enhanced. [29, 30]

AIBO enjoys music and being a natural performer; it will play your favorite music on demand, and even dance for you! AIBO plays internet radio, MP3 or CD. You can have AIBO play a particular CD simply by showing the cover, isn't that particularly clever? [29, 30]

Truly autonomous, AIBO can see, hear, feel for itself and walk. Uniquely skilful, AIBO is able to connect wirelessly with other electronic devices, transmitting photos, sound files and messages. AIBO can even record movie clips. [29, 30]

In House Sitting mode, AIBO takes pictures or records sounds from its Energy Station in response to moving objects, faces, or sounds. When it detects a movement, face or sound, you can set it to snap a picture and notify you by e-mail. [29, 30]

Make AIBO adapt to your daily routine, what's more remind you of it. Thanks to the Scheduler feature you can import calendars and have AIBO read the schedules out. There's no need for a clock radio either as Scheduler ensures that AIBO will play music at your programmed wake-up time. [29, 30]

Through sharing your memories, learning your likes, getting to know your environment, AIBO will become in every way a truly unique individual. Entertaining and comforting you when you're glad, sad or angry. Reflecting a wide range of emotions through its uniquely LED-guided face, AIBO will become, in fact, your best friend. [29, 30]

The question that arises regarding AIBO is what is its cognitive process? Unfortunately Sony seems unlikely to ever disclose the algorithms and models that represent the way AIBO reasons with emotions. Work that was done in our department earlier tried to answer this question experimentally. Results of the experiments conducted were summarized in the paper: "Emotional AIBO" by Chi Hyun Angela Lee. According to our experiments the following were speculated regarding AIBO's cognitive model:

- The user interaction can be classified into three different categories: positive (encouragement), negative (scolding) and neutral (commands).
- Positive emotional behaviours include: green LEDs, some white LEDs, tail wagging, raising a paw to be stroked, dancing, some sound/music playing etc.
- Negative emotional behaviours include: red LEDs, purple LEDs, some white LEDs, head shaking, some sound/music playing etc.
- However, it seems unlikely that these behaviours are the representation of the different intensity of AIBO's emotional states. They are more likely to be random displays.



- Positive and negative interactions with user **always** trigger corresponding emotional responses in AIBO irrespective of its instinctive state (e.g. even when AIBO is hungry and is in search for the charge station, or when it is feeling sleepy and is inactive, it will still react happily when stroked on the back).
- However, these responses and even its emotional state appear to be only **momentary**, that is, AIBO does not seem to have an on-going emotional state. For instance, when AIBO was scolded by being tapped on the third back button, it showed its anger by flashing red LEDs. However, it soon approached the user and raised its paw to be stroked. It's frustration of only few minutes ago seem to have been soon forgotten.
- The emotional behaviours, such as flashing of the green, purple and red LEDs, expressed when there was no interaction with the user, that is, the emotional behaviours linked to the instinctive states of AIBO, seem to be displayed at random or are displayed at such low probabilistic figure that its patterns are difficult to recognise. For example, AIBO would flash green LEDs for no apparent reason even when there is no positive interaction with the user. Further observation may be needed in order to determine the pattern of these behaviours, if there is a pattern. [23]

### 2.3 From Computers to Actors

The social factor of computers was spotted by Fogg awhile ago and thus his interpretation and classification stands proof for how much more needs to be developed in the field of human-computer interaction. B.J. Fogg introduced the “functional triad” [10] that is a conceptual framework that illustrates the different roles that computing technology can play from the perspective of the user. The “functional triad” shows that interactive technologies can operate in three basic ways: as tools, as media and as social actors. One basic function of computers is to serve as tools (this is the first corner of the functional triad). In their role as tools, the goal of computing products is to make activities easier or more efficient to do or to do things that would be virtually impossible without technology. Computers also function as symbolic (when they use symbols to convey information, e.g. text and graphics) and sensory media (when they provide sensory information, e.g. audio). The third corner of the functional triad depicts the role that computers play as social actors or living entities. (e.g. digital pets) [10]

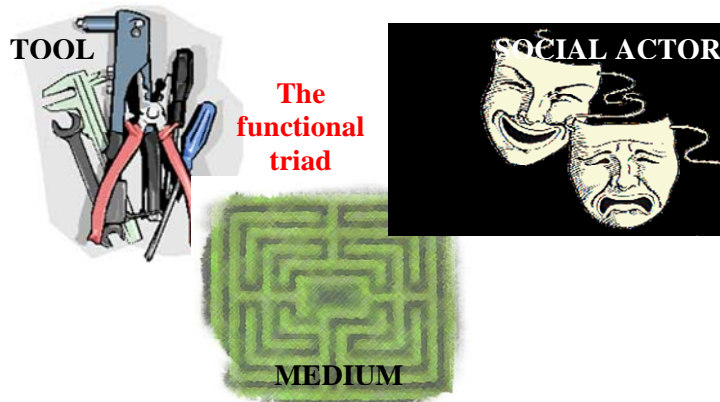


Figure 2: The functional triad

The functional triad [10] :

- Computers as tools – create capability (by making targeted behaviors easier to do, by leading people through a process, performing calculations or measurements)
- Computers as mediums – provide experience (by allowing people to explore cause-and-effect relationships, providing people with vicarious experiences that motivate and by helping people rehearse behavior)
- Computers as social actors – create relationship (by showing and giving feedback, by modeling a behavior or attitude, by providing social support)

Most computing products are a mix of these three functions and practically computing products have evolved from being just tools to blends of tools and mediums or tools and social actors. In our project the computing system that results (a socially responsive AIBO) is a blending of the two corners of **medium and social actor**. By framing AIBO in this category (medium and social actor) we place it on the top layer of computing products giving it the ultimate role in computing systems.

Extensive research has been put into developing real social actors from computers. This work was an inspiration for this project. Virtual actors are used nowadays in chat systems and games. Such an example is the “Conversational Agents” project at MIRALab – University of Geneva. The model proposed by this project is a layered personality-mood-emotion model that can be used in generic characters.

A short overview of a few mental models for believable agents or synthetic actors is presented in [28]. In this paper the authors present the “OZ Project “ that tried to build synthetic actors to exhibit goal-driven behavior in interactive environments. Even though this model uses the same OCEAN Model of personality, in this system personality is seen as a dynamic variable and changes over time. In the “Virtual Theater Project” presented in the same paper, agent behaves like intelligent actors portraying fictions characters. In this model only preferences are used and goals and standards are left apart. In the “GULL Project” of the same paper, the personality is taken much more in serious somehow alike in our nPME model, but preferences and standards are not accounted for. The last model presented by the paper is used in conversational agents and relies on personality as seen in the OCEAN model, on mood. While standards are left out of the model and needs are not taken into consideration the reasoning is being done solely on actions and not on emotions. For the implementation the system uses 3 well coupled expert systems. [28]

Finally the mental model that was developed by this project started from an already existing layered model of personality presented in [20]. The model is used to develop virtual humans which are then integrated into a chat application. The model uses Bayesian Belief Networks for implementation and inspired a lot of the decisions made in the development of the current model including: transforming the five traits of personality in parameters, using a layered approach for the model, including personality, mood and the goals, preferences and standards.

## 2.4 Related Research

Ever since the release of AIBO, the research community oriented activities on the study and development of new applications for the robot.

The first research area that exploded after the release of AIBO was AIBO soccer teams. Currently there are world championships with autonomous AIBOs. Except for the common hardware our system and the software systems for soccer AIBOs have nothing in common.

Later research was focused on testing interaction with humans as explained in previous chapter, and conclusions are encouraging research to develop further and more complex cognitive models. [13]

In the last years research started to concentrate on small problems and address them separately in order to find better answers. Such problems addressed were: curiosity, attention, motivation, vision and sound recognition, etc. Sony research lab in Paris concentrates on learning and motivation, thus their work concentrates on developing a generic reward system that drives an agent to increase the complexity of its behavior. This reward system does not reinforce a predefined task, its purpose is to drive the agent to progress in learning given its embodiment and the environment is which it is placed. [17] A second focus point is concentrated on motivational principles, thus the experiment investigates an AIBO that develops visual competences from scratch driven only by internal motivation. The motivational principles used by the robot are independent of any particular task, as a consequence the developers believe it can constitute the basis for a general approach for sensory-motor development.[18]. A third focus research is on verbal interactions with AIBO. According to this experiment, AIBO learns to interact with humans using real words. AIBO is able to build a vocabulary concerning the objects it perceives visually. [16] Further research was conducted in developing curiosity in AIBO. Therefore the “playground experiment” was started, where AIBO is placed in a child’s playground and focuses on situations which are neither too predictable nor too unpredictable. The developers claim that this mechanism of Intelligent Adaptive Curiosity is the source of autonomous mental development for AIBO.[26]

Other projects investigate different issues regarding two robots like: “joint attention” in [19] where the authors conclude that the key for developing joint attention between two robots is in modeling the mechanism responsible for the emergence of the intentional stance. Another independent project focused on perception while two AIBO’s interact [3].

While these research activities have been conducted outside of our department, in the MMI Department of TU Delft several research areas are exploited including: vision and sound recognition with AIBO’s, fulfilling rescue missions in maze situations by AIBO and performing watch Dog actions. This is the context in which this project was born.

### 3 Modeling a “brain”

*“Emotion, far from being a biological oddity, is actually an integral part of cognition. Reasoning and emotions are not separate: in fact, they cooperate.”*

**Antonio Damasio**



This chapter will present the rationale behind the implementation of the system. The complexity of this project is given by the fact that it does not consist solely on software but it heavily relies on psychological knowledge and requires the modeling of a cognitive model that should resemble that of humans. Due to this we encountered also the first major challenge of the project: the human brain is still an open issue in psychological research. The main problem in designing a model for **human mind** is not the limitation of the machine but the fact that we do not know what the problem is, **we do not know what capabilities humans actually have**.

This chapter will concentrate on these aspects of the project presenting first some concepts and existing models that we used in building up our model like: **personality, mood, standards, goals, emotions, preferences, needs** etc. Later we will introduce the cognitive model for AIBO that we developed - the **nPME model**. The reader will also find here the reasons that made us adopt different types of existing models as well as the way these interact in order to create a working system.

Concepts and ideas like: **values, vision, intuition, curiosity and instinct** have been omitted from the discussion but the existing model does not exclude the possibility of later integrating them. A lot of questions in these matters will remain though unanswered for the time being and require close and careful future examination.

**Learning and memory** are also two issues that are just superficially regarded in this project and should be later researched in detail.

#### *Chapter Overview:*

-  Concepts Used
-  Personality Model

### 3.1 Concepts Used

*“Emotions are the brain's interpretation of reactions to changes in the world.”*

**Antonio Damasio**

Assuming that Damasio is right and changes in the world affect emotions, then all we have to develop here is a system that models the way the brain interprets these changes and reacts to them.

Following we will explain the concepts and models that were taken into consideration in the design of the nPME model for AIBO: personality, mood, needs, goals, preferences and standards, emotional states and expressions.

#### 3.1.1 Personality

The study of personality includes multiple approaches to the question of who we are and how and why we are similar and different to other individuals. Some of the ways in which we study personality are developing descriptive taxonomies of individual differences.

*“Zimbardo defines personality as the psychological qualities that bring continuity to an individual’s behavior in different situations and at different times. It is the thread of continuity in an individual in different situations. Some theories attribute personalities to stable patterns known as traits, types, and temperaments. **Traits are the stable personality characteristics that are presumed to exist within the individual and guide his or her thoughts and actions under various conditions.**” (Zimbardo)*

A trait is a temporally stable, cross-situational individual difference. Currently the most popular approach among psychologists for studying personality traits is the five-factor model or Big Five dimensions of personality.

##### 3.1.1.1 The Big Five

The five-factor model of personality asserts that personality differences can be described by the five independent factors of openness, conscientiousness, extraversion, agreeableness, and neuroticism.

For all the popularity and evident orienting usefulness of the big five, two issues remain problematic. The first concerns whether the five traits are independent of each other. A second and more important issue concerns whether the big five subsume all there is to say about personality. The answer is almost certainly no: Whereas almost any personality construct can be mapped onto the big five, you cannot derive every personality construct from the big five.

Trait	North Pole Name	South Pole Name
neuroticism	negative affectivity, nervousness, anxiety, emotionality	Emotional stability, emotional control
extraversion	energy, enthusiasm, social adaptability, assertiveness, sociability, boldness, self-confidence	Introversion

openness to experiences	Originality, Open-mindedness, Inquiring intellect, imagination, curiosity, independence, cultured	Closed to experience, close-mindedness
agreeableness	altruism, affection, conformity, likeability, friendly, compliance, warmth	Antagonism, coldness, negativity
conscientiousness	control,constraint,dependability,cautiousness, perseverance,superego, strength, prudence	Lack of direction, impulsiveness, carelessness, irresponsibility

**Figure 3: Personality Traits in the Big Five Model**

(Adapted from John (1999), and Zimbardo (2002))

The acronym OCEAN can be used to help recall the letter designations.

### 3.1.1.2 Extraversion

**Extraversion** is marked by pronounced engagement with the external world. Extraverts enjoy being with people, are full of energy, and often experience positive emotions. They tend to be enthusiastic, action-oriented, individuals who are likely to say "Yes!" or "Let's go!" to opportunities for excitement. In groups they like to talk, assert themselves, and draw attention to themselves.

**Introverts** lack the exuberance, energy, and activity levels of extraverts. They tend to be quiet, low-key, deliberate, and disengaged from the social world. Their lack of social involvement should not be interpreted as shyness or depression; the introvert simply needs less stimulation than an extravert does and prefers to be alone. The independence and reserve of the introvert is sometimes mistaken as unfriendliness or arrogance. In reality, an introvert who scores high on the agreeableness dimension will not seek others out but will be quite pleasant when approached.

### 3.1.1.3 Agreeableness

**Agreeableness** reflects individual differences in concern with cooperation and social harmony. Agreeable individuals value getting along with others. They are therefore considerate, friendly, generous, helpful, and willing to compromise their interests with others'. Agreeable people also have an optimistic view of human nature. They believe people are basically honest, decent, and trustworthy.

**Disagreeable** individuals place self-interest above getting along with others. They are generally unconcerned with others' well being, and therefore are unlikely to extend themselves for other people. Sometimes their skepticism about others' motives causes them to be suspicious, unfriendly, and uncooperative. Agreeableness is obviously advantageous for attaining and maintaining popularity. Agreeable people are better liked than disagreeable people. On the other hand, agreeableness is not useful in situations that require tough or absolute objective decisions. Disagreeable people can make excellent scientists, critics, or soldiers.

### 3.1.1.4 Conscientiousness

Conscientiousness concerns the way in which we control, regulate, and direct our impulses. Impulses are not inherently bad; occasionally time constraints require a snap decision, and acting on our first impulse can be an effective response. Also, in times of play rather than work, acting spontaneously and impulsively can be fun. Impulsive individuals can be seen by others as colorful, fun-to-be-with, and zany. Nonetheless, acting on impulse can lead to trouble in a number of ways. Some impulses are antisocial. Uncontrolled antisocial acts not only harm other members of society, but also can result in retribution toward the perpetrator of such impulsive acts. Another problem with impulsive acts is that they often produce immediate rewards but undesirable, long-term consequences. Examples include excessive socializing that leads to being fired from one's job, hurling an insult that causes the breakup of an important relationship, or using pleasure-inducing drugs that eventually destroy one's health.

Impulsive behavior, even when not seriously destructive, diminishes a person's effectiveness in significant ways. Acting impulsively disallows contemplating alternative courses of action, some of which would have been wiser than the impulsive choice. Impulsive behavior also sidetracks people during projects that require organized sequences of steps or stages. Accomplishments of an impulsive person are therefore small, scattered, and inconsistent. A hallmark of intelligence, what potentially separates human beings from earlier life forms, is the ability to think about future consequences before acting on an impulse. Intelligent activity involves contemplation of long-range goals, organizing and planning routes to these goals, and persisting toward one's goals in the face of short-lived impulses to the contrary. The idea that intelligence involves impulse control is nicely captured by the term prudence, an alternative label for the Conscientiousness domain. Prudent means both wise and cautious. Others in fact, perceive persons who score high on the Conscientiousness scale as intelligent. The benefits of high conscientiousness are obvious. Conscientious individuals avoid trouble and achieve high levels of success through purposeful planning and persistence. Others also positively regard them as intelligent and reliable. On the negative side, they can be compulsive perfectionists and workaholics. Furthermore, extremely conscientious individuals might be regarded as stuffy and boring.

**Un-conscientious** people may be criticized for their unreliability, lack of ambition, and failure to stay within the lines, but they will experience many short-lived pleasures and they will never be called stuffy.

### 3.1.1.5 Neuroticism

Freud originally used the term neurosis to describe a condition marked by mental distress, emotional suffering, and an inability to cope effectively with the normal demands of life. He suggested that everyone shows some signs of neurosis, but that we differ in our degree of suffering and our specific symptoms of distress. Today neuroticism refers to the tendency to experience negative feelings. Those who score **high on Neuroticism** may experience primarily one specific negative feeling such as anxiety, anger, or depression, but are likely to experience several of these emotions. People high in neuroticism are emotionally reactive. They respond emotionally to events that would not affect most people, and their reactions tend to be more intense than normal. They are more likely to



interpret ordinary situations as threatening, and minor frustrations as hopelessly difficult. Their negative emotional reactions tend to persist for unusually long periods of time, which means they are often in a bad mood. These problems in emotional regulation can diminish a neurotic's ability to think clearly, make decisions, and cope effectively with stress.

At the other end of the scale, individuals who score **low in neuroticism** are less easily upset and are less emotionally reactive. They tend to be calm, emotionally stable, and free from persistent negative feelings. Freedom from negative feelings does not mean that low scorers experience a lot of positive feelings; frequency of positive emotions is a component of the Extraversion domain.

### 3.1.1.6 Openness to Experience

Openness to Experience describes a dimension of cognitive style that distinguishes imaginative, creative people from down-to-earth, conventional people. Open people are intellectually curious, appreciative of art, and sensitive to beauty. They tend to be, compared to closed people, more aware of their feelings. They tend to think and act in individualistic and nonconforming ways. Intellectuals typically score high on Openness to Experience; consequently, this factor has also been called Culture or Intellect. Nonetheless, Intellect is probably best regarded as one aspect of openness to experience. Scores on Openness to Experience are only modestly related to years of education and scores on standard intelligent tests.

Another characteristic of the open cognitive style is a facility for thinking in symbols and abstractions far removed from concrete experience. Depending on the individual's specific intellectual abilities, this symbolic cognition may take the form of mathematical, logical, or geometric thinking, artistic and metaphorical use of language, music composition or performance, or one of the many visual or performing arts. People with low scores on openness to experience tend to have narrow, common interests. They prefer the plain, straightforward, and obvious over the complex, ambiguous, and subtle. They may regard the arts and sciences with suspicion, regarding these endeavors as abstruse or of no practical use. Closed people prefer familiarity to novelty; they are conservative and resistant to change.

Openness is often presented as healthier or more mature by psychologists, who are often themselves open to experience. However, open and closed styles of thinking are useful in different environments. The intellectual style of the open person may serve a professor well, but research has shown that closed thinking is related to superior job performance in police work, sales, and a number of service occupations.

### 3.1.1.7 How to use the five factor model

**The descriptions of the 5 personality traits together with the table above are not the product of this author and do not present the author's view on the subject but is text adapted from psychology literature and summarized in the paper: “Five factor Constellations and Popular Personality Types” by Leland R. Beaumont (2003).** While the definitions and descriptions presented do not represent the knowledge of the author itself the author strongly believes in the correctness of them and intensively used the knowledge in developing the system and the software. [5

Appendix G – The abridged big five circumplex (AB5C) Model

	Extraversion		Agreeableness		Conscientiousness		Emotional Stability		Intellect	
	I+	I-	II+	II-	III+	III-	IV+	IV-	V+	V-
I+	Talkative Extraverted Aggressive		Merry Cheerful Happy	Rough Abrupt Crude	Alert Ambitious Firm	Reckless Unruly Devil-may-care	Unselfconscious Weariless Indefatigable	High-strung Excitable Meddlesome	Theatrical Worldly Eloquent	Unscrupulous Pompous
I-		Shy Quiet Introverted	Soft-hearted Agreeable obliging	Cold Unfriendly Impersonal	Careful Cautious Punctual	Inefficient Lazy Indecisive	Unexcitable Unassuming	Self-pitying Insecure Fretful	Introspective Meditative Contemplating	Unimaginative Uninquisitive Inarticulate
II+	Sociable Social Enthusiastic	Timid Unaggressive submissive	Sympathetic Kind Warm		Responsible Dependable Reliable		Patient Relaxed Undemanding	Emotional Gullible	Deep Diplomatic Idealistic	Simple Dependent Servile
II-	Dominant Domineering Forceful	Unsociable Uncommunicative Seclusive		Unsympathetic Unkind Harsh	Stem Strict Deliberate	Unreliable Negligent Undependable	Unemotional Masculine	Irritable Temperamental Defensive	Individualistic Eccentric	Shallow Terse
III+	Active Competitive Persistent	Reserved Restrained Serious	Helpful Cooperative Considerate	Hard Rigid	Organized Neat Orderly			Particular	Analytical Perceptive Informative	
III-	Boisterous Mischievous Exhibitionistic	Unenergetic Uncompetitive sluggish		Inconsiderate Rude Impolite		Disorganized Disorderly Careless	Informal	Hypocritical Compulsive Nosy		Shortsighted Unobservant Ignorant
IV+	Confident Bold Assured	Tranquil Sedate Placid	Trustful Pleasant Tolerant	Insensitive Unaffectionate Passionless	Through Steady Consistent		Unenvious		Intellectual Inventive Intelligent	Unreflective Unsophisticated Imperceptive
IV-	Flirtatious Explosive Wordy	Lonely Weak Cowardly	Sentimental Affectionate Sensitive	Demanding Selfish Ill-tempered		Inconsistent Scatterbrained Unstable		Moody Jealous Possessive	Sensual	
V+	Expressive Adventurous Dramatic	Inner-directed	Genial Actful	Shrewd	Industrious Perfectionistic Sophisticated	unconventional	Versatile		Creative Imaginative Philosophical	
V-	Verbose	Passive Meek Dull		Uncharitable Ruthless Coarse	Conventional Traditional	Haphazard Illogical Immature	Imperturbable	Contemptuous		Uncreative Unintellectual Unintelligent

Figure 4: The big five model and correlations

### 3.1.2 Mood

Mood is a conscious and prolonged state of mind that directly controls the emotions. While emotions are instantaneous, mood is constant for longer time spans [9].

I categorized mood into 3 basic categories: good, bad and neutral. The emotions categorized by the OCC model under the negative group are more likely to be expressed when in bad mood. The same strategy applies for positive group. It is possible though that our mood forbids us from being expressive – this is the neutral mood. In this mood AIBO will tend to not change his expression and the expressions will tend to be less intense. The presence of mood in a well established personality model is doubtless, without mood we can not properly link personality which is in a sense “eternal” to emotional expressions that are instantaneous. Unfortunately there isn’t sufficient literature in psychology that would allow a systematic theory of mood. Therefore the mood in our model will only have one interpretation with different values ranging from positive to negative.

The mood model we have used has its fundamentals in the mood model proposed by Lang in 1995 [21] that basically plots mood on a two dimensional system with *valence* and *arousal* as axes. Valence refers to whether the mood is positive or negative and arousal refers to the intensity of the mood.

### 3.1.3 Goals, Preferences, Standards

In order to explain the importance of these three concepts in the model we will give the following example. Let’s imagine AIBO likes bones and his owner gives him a whole bunch of bones. AIBO should evaluate the consequence of this event for itself, and it will result in satisfaction since it received a big bunch of bones. In the same time AIBO should

evaluate the consequence of the event for his owner. (Should result in pity because the owner lost a whole pile of bones. This however can be correlated with the knowledge that the owner doesn't need/likes bones). In order to do this kind of judgment a robot needs knowledge. First it has to know the relationship to the user, and it needs to know what this event means for the user. Going back to strictly the consequences of the event on itself AIBO should have the following knowledge:

- It should have a goal: “staying alive” to which the bones contribute.
- It should know what kind of actions it can expect from the user – standards.
- It should know weather it likes or not bones - preferences

Since I decided to use the OCC Model as a guiding line for the model, including the preferences, standards and goals in the model is a must.

By goals we understand tasks orientated objectives that are SMART (simple, measurable, acceptable, realistic, time frame) (e.g.: AIBO find ball).

By preference we understand appealing ness to aspects of objects (e.g. like/dislike of ball).

By standards we understand approval/disapproval regarding actions of agents. Standards can focus on self agent or other agents (e.g. approval of being touched on back).

The goals, preferences and standards parameters will be set dynamically based on needs and personality. Every time changes have occurred in the parameters of needs the goals, preferences and standards (abbreviated as the GPS system) will also be updated.

### 3.1.4 Needs

It is generally agreed that the simpler emotions, those whose expression and recognition Eckman (1972, 1989) has shown to be universal are driven by the basic need of organisms such as mating, defense or avoidance of predators, and social affiliation [9]. Therefore here is the theoretical framework we were looking for. In order to develop a realistic AIBO companion dog that shows emotions in a changing and unpredictable environment we have to take into consideration his needs. There are a few need theories known in human psychology but one stands up among all: “Maslow Pyramid of Needs”[24] (see Figure 5).

The use of Maslow Pyramid of Needs is not accidental but based on very solid reasons: it seems the one most widely accepted, it has been used successfully in other occasions and in other ways for modeling human behaviors and it is a model simple enough to be implemented and sufficiently complex to show realistic results. The needs model proposed by Maslow is based on a structure (the pyramid), is based on some rules and principles that can easily be translated into implementation and uses priorities that will be in our case a way to support a priority management system in AIBO.

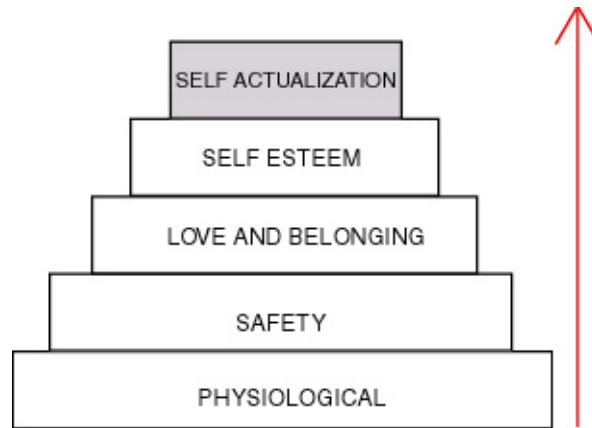


Figure 5 Maslow Pyramid of Needs

The pyramid of needs that Maslow introduced as a base for motivation theory contains 5 categories of needs: physiological, safety, love and belonging, self esteem and self actualization. The first 4 categories of needs have been introduced by Maslow as “deficit needs” while the last category of needs is known as “being needs”. In our *nPME* model for AIBO we will take into consideration only the “deficit needs” since these needs can be met fully while self actualization needs are seen as growing (a continuous driving force). Maslow supports our decision by explaining that not everyone ultimately seeks self-actualization [24]. Maslow's Hierarchy of Needs states that we must satisfy each need in turn, starting with the first, which deals with the most obvious needs for survival itself. Only when the lower order needs of physical and emotional well-being are satisfied we are concerned with the higher order needs of influence and personal development. Conversely, if the things that satisfy our lower order needs are swept away, we are no longer concerned about the maintenance of our higher order needs.

To use the pyramid of needs in AIBO’s architecture we translated the different categories into AIBO specific needs (e.g. physiological needs are represented by the battery level, etc.). Needs have priorities and act as thermometers, once critical values have been reached by different categories of needs depending on their priorities they lead to different structures of goals, preferences and standards. (e.g. once the battery reaches a critical value of 10 % a new goal with the highest priority is being generated that states: recharge, etc.).

### 3.1.5 Emotional States

By emotional state I understand a particular state of mind that is reflected visually by way of an **emotional expression**.

Both emotional state and emotional expression refers to the same thing. I will use the emotional categories proposed by the OCC Model [26]. The model categorizes various emotional states based on positive or negative reactions to events, actions and objects. The OCC model defines 22 such emotions. Due to the complexity of the model only 12 of these emotional states will be taken into consideration in this model.

The emotional states are presented in the Figure6:

<b>Positive</b>	<b>Negative</b>
-----------------	-----------------

Joy	Distress
Pride	Hate
Love	Fear
Hope	Remorse
Relief	Anger
Gratitude	Disappointment

**Figure 6: Basic Emotional States**

Even though the difference between emotional states and expressions it is big enough, and it was taken into consideration initially the two were not implemented as such in the system in order to simplify the prototype.

### 3.1.6 Emotional Expressions

To reduce the computational complexity and because of the limitations of the robot itself I will use only 6 basic emotional expressions to represent the emotional states.

Expression	State
<b>Joy</b>	joy, pride, love, hope, relief, gratitude
<b>Sadness</b>	distress, remorse, disappointment
<b>Anger</b>	anger, hate
<b>Fear</b>	fear

**Figure 7: Emotional Expressions /Emotional States**

In addition to these emotional expressions AIBO will also be able to show **surprise and disgust**. They are not present in the OCC model mainly because they do not involve much cognitive processing and they do not correspond to reactions.[4]

## 3.2 Personality Model

*“The ancestor of every action is a thought.”*  
**Ralph Waldo Emerson**

Below we will introduce the model that unifies these concepts in a mental model for robots and virtual characters.

### 3.2.1 Transforming concepts into modeled pieces

#### Personality – a continuous thread in the life of the application

Conventional wisdom has long set that our personality is genetic and therefore set in stone but according to some scientist those five key personality characteristics change throughout our lives. In the case of robots that don't really have a life span, it is acceptable to have a persistent personality that does not change due to external influences. We based our decision on Paul Costa Jr. whose work revealed that we don't see major personality changes but we see “nuanced” changes [8] that we consider to be insignificant in the case of robots. Therefore in the model we introduce, personality is not changed through time and the values for the 5 dimension of personality are constant through the application.

### **Mood – a slightly changing variable in the life of a character**

Mood is the one variable of the system that is influenced by both changes in personality and by changes in emotions. Therefore mood is constantly changing with little amounts. Special care needs to be taken in order to prevent mood from oscillating between extreme positive and negative values in very short amounts of time.

### **Emotions**

Emotional expressions and states became one single unit in the system that is presented in this paper (even though the model can encapsulate both concepts). The six emotional reactions that AIBO can show are: happy, sad, angry, fear, surprise, disgust.

### **3.2.2 Overview of the model**

Until now in the design of virtual humans, agents or game characters, two categories of personality models were mostly used: PME models that take into consideration personality, mood and emotions and PE models that are based solely on personality and emotions.

In designing our nPME model we started with a layered PME architecture as described in [20]. Here mood is seen as an intermediate layer between personality and emotions and therefore is influenced by both. Since we want AIBO to act independently in a changing environment we have to take into consideration his individual needs, therefore we included in the layered architecture the needs parameter. By adding needs we also had to make some major changes in the existing architecture and therefore mood is not directly influenced anymore by neither emotions or personality, but it is indirectly influenced by them as a consequence of the evaluation of the goals, preferences and standards and the needs factor.

Figure 8 introduces the time based nPME model developed by us that has as ultimate goal the development of a companion AIBO that acts independently in a changing environment. According to this design AIBO shows an emotional reaction and will follow a specific set of actions only if some kind of event triggers the engine that will generate them.

Initial set-up requires values for all traits of personality (that will remain constant) and an initial set of needs. Every time there is a new set of needs present in the system a small engine will be activated to update values for goals, preferences and standards. If an event occurs (by event we understand a registered happening that is detected by AIBO) the response engine is triggered and based on the evaluation of goals, preferences and standards and on the current mood a new emotional state and a set of actions results.

Figure 9 presents the initialization of the entire system at moment  $t=0$ . At this moment we have to take into consideration the possibility that no event happens in a certain amount of time, therefore based on the personality of AIBO and its needs we decide on a few actions randomly chosen from the list of possible actions. It is important to mention that even if no event occurs in a specified amount of time an event will be generated. (e.g. the absence of any human for prolonged amounts of time in the vicinity of an AIBO represents an event that will lead to an increase of the need for love and belonging and a decrease of the positive value of mood with a precise amount.)

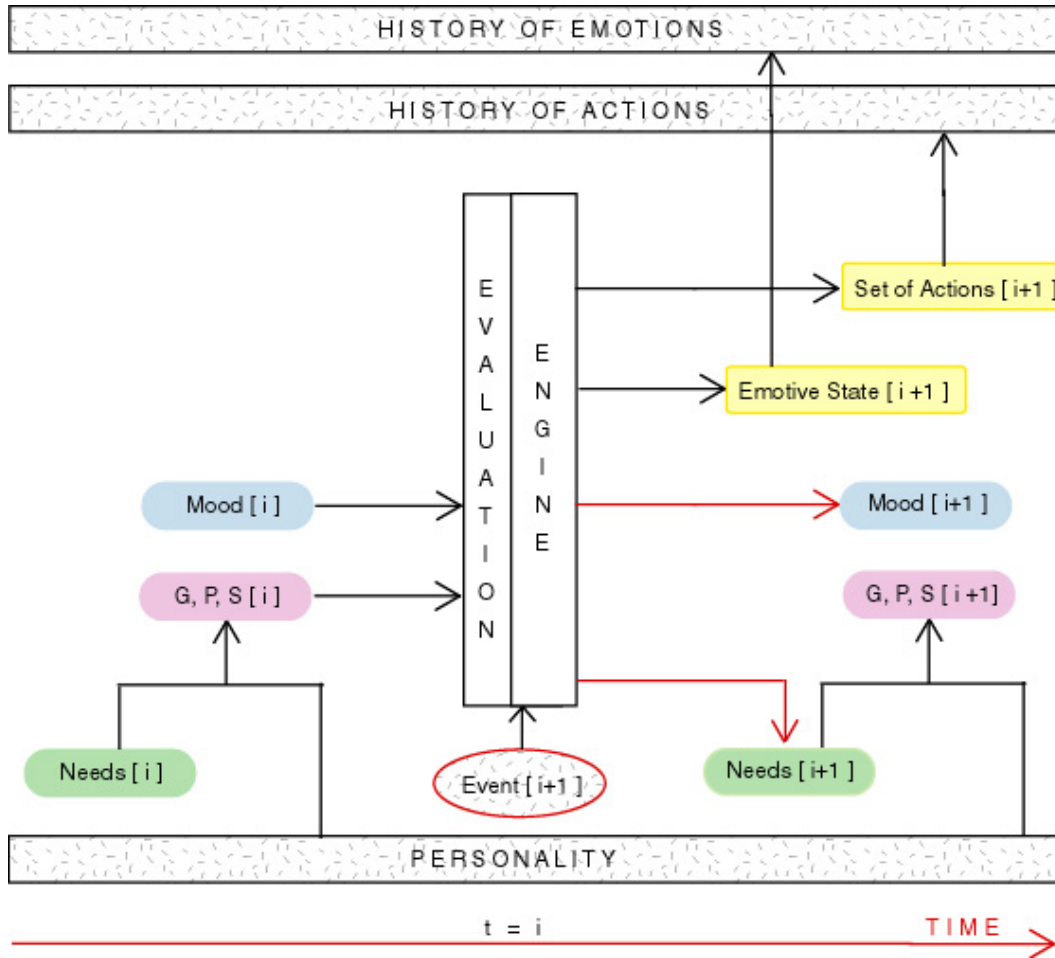
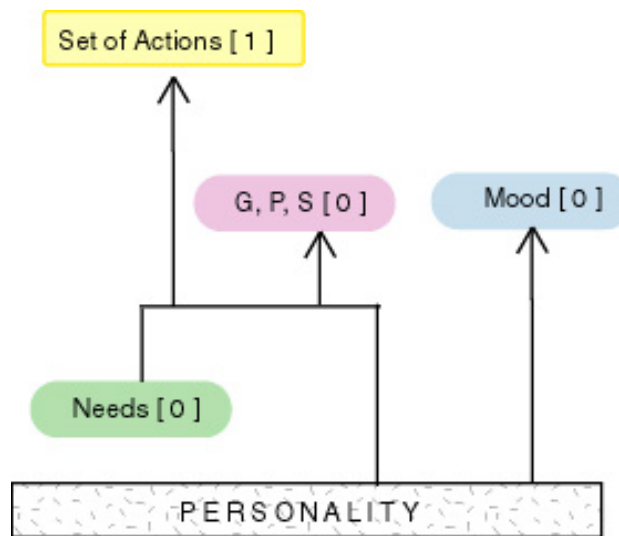


Figure 8:nPME model at moment t=i



Needs [0] - Initial values are set by system user  
 Personality - as a constant variable in the system, is set by user of the system at start-up.

Figure 9: nPME Model at moment t=0





## 4 A Software Implementation Perspective

*“I have made my world and it is a much better world than I ever saw outside.”*

**Louise Nevelson**





More than anything this system is considered by the author a software development process. In order to develop a quality system from the point of view of the programming effort the process was aligned with the Unified Process. (UP)

The UP promotes several best practices, but one stands above the others: **iterative development**. In this approach, development is organized into a series of short, fixed-length (for example, four week) mini-projects called **iterations**; the outcome of each is a tested, integrated, and executable system. Each iteration includes its own requirements analysis, design, implementation, and testing activities. [22]

Although this was not followed thoroughly the approach taken in the development of this system is an **iterative and incremental development**

This chapter explains all the software issues that arise without trying to present the development process neither in the time-line that it took place neither based on the components it is made up of. The system is presented divided on the issues that addressed. Hopefully the reader will find this approach easy to understand.

### *Chapter Overview:*

-  Software premises
-  UML Diagrams
-  Design Patterns
-  Implementation

## 4.1 Software premises

*“The greatest challenge to any thinker is stating the problem in a way that will allow a solution.”*

**Bertrand Russell**

This sub-chapter presents the technical concepts used to develop the prototype for the system. I used a bottom-up approach in deciding what programming tools and languages I should use. Starting with the bottom layer that directly communicates to AIBO I made the decision of using **URBI**. The reasons behind this are explained below. Further by using URBI I was faced with choosing a programming language (in between C, C++ and Java) and an operating system. Choices for development environment were: a Linux machine, a Windows machine using the Linux emulator: *cygwin* and a pure **Windows machine**. I made my choice for the last one and choose **Java** to assure flexibility. By choosing Java it is fairly realistic to later move the application to any other kind of machines including PDA's or anything else. (Attention: choosing the development environment (the place where you do your development) can be different then the environment that you use to test and run your application.) Making the choice for developing environment forced in a way making a choice for the **architecture of the system** therefore it is also described in this section. Further in development a choice had to be made regarding the method used to implement the cognitive component. It was obvious that an **AI** method needed to be used. Due to time constraints the choice was made for using **rule-based systems** (the simplest AI technique). Therefore I made the choice of using **Jess** for the implementation which is an expert system shell for java. A very brief introduction to all of these technologies is presented above. Some of the details regarding them are included in section 4.5.

### 4.1.1 Programming AIBO

Even though a lot of effort has been put into developing programming kits to aid development of different types of applications for the AIBO one can find them scarce and never able to provide everything one might need. Even Sony had put considerable effort in providing tools but as far as the low-level tool is concerned: Open-R programming, debugging and developing was found way too complicated by the author (and rather restrictive then permitting).

More details regarding all of the tools presented above are to be found in the references and the following web sites: <http://www.urbiforge.com/>, <https://openr.aibo.com/openr/eng/>, <http://www-2.cs.cmu.edu/~tekkotsu/> .

#### 4.1.1.1 Sony Development Tools

Although AIBO was created initially as an entertainment robot for the home, it has been embraced by many academics and researchers looking for a low-cost programmable robot platform. AIBO is completely programmable at a variety of different levels and is an excellent platform for research as well as education. Sony has developed a family of different programming kits for AIBO, suitable for a wide variety of applications.

##### 4.1.1.1.1 Remote Framework

The AIBO Remote Framework is a Windows PC application development environment based on Visual C++ with which you can make software that works on a Windows PC. The software can control AIBO (ERS-7) via a wireless LAN. Commercial usage is allowed, and the license fee is free. [see 31]

#### 4.1.1.1.2 R-Code SDK

R-CODE is a high-level scripting language for AIBO, similar to BASIC. R-CODE allows you to very easily create simple programs for AIBO to follow. While it does not allow the detailed control that the OPEN-R SDK has, what it lacks in power it makes up for in simplicity. Sony offers freely-available R-CODE interpreters for all AIBO models. Commercial usage is allowed, and the license fee is free.[see 31]

#### 4.1.1.1.3 OPEN-R SDK

The OPEN-R SDK is a C++ based programming development environment, based on open-source tools (like gcc), that allows you to make software that works on AIBO Entertainment Robots. The kit is considered "low-level" and allows you to control everything from the gain values of AIBO's actuators to retrieving AIBO's camera data and doing computer vision computations. It is an excellent choice for researchers doing low-level robotic research. [see 31]

#### 4.1.1.2 Tekkotsu Framework

**Tekkotsu** is an application framework for robotic platforms. An application framework is code which handles the low level or routine tasks for you, so that you can concentrate on whatever is unique to your application. If you are looking to program your AIBO to do new things, and are comfortable with C++, this may be for you.**Tekkotsu** has been designed with portability in mind, and currently runs on the ERS-210, ERS-220, and ERS-7. **Tekkotsu** is an object oriented and event passing architecture, similar in design to Java. Events and class interfaces make it a snap to add new functionality, because you don't have to edit the framework to add hooks to your code. [see 33]

#### 4.1.1.3 URBI

**URBI** (Universal Robotic Body Interface) is a scripted language designed to work over a **client/server architecture** in order to remotely control a robot or, in a broader definition, any kind of device that has actuators and sensors. The main characteristics of URBI, which make it different from other existing solutions, are:

- ∅ URBI is a **low level** command language. Motors and sensors are directly read and set. Although complex high level commands and functions can be written with URBI, the raw kernel of the system is low level by essence.
- ∅ URBI includes powerful **time oriented** control mechanisms to chain commands, serialize them or build complex motor trajectories.
- ∅ URBI is designed to be **independent** from both the robot and the client system. It relies on TCP/IP or Inter-Process Communication if the client and the server are both running onboard.

- Ø URBI is designed with a constant care for **simplicity**. There is no "philosophy" or "complex architecture" to be familiar with. It is understandable in a few minutes and can be used immediately.

URBI robot-specific server code and libraries are released under the GNU General Public License. The URBI Language, the URBI Kernel and the URBI Language Specification are protected by a separate and specific license. [1]

The main component of the URBI framework is the **URBI Server** which practically is an OPEN-R object running on AIBO and accomplishing some tasks. The URBI Server is a C object compiled with gcc and copied on the memory stick. The server comprises of a **server kernel** that is robot independent and a robot specific add-on. The URBI server receives **commands** from a client and returns **messages** to this client. The normal way of using a URBI controlled robot is to send commands using **TCP/IP** on the URBI port (54000) and wait for messages in return. A simple telnet client is enough to do that for simple applications, otherwise libraries (**liburbi**) are available in most programming languages to wrap the TCP/IP sending/receiving job in simple functions.[see 32]

Using URBI with a telnet is too limited. You need to be able to send commands and receive messages based on tag filter, using a programming language of your choice. This is what **liburbi** is made for. There is currently a **C++** and **java** version of **liburbi** if you want to control your robot using **C++** or **Java** from an independent device like a PC/PDA and a **liburbi-OPENR** version if you want to recompile a **liburbi-C++** based program to let it run on the robot.

To know what you can do with AIBO through URBI, simply check the list of available devices and their fields and methods. This is already enough to control your robot. But URBI is more than that. You can give **complex motor commands** with complex trajectories, you can set several commands in **series** or **in parallel**, you can do some usual C programming using while, for, if. There are also some advanced features for **event catching** (at, whenever). [2]

A schematic approach to the way the entire URBI framework is composed is presented in Figure 10. The scheme is a representation of all alternatives in URBI rather than a concrete functional scheme of the system.

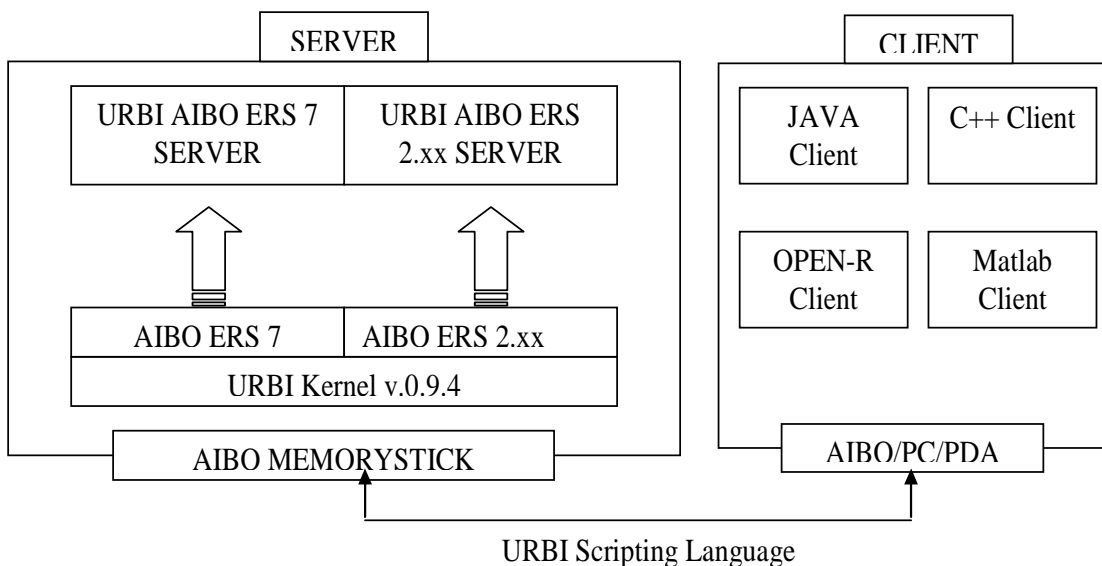


Figure 10: A schematic approach to URBI

Actually URBI (Universal Robotic Body Interface) is a client/server based interpreted language that can be used to control robots or complex systems of any kind. (The author prefers to call URBI – the URBI scripting language in order to identify it from the rest of the components of the framework) The **URBI language** is the core of the framework and it defines a standard protocol to give commands and receive messages from the machine to be controlled. The code written in the URBI language that can be executed by AIBO are called **URBI Scripts**. The creators of URBI provide some example scripts that proved to be more than helpful in creating new scripts. (please refer to: <http://www.urbiforge.com/urbiscripts.html> and to [2] for a specification of the URBI language)

#### 4.1.2 Why use URBI?

There's no doubt that all development choices have their advantages and disadvantages. If the author learned one thing during the development of this project is that the choices you make have to be tightly related to the outcomes that you expect. Therefore choosing one of these tools (or any other that will be available in the near future) is a matter of personal taste combined with well structured reasons related to the goals that you have.

Among the selection criteria that the author took into consideration in choosing a developing platform one can count: reusability on other platforms and robots, ease of use, programming language, testing and debugging methods, and flexibility, development speed.

URBI was preferred for this author because it is a new solution, under development and continuous improvement. Apart from being practical and offering a big speed in development this solution was found by the author to be innovative and oriented towards the future. Future developments in URBI promise to make it “the choice” for AIBO programming. (Please refer to <http://www.urbiforge.com/whatisurbi.html> )

The author wanted to concentrate the attention of this project on a new personality model and on the implications it has rather than concentrating on developing movements for a specific kind of robot. In this approach URBI helped a lot, first it is not dependent on AIBO and thus it will allow for the current prototype (with minor adaptations) to be tested on other robots also, and second writing scripts that make AIBO do actions is quick, easy to test and approachable even by non-software developers. Another aspect proffered in URBI was the division of actions in scripts. Thus AIBO's actions can be divided in small sub-actions that can be implemented by very small URBI scripts. Taking advantage of the fact that URBI scripts are represented as Strings in java, the author created a library of scripts that can be concatenated in order to create complex AIBO actions.

For example in creating an emotion for AIBO, taking “happiness” as an example the following scripts have been used:

```
String HappyEmotion = "";
...
HappyEmotion = Face.HAPPY(1) + "&" + Head.BARK(3.0)+ "&" + Tail.WAG(3,20)+ "&" +
Speak.BARKING(3)+ "; wait 2000;";
```

So the string that represents the command for an emotion is a concatenation of smaller commands and time operators. For example *Face.HAPPY(1)* returns a string that looks like this:

```
String sInt = String.valueOf(intensity);
HAPPY = "ledF7.val = "+ sInt+ " | ledF8.val = "+sInt+" | ledF11.val = "+sInt+" |
ledF12.val = "+sInt+" | modeG.val = 1" ;
```

where *sInt* is the string representation of the intensity of the emotion that is given as a parameter.

The library that sums all the URBI scripts used in this system is called *RobotOutput* and a Java API is available while the structure is presented in Appendix E.

### 4.1.3 Architecture

Choosing URBI had its consequences on the architecture of the system. Therefore I decided to place URBI Server on AIBO and don't bring any modifications to it and in return build the mind from AIBO on *liburbi-java* on a PC. While the system has the configuration of a remote control it runs differently and does not need human input. The unfortunate consequence of this system is that AIBO by itself is just a piece of hardware, only wirelessly connected to the PC it ketches life.



Figure 11: System Architecture

### 4.1.4 Java

In the development process of this system one major programming language was used: Java version 1.4. There are intrinsic and extrinsic reasons for choosing Java over other development environments or programming languages.

The intrinsic reasons for using java are practically the reasons that made it successful as stated by Bruce Eckel in [8].

- ∅ The reason Java has been so successful is that the goal was to solve many of the problems facing developers today. The goal of Java is improved productivity. This productivity comes in many ways, but the language is designed to aid you as much as possible, while hindering you as little as possible with arbitrary rules or any requirement that you use a particular set of features. Java is designed to be practical; Java language design decisions were based on providing the maximum benefits to the programmer.
- ∅ Classes designed to fit the problem tend to express it better. This means that when you write the code, you're describing your solution in the terms of the problem

- space rather than the terms of the computer, which is the solution space. You deal with higher-level concepts and can do much more with a single line of code.
- ∅ The other benefit of this ease of expression is maintenance, which (if reports can be believed) takes a huge portion of the cost over a program's lifetime. If a program is easier to understand, then it's easier to maintain. This can also reduce the cost of creating and maintaining the documentation.
  - ∅ The fastest way to create a program is to use code that's already written: a library. A major goal in Java is to make library use easier. This is accomplished by casting libraries into new data types (classes), so that bringing in a library means adding new types to the language. Because the Java compiler takes care of how the library is used—guaranteeing proper initialization and cleanup, and ensuring that functions are called properly—you can focus on what you want the library to do, not how you have to do it.
  - ∅ Error handling in C is a notorious problem, and one that is often ignored—finger-crossing is usually involved. If you're building a large, complex program, there's nothing worse than having an error buried somewhere with no clue as to where it came from. Java *exception handling* is a way to guarantee that an error is noticed, and that something happens as a result.
  - ∅ Many traditional languages have built-in limitations to program size and complexity. BASIC, for example, can be great for pulling together quick solutions for certain classes of problems, but if the program gets more than a few pages long, or ventures out of the normal problem domain of that language, it's like trying to swim through an ever-more viscous fluid. There's no clear line that tells you when your language is failing you, and even if there were, you'd ignore it. You don't say, "My BASIC program just got too big; I'll have to rewrite it in C!" Instead, you try to shoehorn a few more lines in to add that one new feature. So the extra costs come creeping up on you.
  - ∅ Java is designed to aid *programming in the large*—that is, to erase those creeping-complexity boundaries between a small program and a large one. You certainly don't need to use OOP when you're writing a "hello world" style utility program, but the features are there when you need them. And the compiler is aggressive about ferreting out bug-producing errors for small and large programs alike.

The biggest issue with Java is performance. Interpreted Java has been slow, even 20 to 50 times slower than C in the original Java interpreters. This has improved greatly over time, but it will still remain an important number. Given the fact that our system need to perform as a real-time system the question of speed was crucial but I decided to take the risk.

The extrinsic reasons for using java:

- ∅ Java is platform independent, meaning that it should have little problems running on a handheld or any other platform in the future (including other future robots, etc).
- ∅ There is a lot of knowledge available about how to handle problems that may arise and the author feels the most comfortable with this language.
- ∅ The availability of a package of URBI classes written in Java (liburbi\_java\_0.9.1 that was a major speed up in the development).

### 4.1.5 Rule-based systems

Rule-based programs are everywhere. Their applications include everything from mail filtering to order configuration, and from monitoring chemical plants to diagnosing medical problems. Rule-based programs excel at solving problems that are difficult to solve using traditional algorithmic methods, and they work even when the input data is incomplete. A rule is a kind of instruction or command that applies in certain situations. [Jess in Action] Using this very general definition, you might conclude that all the knowledge you have about the world can be encoded as rules. Experience shows that this is often (but not always) the case. In general, any information you can think about in logical terms can be expressed as rules. Rules are a lot like the *if-then* statements of traditional programming languages. The *if* part of a rule is often called its *left-hand side* (often abbreviated **LHS**), *predicate*, or *premises*; and the *then* part is the *right hand side* (**RHS**), *actions*, or *conclusions*. The domain of a rule is the set of all information the rule could possibly work with. **A rule-based system is a system that uses rules to derive conclusions from premises.**

Expert systems, rule-based computer programs that capture the knowledge of human experts in their own fields of expertise, were a success story for artificial intelligence research in the 1970s and 1980s. Early, successful expert systems were built around rules (sometimes called heuristics) for medical diagnosis, engineering, chemistry, and computer sales. Today, general rule-based systems, both those intended to replace human expertise and those intended to automate or codify business practices or other activities, are a part of virtually every enterprise.

A typical rule engine contains:

- ∅ An inference engine
- ∅ A rule base
- ∅ A working memory

The inference engine, in turn, consists of:

- ∅ A pattern matcher
- ∅ An agenda
- ∅ An execution engine

The **inference engine** controls the whole process of applying the rules to the working memory to obtain the outputs of the system.

The **rule base** contains all the rules the system knows

You also need to store the data your rule engine will operate on. In a typical rule engine, the working memory, sometimes called the **fact base**, contains all the pieces of information the rule-based system is working with. The working memory can hold both the premises and the conclusions of the rules.

Your inference engine has to decide what rules to fire, and when. The purpose of the **pattern matcher** is to decide which rules apply, given the current contents of the working memory.

Once your inference engine figures out which rules should be fired, it still must decide which rule to fire first. The list of rules that could potentially fire is stored on the **agenda**. The agenda is responsible for using the conflict strategy to decide which of the rules, out of all those that apply, have the highest priority and should be fired first.



Finally, once your rule engine decides what rule to fire, it has to execute that rule's action part. The **execution engine** is the component of a rule engine that fires the rules.

#### 4.1.6 Jess

The reasoning component of AIBO's mind is developed using Jess [JESS]. Jess stands for Java Expert System Shell. In short Jess is an expert system that works with facts, and rules that are automatically triggered when the conditions are met. Jess is actually the Java version of CLIPS [CLIPS], with some added functionality to cooperate better with other Java classes. The fact that Jess is written in Java made the choice of using it easy. It should give little problem to embed the Jess component in the rest of the application, written in Java.

Jess uses a very efficient version the rules finding facts algorithm, known as the *Rete* algorithm. Briefly, the *Rete* algorithm eliminates the inefficiency in the simple pattern matcher by remembering past test results across iterations of the rule loop. Only new or deleted working memory elements are tested against the rules at each step. The *Rete* algorithm is implemented by building a network of nodes; each representing one or more tests on a rule LHS. [11]

In the current prototype AIBO's central cognitive unit comprises of two coupled rule-based systems where the first one generates the fact-base for the second one. In the same way the second rule-based system modifies facts in the first one and resumes its execution. The two rule-based systems run in parallel if necessary and control each other's execution. The two expert systems present here extensively use shadow facts (Java Beans) and comprise of very large rule-bases that are set according to the goals of any specific application.

## 4.2 UML Diagrams

*"The indispensable first step to getting the things you want out of life: decide what you want."*

**Ben Stein**

The Unified Modeling Language (UML) is a modeling language for specifying, visualizing, constructing, and documenting the artifacts of a system intensive process. It was originally conceived by Rational Software Corporation and three of the most prominent methodologists in the information systems and technology industry, Grady Booch, James Rumbaugh, and Ivar Jacobson (the Three Amigos). The language has gained significant industry support from various organizations via the UML Partners Consortium and has been submitted to and approved by the Object Management Group (OMG) as a standard (November 17, 1997). [22]

Diagrams depict knowledge in a communicable form. The UML provides the following diagrams, organized around architectural views, regarding models of problems and solutions:

- The User Model View
  - *Use case diagrams* depict the *functionality* of a system.
- The Structural Model View
  - *Class diagrams* depict the *static structure* of a system.
  - *Object diagrams* depict the static structure of a system at a particular time.

- The Behavioral Model View
  - *Sequence diagrams* depict the *specification* of behavior.
  - *Collaboration diagrams* depict the *realization* of behavior.
  - *State diagrams* depict the *status conditions* and *responses* of participants involved in behavior.
  - *Activity diagrams* depict the *activities* of participants involved in behavior.
- The Implementation Model View
  - *Component diagrams* depict the *organization* of solution components.
- The Environment Model View
  - *Deployment diagrams* depict the *configuration* of environment elements and the mapping of solution components onto them.

Fundamentally, diagrams *depict* knowledge (syntax).

Because the foundation of the UML constitutes the *necessary* and *sufficient* engineering practices for problem solving, processes that utilize the UML are assured of resting upon a foundation that provides the *potential* for success.[22]

In the next sections we will describe the system according to these five model views, and will use one UML diagram per model. Due to the complexity of the software presented and to only some of the most relevant diagrams will be shown and explained.

#### 4.2.1 The user model view

Use cases are used during requirements elicitation and analysis to represent the functionality of the system. Use cases focus on an external view of the system.

A Use Case represents a discrete unit of interaction between a user (human or machine) and the system. A Use Case is a single unit of meaningful work; for example **login to system**, **send input**, **reason**, are all Use Cases. Each Use Case has a description which describes the functionality that will be built in the proposed system. A Use Case may 'include' another Use Case's functionality or 'extend' another Use Case with its own behavior.

Use Cases are typically related to 'actors'. An Actor is a user of the system. This includes both human users and other computer systems. An Actor uses a Use Case to perform some piece of work which is of value to the business. The set of Use Cases an actor has access to define their overall role in the system and the scope of their action.

Figure 12 presents the Use Case diagram developed in the analysis phase of this project.

Actors:

- Human User- under this category we include: any human, robot or animal that will interact in any way with AIBO: family, best-human friend, other AIBOs, other animals, etc
- AIBO - this actor symbolizes practically the physical AIBO with it's sensors and actuators. This actor is practically represented by the URBI Server present on AIBO.
- AIBO Brain – this actor is practically the central unit of the robot that takes all decisions regarding the actions and emotions of the robot itself. This actor is practically represented by the URBI Client running on a PC/PDA.

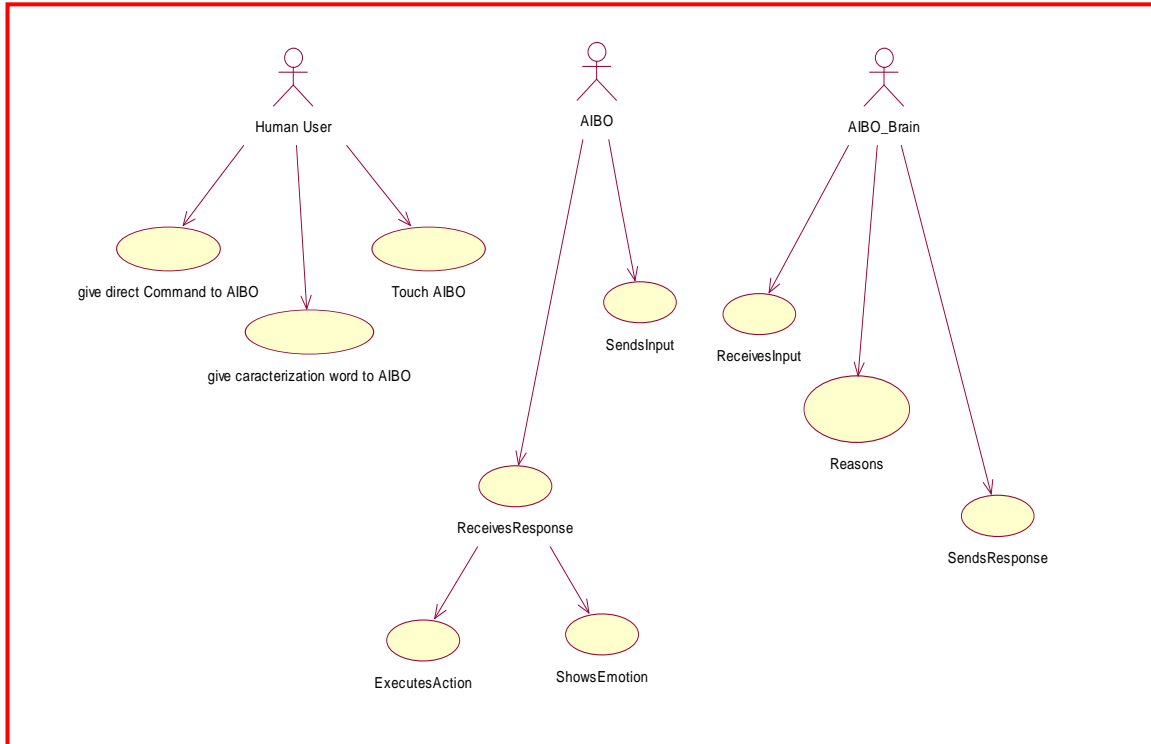


Figure 12: Use Case Diagram

Use-cases

- Touch AIBO – The human actor can interact with the physical robot by touching one of his sensors.
- Give direct command to AIBO – The human actor interacts with the physical robot by giving AIBO a direct command. In this prototype the human gives AIBO a direct command by activating a window and choosing one of the possible predefined commands.(“play ball”, “follow me”, etc) We expect that in the future AIBO will receive commands from humans by recognizing designated cards or designated voice commands.
- Give characterization word to AIBO - The human actor interacts with the physical robot by encouraging or discouraging him. In this prototype the human praises AIBO by activating a window and choosing one of the possible predefined commands (“good boy”, etc). We expect that in the future humans will praise AIBO through designated cards or voice commands.
- Send Input – AIBO Server monitors all sensors and sends corresponding messages to AIBO Client.
- Receive Response – AIBO Server receives from AIBO Client commands to be executed by actuators.
- Execute Action – AIBO Server interprets the commands received and activates the corresponding actuators resulting in actions like motions, etc
- Show Emotion - AIBO Server interprets the commands received and activates the corresponding actuators resulting in displaying an emotion. (a combination of small movements of body parts, sound and LEDs)
- Receive Input – AIBO Client receives from the Server messages representing sensor inputs.

- Reason – AIBO Client judges based on the inputs he receives and the constant elements in the system resulting in reactions.
- Send Response – AIBO Client takes the reaction outputted by the reasoning use case and sends it back to the server for interpretation and execution.

#### 4.2.2 The structural model view

The Structural Model is at the core of object-oriented development and design - it expresses both the persistent state of the system and the behavior of the system. A class encapsulates state (attributes) and offers services to manipulate that state (behavior). Class Diagrams are used to describe the structure of the system. Classes are abstractions that specify the common structure and behavior of a set of objects. Objects are instances of classes that are created, modified and destroyed during the execution of the system. An object has a state which includes the values of its attributes and its relationships with other objects.

Due to the complexity of the system the classes are practically divided in a few class diagrams presented and explained in the following pages.

First the author will introduce the class diagram for the sub-unit that controls the running of the entire system. The sub-unit is symbolically named Main (see Figure 13), even though it does not exist as an entity. In this diagram are presented just the classes that have a central role in the running of the entire system: *AIBOClient* and *CompanionAIBO*.

*AIBOClient* is a singleton class that manages the connection to the server on AIBO. This class initializes *UClient* from *liburbi* and has a *RobotCallback* attribute. All communication with AIBO is made through this class. *AIBOClient* uses *RobotOutput* package that is a collection of commands that can be interpreted by the Server on AIBO. One of this classes attribute is <COMMANDS> a string that contains at any given time the string that is sent to AIBO for execution. Any class that needs to send commands to AIBO has to append the command string to <COMMANDS> (which is static) and then call the method <SendCommands()>. For explanations regarding a singleton class and the singleton pattern please refer to section 4.4.1.

*RobotCallback* is the class that implements *CallbackListener* interface from *liburbi* package. Its role is to signal *AIBOClient* that AIBO have finished executing the previous command.

*CompanionAIBO* is the class that contains the main of the entire system and in the same time is the class that initializes and runs the main threads of the system. Indirectly *CompanionAIBO* initializes the entire system in a ready to run state.

These classes have been included in package *companionAIBO* solely for organizational purposes.

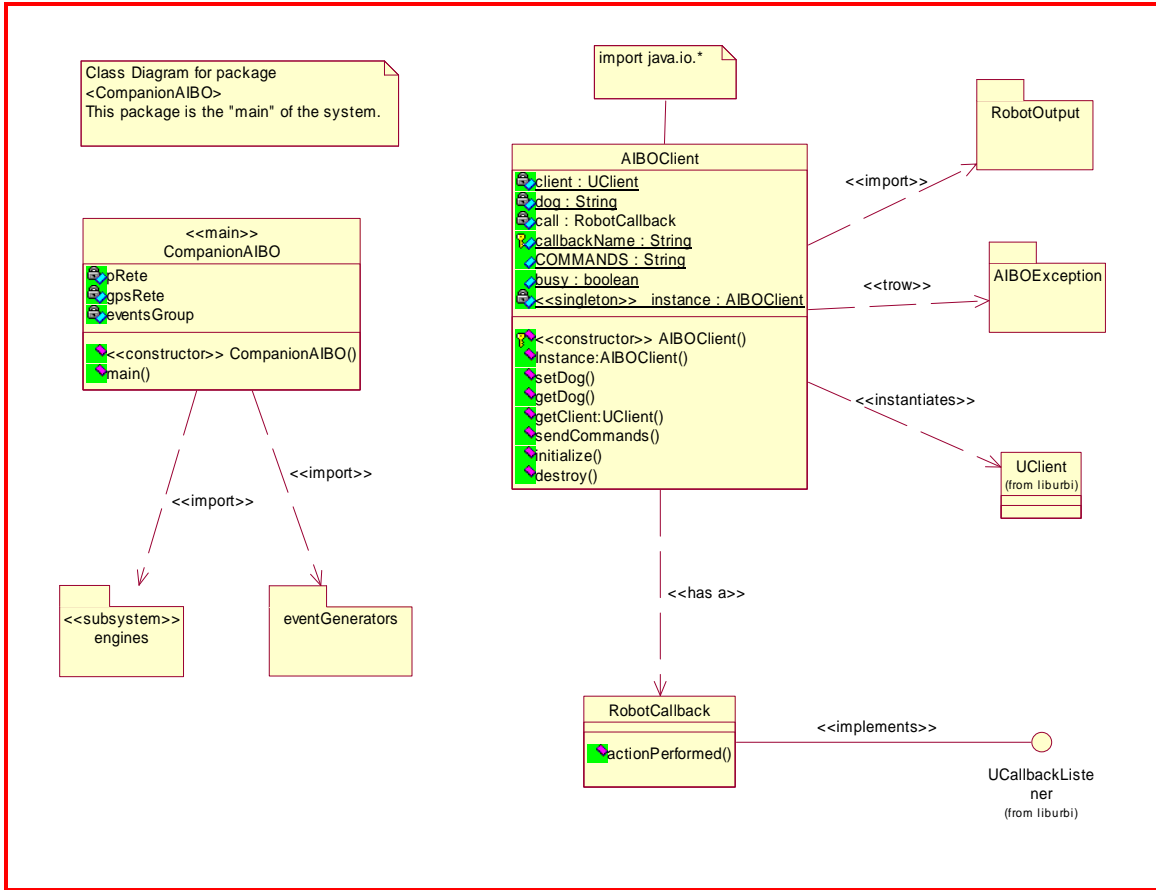


Figure 13: Class Diagram for Main

The system includes a package that has the responsibility of initializing the cognitive module of AIBO. This package makes up the *initialization* component. The class diagram for this package is presented in Figure 14.

*InitPersonality* has the responsibility of reading a text file (traits.txt) that contains the values for the five traits of personality.

*InitNeeds* has the responsibility of reading a text file (needs.txt) that contains the values for the 6 need layers in Maslow’s pyramid of needs.

<p><b>traits.txt</b></p> <pre> O = openness 0 C = conscientiousness 0 E = extraversion 75 A = agreeableness 85 N = neuroticism 60                 </pre>	<p><b>needs.txt</b></p> <pre> BATTERY = 0 1 SAFETY = 0 2 LOVE = 40 3 BELONGING = 40 3 ACHIEVEMENT = 60 4 RECOGNITION = 60 4 FULLFILMENT = 60 5                 </pre>
--	---

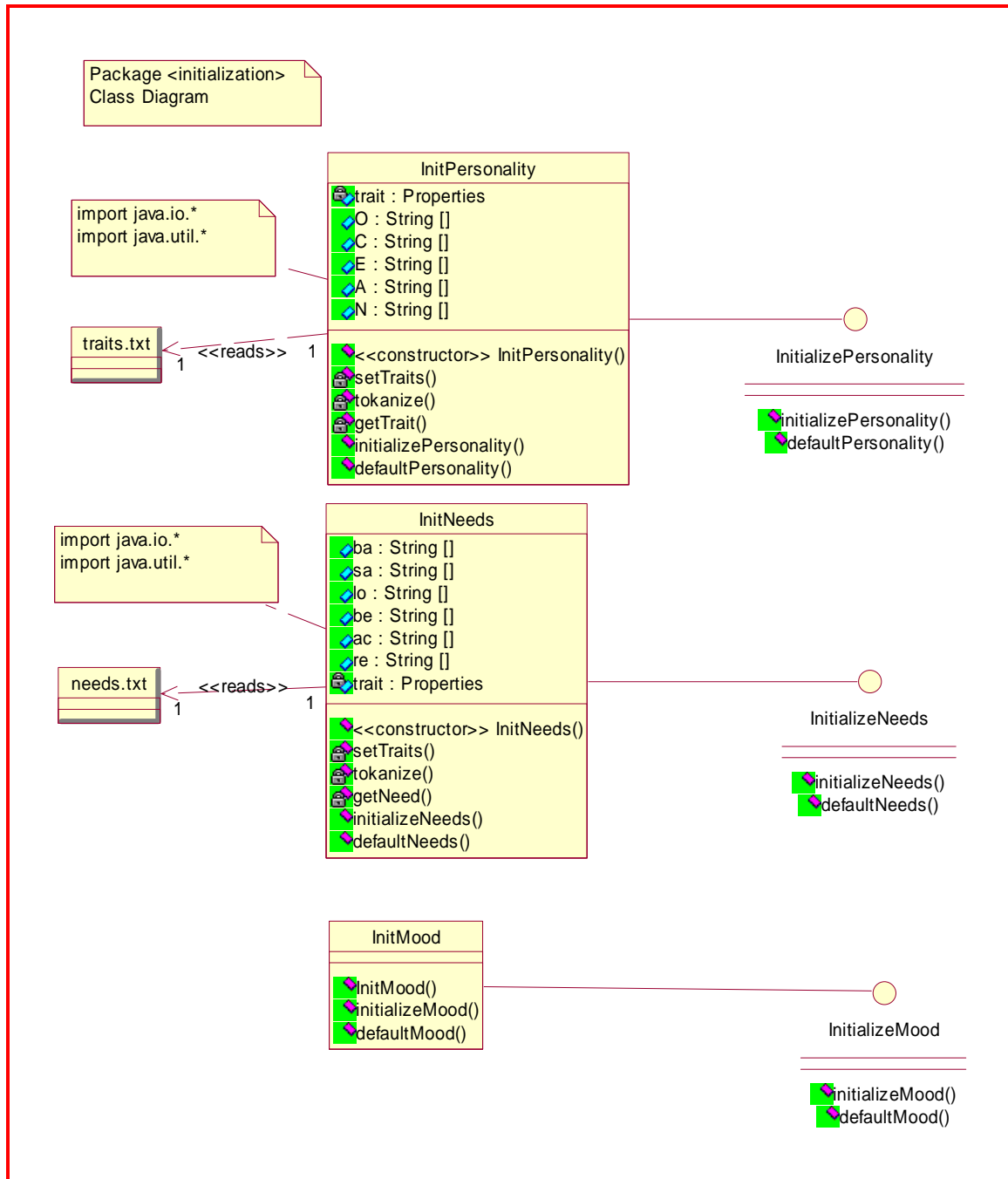


Figure 14: Class Diagram for package <initialization>

Package *eventGenerators* contains the classes that control the generation of events (input) upon which AIBO reasons (see Figure 15). Each of the event generators runs in its own thread that are controlled by a thread group. Thread groups provide a mechanism for collecting multiple threads into a single object and manipulating those threads all at once, rather than individually.

*EventsGroup* instantiates all threads that generate events and starts them. The *start* function is only called once in the system. This group dies only when the system dies since we want the brain continuously listening for events.

*SensorEvents* has the responsibility of creating events that come from the external sensors of AIBO (touch, distance).

*EventGenerator* throws events that relate to internal factors in AIBO: timer and battery.

*CharacterizationGenerator* creates characterization events received through the *CharacterizationForm* from the user.

*DirComGenerator* creates command events received through the *DirectCommandsForm* from the user.

All event generator threads instantiate event classes contains in package *engines.events*.

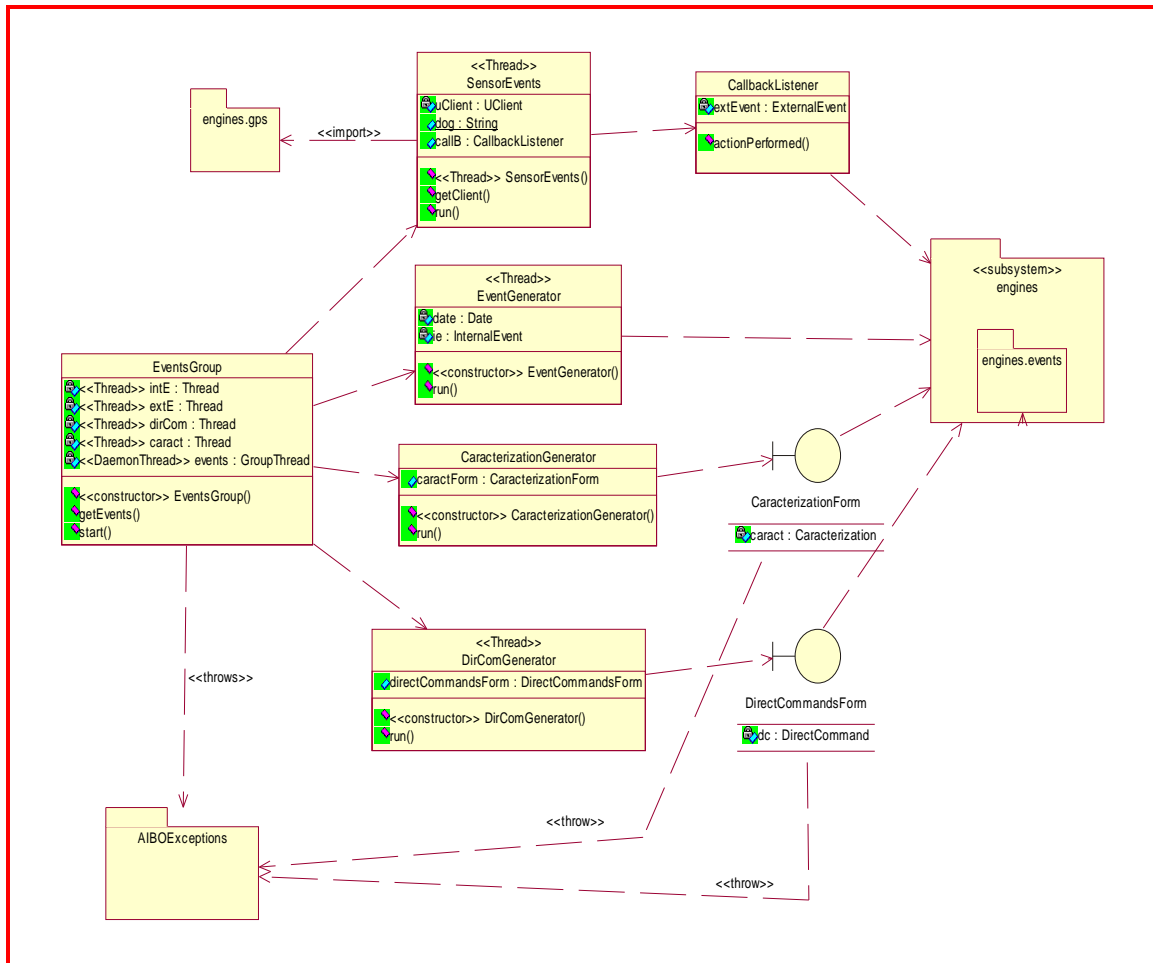


Figure 15: Class Diagram for package <eventGenerators>

Below we present the central unit of the system: the brain that reasons on input and generates reactions (actions and emotions). This unit is comprised in package *engines* that contains two other sub-packages: *engines.events* and *engines.gps*. As was explained before the central unit comprises of two rule-based systems that are tightly coupled and that run in parallel. The functioning of the two rule-based systems is explained in section 4.5.3.

*Personality*, *Mood*, *Needs* and *Emotion* are all classes that represent circumstances for the reasoning. They have been implemented as Java Bean components (see section on Java Beans) all implement the *Serializable* interface in order to assure a coherent functioning of the system. *Personality*, *Mood* and *Needs* are all singleton classes (see section on Singleton

Pattern) since we want to allow the existence of only one instance of these classes in the system. The existence of two Personality objects in the system would compromise the entire functioning therefore the singleton pattern enforces the initialization of solely one object.

The two rule-based systems are encapsulated in two .clp files: *persRules.clp* and *gpsRules.clp*. These two expert systems are wrapped in two threads that control their execution in the system. The two expert systems run in an infinite loop in parallel until the system is stopped by an external factor. *PersThread* and *GPSThread* both are thread that instantiate a *Rete* object. *PersThread* is started by the main class of the system: *CompanionAIBO*. *GPSThread* is started by *PersThread* and further the two threads control each others sleeping and resume time.

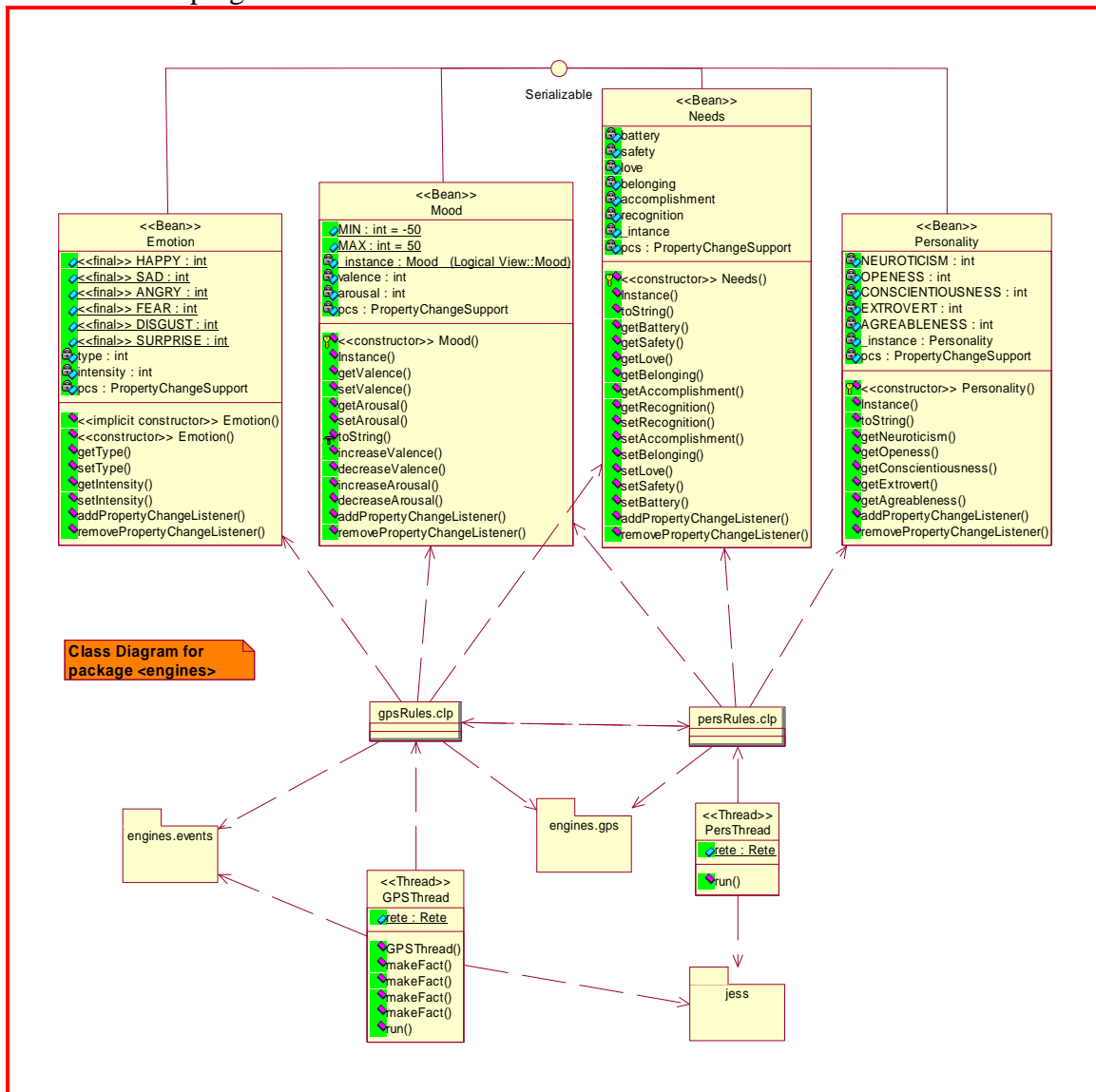


Figure 16: Class Diagram for package <engines>

Part of package engines is *engines.gps*. The package is fairly simple and not presented in detail. It practically encapsulates the concepts of *preferences*, *standards* and *goals*.



Everything has been created as a bean in order to ensure that object that instantiate these classes can be used by the two expert systems and in the same time they can be accessed by external applications to be used in other contexts. This is one of the key elements that confer to this system flexibility and in the same time **extensibility**. For example it would be easy to collect all preferences in a data structure and output results in other application or do some other kind of operation with them. Classes *Goal*, *Preference* and *Standard* extend the main class *GPS* that contains all the common attributes of the child classes.

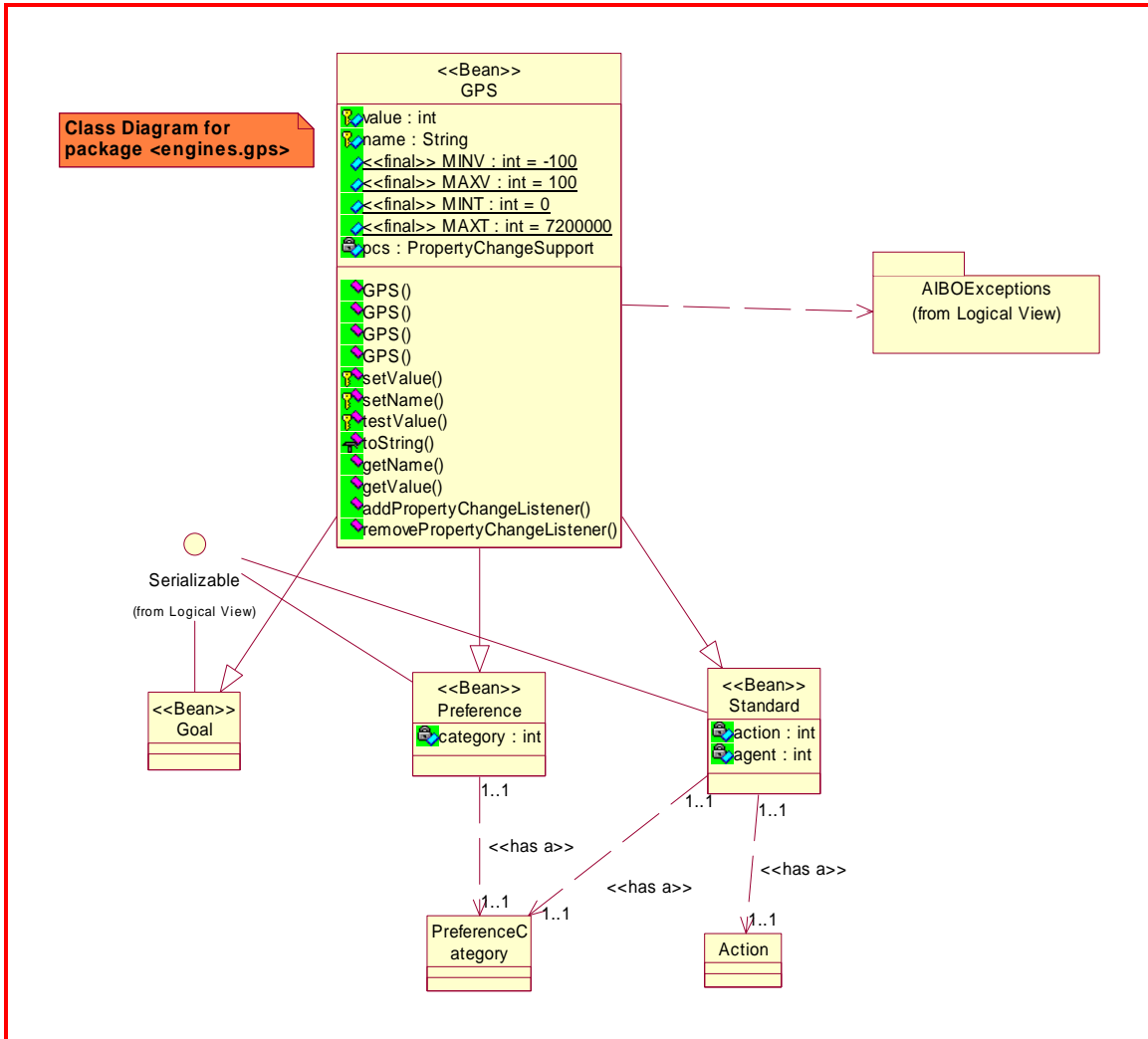


Figure 17: Class Diagram <engines.gps> (not-detailed)

In the same global package *engines*, I also included a sub-package *engines.events*. This package includes all the possible events that can be signaled to the reasoning component. As in the case of the *engines.gps* package, there is a parent class *Event* that is inherited by all categories of events: internal, external, direct commands and characterizations. Also all classes are Java Beans in order to be able to use the instances of the class as facts in the expert system and in the same time are able to handle the object separately as needed. In the same way the child classes implement the *Serializable* interface in order to make sure that only one access is permitted at a given time. The class diagram for this package is

presented in Figure 18. Details regarding the contents of this package are to be found in Appendix B.

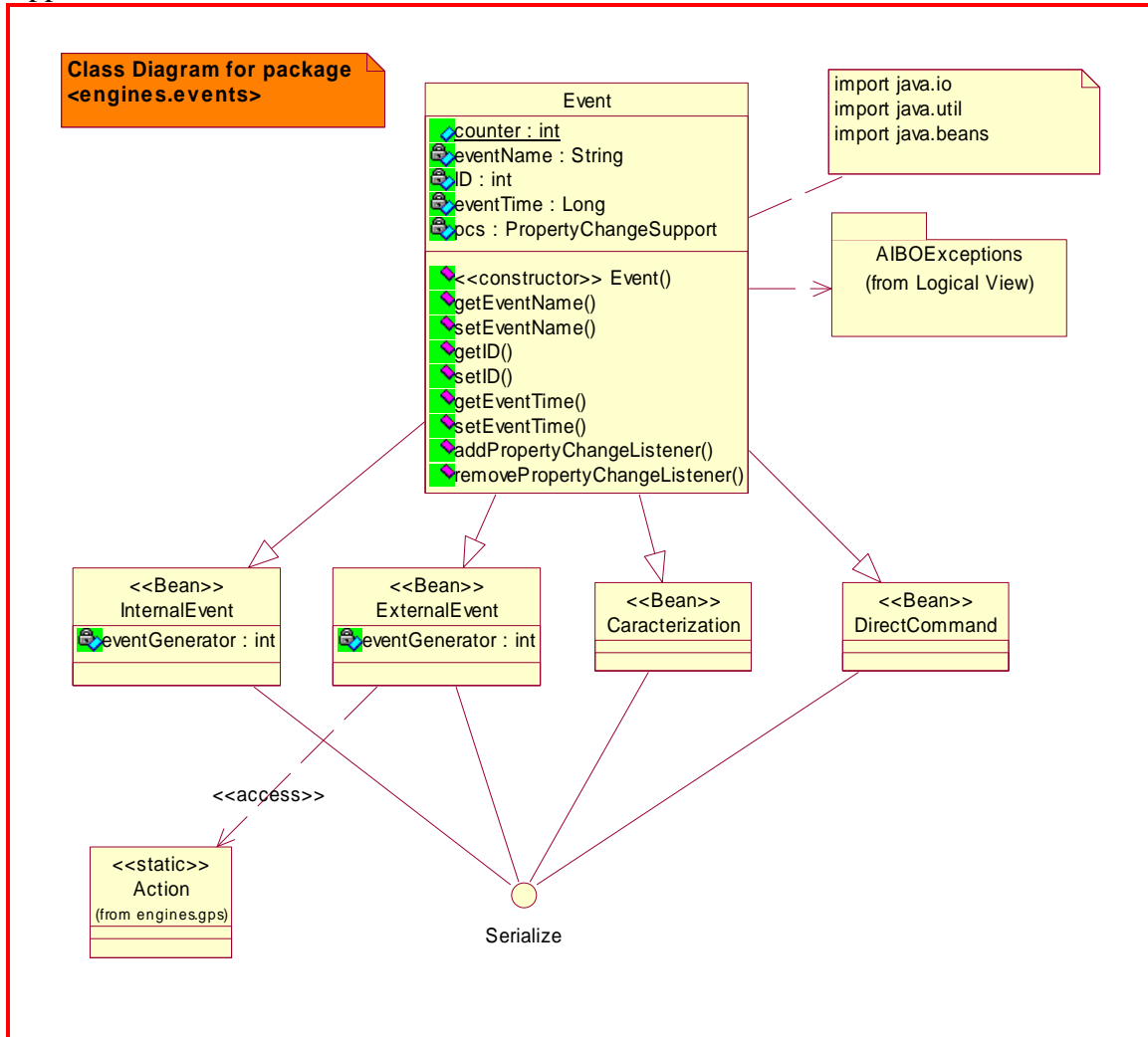


Figure 18: Class Diagram for package <engines.events> (not-detailed)

### 4.2.3 The behavioral model view

UML provides a graphical means of depicting object interactions over time in Sequence Diagrams. These typically show a user or actor, and the objects and components they interact with in the execution of a use case. One sequence diagram typically represents a single Use Case 'scenario' or flow of events. Sequence diagrams are an excellent way to document usage scenarios and to both capture required objects early in analysis and to verify object usage later in design. Sequence diagrams show the flow of messages from one object to another, and as such correspond to the methods and events supported by a class/object.

A sequence diagram is presented below to give an overview of the way objects mainly in the “reason” use-case are interacting between themselves and with the other components of the system. (Please consider this an overview diagram made in order for the reader to understand the way the system is created. The diagram presented below is not a detailed sequence diagram for any of the use-cases)

The second expert system is initialized by the first expert system when this is creating part of its fact-base. Then they both go to sleep until an event generator throws to the second expert system an

event. This starts the pattern matching and executes one of the rules. Executing a rule has some consequences one of them being starting an interpreting thread and the second one being waking up the first expert system to execution. The first expert system then updates the fact-base for the second one.

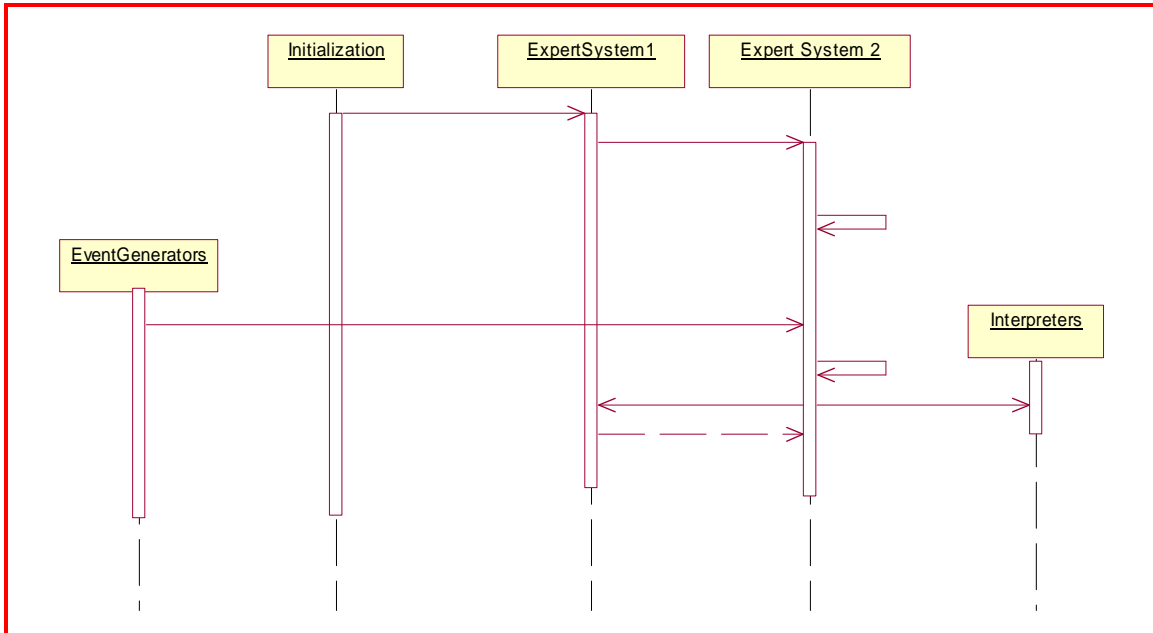


Figure 19: Sequence Diagram

Activity diagrams are used to show how different workflows in the system are constructed, how they start and the possibly many decision paths that can be taken from start to finish. They may also illustrate the where parallel processing may occur in the execution of some activities.

The following activity diagram gives an overview of the main threads in the application. (Please consider this a scheme for an actual activity diagram. The diagram is presented below just to make sure the reader understands the way the threads are scheduled in the application)

The application starts and the first things it does is to take care to create a connection to the robot AIBO, to initialize the first expert system and to start the event generators. The application terminates when the event generators stop running. All event generators are grouped in a thread group. Each event generator owns its own thread but this is not shown in the diagram. The two rule-based engine threads run in parallel in an endless loop and are destroyed only when there are no more event generators that send them events. The entire engine is awakening only when an event is caught; in the absence of any events being caught the two engines are sleeping and waiting for events to activate them. In the eventuality of events coming in very quickly the processor allocates time in turns to each thread creating the impression of a parallel application.

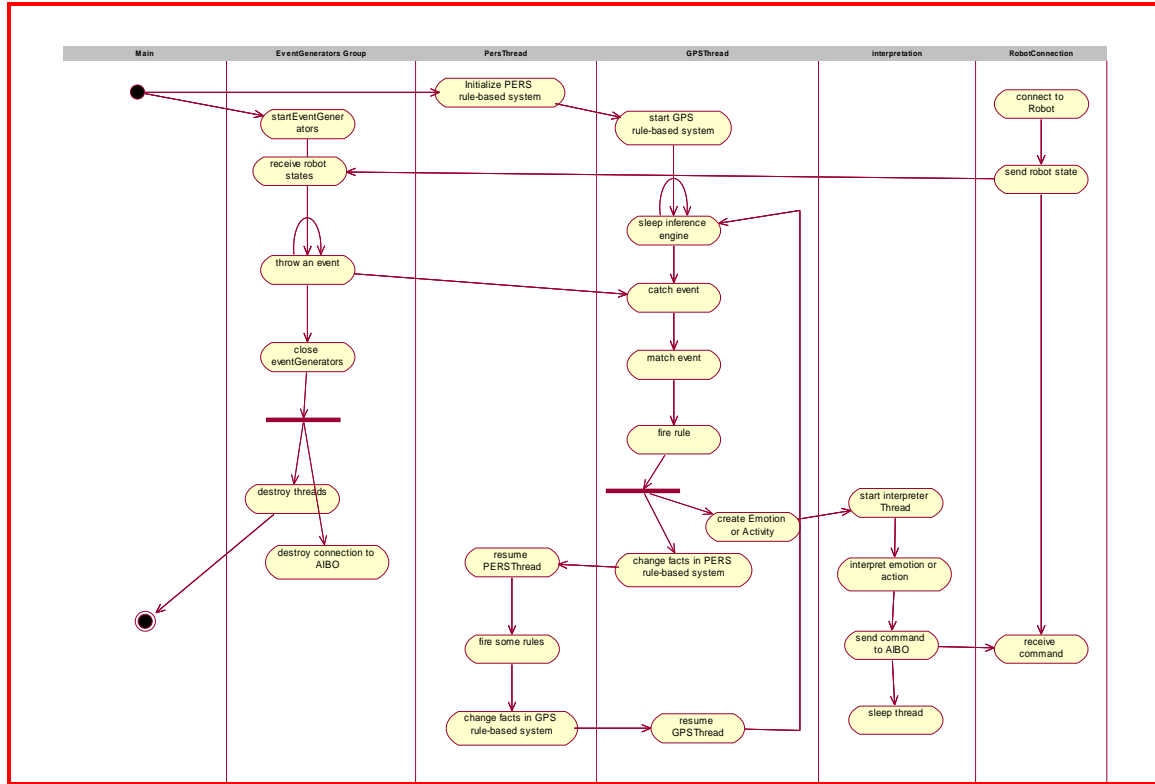


Figure 20: Activity Diagram

#### 4.2.4 The implementation model view

##### Component Diagrams:

The component model illustrates the software components that will be used to build the system. These may be built up from the class model and written from scratch for the new system, or may be brought in from other projects and 3rd party vendors. Components are high level aggregations of smaller software pieces, and provide a 'black box' building block approach to software construction.

A **component** represents a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces. It may, for example, be source code, binary, or executable. Examples include executables such as a browser or HTTP server, a database, a DLL, or a JAR file. [22]

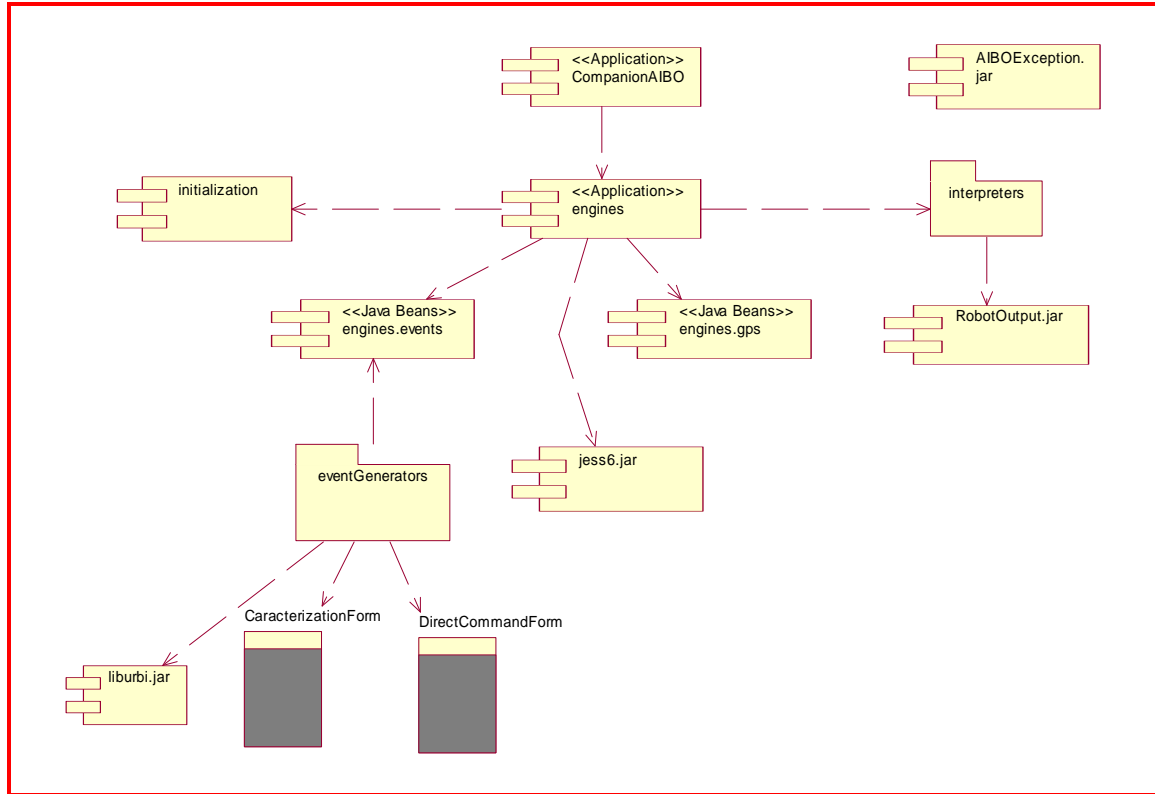


Figure 21: Component Diagram

**Package Diagrams:**

Even though Package Diagrams are not a compulsory part of a correctly modeled system, in this case the author presents the diagram in order to make the system more understandable for the reader.

Figure 22 presents an overview of the units that make up the entire system that resides on the PC. The *URBI Server* and the software that runs on the AIBO will not be presented since it was not altered in any way by the author and is presented in the literature referenced.

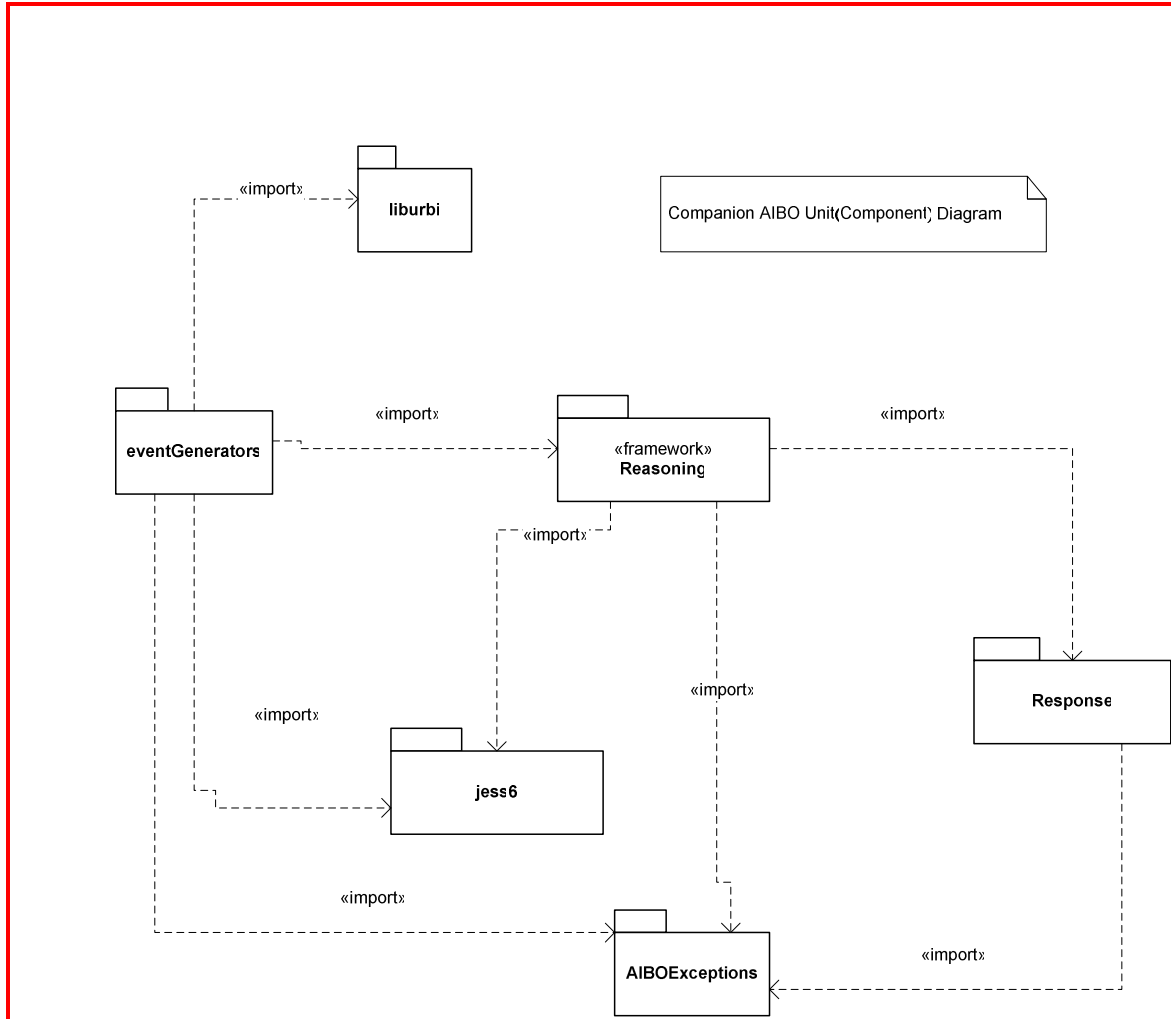
Since the *Response* and *Reasoning* units are comprised of more packaged that are correlated are presented in detail in Figure 23 and 24. These diagrams are important mainly to explain the coupling of the different parts of the system and their roles in the architecture.

Practically the system is based on a input-reason-response architecture. Where input is encapsulated in a *eventGenerator* package, reason is a complex framework that comprises of more packages and response is a unit made up on more output packages. There are two external components being used: *liburbi* that is a package that offers an easy link towards the URBI Server on AIBO, and *jess6* which is a package used to create the two rule-based systems in the reasoning component.

Package *eventGenerators* – contains all threads and classes that encapsulate the mechanisms that transform internal and external happening in events that are recognizable by the central reasoning unit.

Package *AIBOException* is a package that contains all possible exceptions thrown by our system. A class diagram is not presented since all exceptions inherit *java.Exception* and the diagram and implementation is trivial.

Unit *Reasoning* comprises of packages: *initialization*, *engines*, *engines.gps* and *engines.events*.



**Figure 22: Programming Units Diagram: Companion AIBO Overview**

- ∅ Package *initialization* comprises of the classes and external files that assure the initialization of the circumstances for reasoning: personality and needs.
- ∅ Package *engines* comprises of the thread classes that assure the running of the central unit of the system: AIBO’s brain. The two rule based systems running in parallel are present here.
- ∅ Package *engines.gps* contain all the classes that instantiated represent facts in the first rule-based systems (goals, preferences and standards).
- ∅ Package *engines.events* contains all the classes that encapsulate possible events that are thrown into the system by the event generators and that instantiated represent facts in the second rule-based system. (Internal and external events, etc)

Unit *Response* comprises of packages: *interpreters* and *RobotOutput*

- ∅ Package *Interpreters* is a trivial containing just two classes that act as dictionaries in-between the results of the reasoning component and the messages sent to the server.
  - ∅ Package *RobotOutput* compiles in java class files strings representing URBI – AIBO commands. This package is presented in detail in Appendix D.
- Class diagrams for all these packages are described in section 4.3.2 therefore no further details will be added.

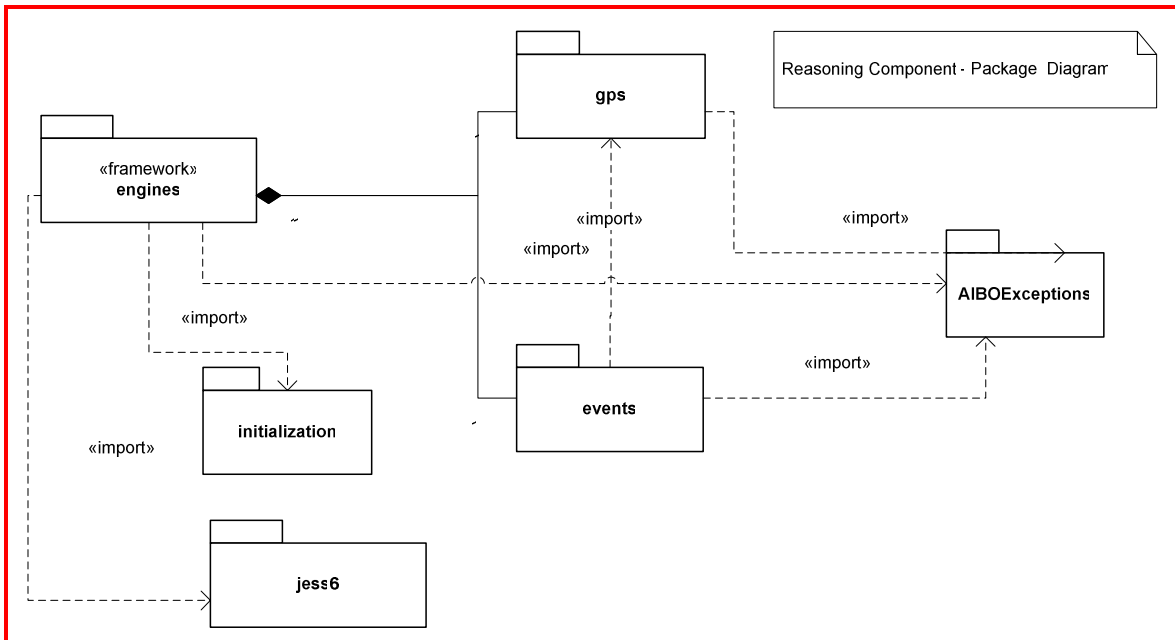


Figure 23: Package Diagram: Reasoning Component

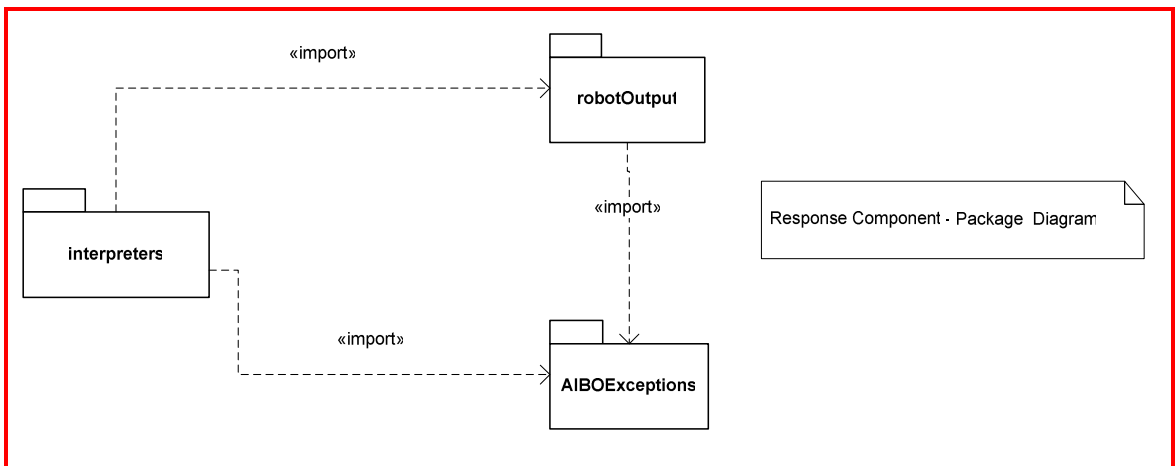


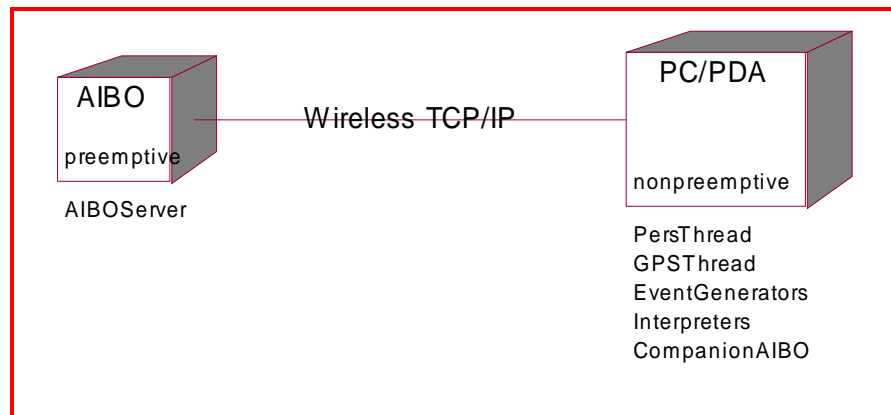
Figure 24: Package Diagram: Response Component

#### 4.2.5 The environment model view

The Physical/Deployment Model provides a detailed model of the way components will be deployed across the system infrastructure. It details network capabilities, server

specifications, hardware requirements and other information related to deploying the proposed system. The physical model shows where and how system components will be deployed. It is a specific map of the physical layout of the system. A deployment diagram illustrates the physical deployment of the system into a production (or test) environment. It shows where components will be located, on what servers, machines or hardware. It may illustrate network links, LAN bandwidth & etc.

Figure 25 presents the deployment diagram for our system. The system comprises of two entities: the server on the physical AIBO hardware and a client that runs on a PC/PDA without restrictions concerning the operating system since the entire system is written in java 1.4. The two communicate through a TCP/IP wireless connection.



**Figure 25: Deployment Diagram**

### 4.3 Design Patterns

*“Few things are harder to put up with than a good example.”*

**Mark Twain**

The term “**design patterns**” sounds a bit formal to the uninitiated and can be somewhat off-putting when you first encounter it. But, in fact, design patterns are just convenient ways of reusing object-oriented code between projects and between programmers. The idea behind design patterns is simple-- write down and catalog common interactions between objects that programmers have frequently found useful.

The definition of patterns favored by the author is the one given by Gamma in 1993: “Patterns identify and specify abstractions that are above the level of single classes and instances, or of components.” [14]

The 23 design patterns selected for inclusion in the original “*Design Patterns*” [14] book were ones which had several known applications and which were on a middle level of generality, where they could easily cross application areas and encompass several objects. The authors divided these patterns into three types: creational, structural and behavioral.



- *Creational patterns* are ones that create objects for you, rather than having you instantiate objects directly. This gives your program more flexibility in deciding which objects need to be created for a given case.
- *Structural patterns* help you compose groups of objects into larger structures, such as complex user interfaces or accounting data.
- *Behavioral patterns* help you define the communication between objects in your system and how the flow is controlled in a complex program.

Several of these patterns have been spotted by the author in the course of the development of the prototype and are described below together with some characteristics.

### 4.3.1 Singleton Pattern

The Singleton pattern is grouped with the other Creational patterns, although it is to some extent a “non-creational” pattern. There are many numbers of cases in programming where you need to make sure that there can be one and only one instance of a class. For example, your system can have only one window manager or print spooler, or a single point of access to a database engine. In our case it is crucial that in the entire system there is only one instance of the personality, mood, and needs classes therefore the singleton pattern is used. There are many ways to implement the singleton pattern and it is not the purpose of this paper to present them.

For this system the author choose to use the approach suggested by “*Design Patterns*” in [14], in which a Singletons is created using a static method to issue and keep track of instances. To prevent instantiating the class more than once, we make the constructor private so an instance can

```
public class Personality implements Serializable
{
    private static Personality _instance = null;

    protected Personality()
    {
        //constructor
        ....
    }
    /**
     * return the current instance for the personality class
     * @return Personality - the instance of this class
     */
    public static Personality Instance()
    {
        If (_instance == null)
        {
            _instance = new Personality();
        }
        return _instance;
    }
    ... ..
}
```

only be created from within the static method of the class.

For an example please see an extract from “**Personality.java**” class above.

If the singleton already exists-- you simply get a null return from the Instance method:

```
Personality p1 = Personality.Instance();
```

And, should you try to create instances of the Personality class directly, this will fail at compile time because the constructor has been declared as protected.

```
//fails at compile time because constructor is protected
Personality p2 = new Personality();
```

In a large, complex program it may not be simple to discover where in the code a Singleton has been instantiated. In Java, global variables do not really exist, so you can't save these Singletons conveniently in a single place. One solution is to create such singletons at the beginning of the program and pass them as arguments to the major classes or functions that might need to use them.

Normally it can be difficult to subclass a Singleton, since this can only work if the base Singleton class has not yet been instantiated. In our case the constructors of Singleton classes have been declared **protected**, therefore it is easier to subclass the base classes if in the future this will be needed.

### 4.3.2 Other Patterns

Other patterns are also present in the system but the author did not allocate enough time to the study of them and therefore are not presented here in detail.

## 4.4 Implementation

*"In theory, there is no difference between theory and practice. But, in practice, there is."*

**Jan L.A. van de Snepscheut**

In the course of the implementation there are a few ideas and concepts that are widely used and deserve to be exemplified and explained here. Likewise in this section I will explain the functioning of the central reasoning unit of AIBO from a software point of view.

### 4.4.1 Threads and Groups of Threads

A thread is a single sequential flow of control within a program. There is nothing new in the concept of a single thread. The real hoopla surrounding threads is not about a single sequential thread. Rather, it's about the use of multiple threads in a single program, running at the same time and performing different tasks. The `run` method gives a thread something

```
public class PersThread extends Thread
{
    public static Rete rete = new Rete();
    public PersThread() {
        try {
            rete.executeCommand("(batch engines/persRules.clp)");
            rete.executeCommand("(set-reset-globals nil)");
            rete.executeCommand("(reset)");
            rete.executeCommand("(focus PERS)");
        }
        catch(JessException jesse) {
            jesse.printStackTrace();
        }
    }

    public synchronized void run() {
        try{
            rete.runUntilHalt();
        }
        catch(JessException je) {}
    }
}
```

to do. Its code implements the thread's running behavior. It can do anything that can be encoded in Java statements. There are two techniques for providing a run method for a thread. The first way to customize what a thread does when it is running is to subclass `Thread` (itself a *Runnable* object) and override its empty `run` method so that it does something. This was the proffered method used by the author. Please see the `PersThread` class below for an example:

Every Java thread is a member of a *thread group*. Thread groups provide a mechanism for collecting multiple threads into a single object and manipulating those threads all at once, rather than individually. For example, you can start or suspend all the threads within a group with a single method call. Java thread groups are implemented by the *ThreadGroup* class in the *java.lang* package.

The runtime system puts a thread into a thread group during thread construction. When you create a thread, you can either allow the runtime system to put the new thread in some reasonable default group or you can explicitly set the new thread's group. The thread is a permanent member of whatever thread group it joins upon its creation--you cannot move a thread to a new group after the thread has been created.

In our system all event generating threads are placed in a generic thread group. The class that groups all threads is presented below:

```
public class EventsGroup
{
    private static ThreadGroup events;
    private Thread intE;
    private Thread extE;
    private Thread dirCom;
    private Thread caract;

    public EventsGroup() {
        events = new ThreadGroup("Events");
        events.setDaemon(true);
        intE = new EventGenerator(events, "InternalEventsGenerator");
        dirCom = new DirComGenerator(events, "DirectCommandsGenerator");
        caract = new CharacterizationGenerator(events, "CharacterizationGenerator");
        try {
            AIBOClient.Instance();
            extE = new SensorEvents(AIBOClient.getClient(), events, "ExternalEvents");
        }
        catch(UninitializedException uie) {
            System.out.println(uie.getMessage());
        }
    }
    public static ThreadGroup getEvents() [...]
    public void start() {
        intE.start();
        dirCom.start();
        caract.start();
        extE.start();
    }
}
```

It is important to explain the reasons that conducted to this approach. By placing all event threads in one group we have the control over all of them with one single command, therefore if a rule-based system thread needs processor time it will cease all event generating threads. On the other hand with this we make sure that only one event is generated at a given moment of time giving time for the rule based systems to reason upon it before another event is thrown.

#### 4.4.2 Java Beans and Jess

JavaBeans is a portable, platform-independent component model written in the Java programming language. It enables developers to write reusable components once and run them anywhere -- benefiting from the platform-independent power of Java technology. JavaBeans acts as a Bridge between proprietary component models and provides a

```
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeSupport;
import java.io.Serializable;

public class ANYBEAN implements Serializable
{
    private String label;
    public String getLabel() {...}
    public void setLabel(String label){...}

    private PropertyChangeSupport pcs = new PropertyChangeSupport(this);
    public void addPropertyChangeListener(PropertyChangeListener pcl)
    {
        pcs.addPropertyChangeListener(pcl);
    }
    public void removePropertyChangeListener(PropertyChangeListener pcl)
    {
        pcs.removePropertyChangeListener(pcl);
    }
}
```

seamless and powerful means for developers to build components that run in ActiveX container applications. JavaBeans, like other kinds of software components (for instance, Visual Basic controls), often serve as interfaces to more complex systems such as databases or special hardware. An example of a generic Java Bean present in our system is presented below: (all beans in the system fulfill this minimal scheme)

Jess ingeniously uses Java Beans to create facts (actually shadow-facts). We used this feature extensively in our system and it is practically the way we assure extensibility.

You can view Jess's working memory as sort of an electronic organizer for your rule-based system. A piece of data must be part of the working memory for it to be used in the premises of a Jess rule. Ordered and unordered facts are useful in many situations, but in many real-world applications, it's useful to have rules respond to things that happen outside of the rule engine. Jess lets you put regular Java objects in working memory—instances of your own classes that can serve as hooks into a larger software system—as long as those objects fulfill the minimal requirements necessary to be JavaBeans. [11]

The similarity between JavaBeans and unordered facts is that both have a list of slots (for JavaBeans, they're called *properties*) containing values that might change over time. There's plenty more to JavaBeans than just properties; however, those features are not taken into consideration here. A JavaBean property is most often a pair of methods named in a standard way. If the property is a `String` named `label`, the Java methods look like this:

A shadow fact has one slot for each Java Bean property. If a Java Bean has array

```
String getLabel();
void setLabel(String);
```

properties, those properties become multi-slots, and all other properties become normal slots. The slots are automatically populated with the values of the Java Bean's properties. If

you want to have a shadow fact continuously track property changes in a Java Bean, Jess needs to be notified whenever a property changes in that Java Bean. The Java Bean can notify Jess by sending it a special kind of Java event, a *java.beans.PropertyChangeEvent*.

The following classes in our system have been encapsulated as Java Beans: *Personality, Mood, Needs, Emotion, InternalEvent, ExternalEvent, Characterization, DirectCommand, Goal, Preference and Standard*. The reason for this is that all these classes when instantiated represent facts in the two rule-based systems used.

### 4.4.3 AIBOExceptions

All exceptions thrown by any function or classes of this system are contained in a global package AIBOExceptions. The implementation for these classes is trivial. An example exception is presented below:

```
package AIBOExceptions;

/**
 * The role of the class is to provide a project specific exception
 *
 * <br>Project: Companion AIBO
 * <br>Package: AIBOExceptions
 * <br>Jar File: AIBOExceptions
 * <br>University: TU Delft, MMI Group
 * <br>Last modified - 5 April 2005.
 * @author Iulia Dobai
 */
public class InvalidArgumentException extends Exception
{
    /**
     * Default Constructor for the exception
     * @param msg - message to be displayed in case exception is thrown.
     */
    public InvalidArgumentException(String msg)
    {
        super (msg);
    }
}
```

### 4.4.4 Functionality of the Rule-Based Systems

This section is dedicated to explaining the way the two rule-based systems work and the way they relate to each other.

The first rule-based system is encapsulated in the thread: *PersThread* that starts a first Rete object. The fact-base and the rule-base for this system lives in the file: *persRules.clp*. The second rule-based system is encapsulated in the thread: *GPSThread* that starts a second Rete object. The fact base and the rule-base of this system lives in the file: *gpsRules.clp*. (please refer to Appendix C)

#### 4.4.4.1 persRules.clp and the first rule-based system

In this system the only facts present are: personality and needs that are created from the respective JavaBeans like this:

```
(defclass personality engines.Personality )
(defclass needs engines.Needs )
...
(bind ?personality (call engines.Personality Instance))
(definstance personality ?personality dynamic)

(bind ?needs (call engines.Needs Instance))
(definstance needs ?needs dynamic)
```

Since both these classes are singletons there is just one fact for each. The rule-base on the other hand is very consistent being made up mainly of rules examining the traits of personality in combination with the values for the different needs.

There are 3 kinds of rules to accomplish this task: rules that examine each trait of personality on its own, rules that examine every combination of two traits of personality and rules that examine the combination of each trait and needs. Each of these rules results in creating facts in the fact-base of the second rule-based system. There are a few functions that encapsulate the routines necessary for this:

```
(deffunction makePreference(?category ?value)
  (call-on-engine
    (get-member engines.GPSThread rete)
    (bind ?pref (new engines.gps.Preference))
    (definstance preference ?pref )
    (call ?pref setValue ?value)
    (call ?pref setCategory ?category)
    (printout t "Generated preference is " ?category " with value "
?value "." crlf)
    ;(facts)
    (reset)
  )
)
```

The same style of function is used to create goals and standards also.

A fourth function takes care of changing values for the mood that is a unique fact in the second rule-based system also:

```
(deffunction changeMood(?value)
  (call-on-engine
    (get-member engines.GPSThread rete)
    (bind ?mood (call engines.Mood Instance))
    (call ?mood setValence ?value)
    (printout t "mood was changed to " ?value "." crlf)
    (reset)
  )
)
```

Except for the rules that take care of assuring a fact-base for the second rule-based system there are is a low-salience rule that turns the control to the second expert system:

```
(defrule nothing
;This is the lowest salience rule. It is called when Personality Engine have finished
firing all active rules
  (declare (salience -200))
  =>
  (runGPS)
)
```

This rule has no LHS therefore the rule is always matched and is fired anytime the salience reaches this low level. The rule calls a function that is exemplified below:

```
(deffunction runGPS()
  (reset)
  (call-on-engine
    (get-member engines.GPSThread rete)
    (printout t "Starting GPS Engine ..." crlf)
    ;(facts)
    (focus GPS)
    (run-until-halt)
  )
)
```

#### 4.4.4.2 gpsRules.clp and the second rule-based system

The second rule-based system is actually the core of the system. It's fact-base comprises of facts of the type: preference, standard, goal, mood, internalEvent, externalEvent, characterization, directCommand, emotion and action.

```
(defclass mood engines.Mood)
(defclass preference engines.gps.Preference )
(defclass intEvent engines.events.InternalEvent)
(defclass extEvent engines.events.ExternalEvent)
(defclass directCom engines.events.DirectCommand)
(defclass caract engines.events.Characterization)
(defclass emotion engines.Emotion)
(defclass standard engines.gps.Standard )
(defclass goal engines.gps.Goal)
```

The mood fact is created like this:

```
(bind ?mood (call engines.Mood Instance))
(definstance mood ?mood dynamic)
```

A global variable stores also the values for the needs fact from the first rule-based system because these values are modified in the course of the execution:

```
(defglobal ?*needs* = nil)
(bind ?*needs* (call engines.Needs Instance))
```

Facts that shadow event objects are created within the Java classes that generate them. An example of such a function in class GPSThread is shown below:

```
public static void makeFact(DirectCommand dc) throws JessException
{
    rete.executeCommand("focus GPS");
    rete.definstance("directCom", dc, false);
}
```

The vast majority of rules in the rule-base evaluate the events that occur on the premises of goals, preferences and standards that are represented by facts in the fact-base. The output of such a rule consists in the following:

- retraction of the fact representing the event that activated the rule
- change values for the mood fact
- output of a emotion or action fact that needs to be interpreted
- change in the values for the needs fact in the first rule-based system

- a call to the first rule based system and resume of its activity.

The trigger for the typical rule consists of the identification of an event, a value for mood and a specific preference, standard or goal.

An example of such a hypothetical rule is given above:

```
(defrule eventNamedRule_no
  (preference(category ?cat&: (? ?cat ?)) (value ?val&: (? ?val ?)))
  ?id<-(intEvent(eventName ?name&: (? ?name ?)))
  ?m<-(mood (valence ?valence&: (? ?valence ?)))
  =>
  ;the event that generated the emotion is retracted because it is no further needed
  (retract ?id)
  ;change mood. Mood decreases with ? units.
  (modify ?m (valence (- ?valence 10)))
  ;AIBO's emotion is generated and the EmotionInterpreter is called
  (intrepretEmotion (get-member engines.Emotion SAD 40))
  ;change needs.
  (call ?*needs* setNeed ?)
  (runPERS)
)
```

Function runPers is shown below:

```
(deffunction runPERS()
  (reset)
  (call-on-engine
    (get-member engines.PersThread rete)
    (printout t "Starting Personality Engine... " crlf)
    (focus PERS)
    (run-until-halt)
  )
)
```

Functions interpretEmotion and/or interpretAction have two very important role in the system. Every time a new emotion is created an interpreter thread for that emotion is activated and run:

```
(deffunction intrepretEmotion(?type ?intensity)
  (bind ?eInterpreter (new interpreters.EmotionInterpreter))
  (call ?eInterpreter makeEmotion ?type ?intensity)
  (call ?eInterpreter start)
)
```

The lowest salience rule ceases the execution of the Rete algorithm until a valid event is being thrown into the system:

```
(defrule waitEvents
  ;This is the rule with the lowest salience...it has to be called after all other
  rules have fired.
  (declare (salience -200))
  =>
  (printout t "GPS Engine is waiting for events to fire ... " crlf)
  (waitEvent)
)
```

The function waitEvent si described below:

```
(deffunction waitEvent()
  (focus GPS)
  ((engine) waitForActivations)
)
```



#### 4.4.5 URBI Scripts

URBI Scripts are tiny pieces of code written in the URBI language in order to control the execution of commands by AIBO. URBI Scripts can be as simple as moving some AIBO joint from a position to a different one or as complex as a walking algorithm. The advantage of the URBI language however, is that it allows writing complex movement sequences in a very simple way making use of *loops* and the *sin* and *cosin* functions.

I will try to illustrate in a few examples some of the main strengths of URBI. In order to assure reusability of scripts in an elegant way I developed a package of classes (*RobotOutput*) that contains a structural organization of all URBI scripts needed by the system. The package comprises of five main classes: Face, Head, Posture, Speak, Tail. Each of these classes is a collection of static parameterized functions that return Strings containing URBI commands.

The following is a java function that returns a string representing a URBI command for making AIBO play through his speakers a barking sound consecutively for x times with a number of y breaks in-between. (e.g. ham-ham ham-ham ham-ham (3 units each with 2 sounds) or ham-ham-ham ham-ham-ham ham-ham-ham ham-ham-ham (4 units each with 3 sounds)). The actual URBI script is highlighted:

```
public static String BARKING(double noBreaks, double noBark) throws
InvalidArgumentException
{
    if(noBreaks<0 || noBreaks>20 || noBark<0 || noBark>20)
    {
        throw new InvalidArgumentException("Parameters have to be in the range 0
and 20");
    }
    else
    {
        String sNoBark = String.valueOf(noBark);
        String sNoBreaks = String.valueOf(noBreaks);
        return "for (j=0; j<" + sNoBreaks + "; j++)" +
            "{" +
            "for(i=0;i<"+ sNoBark +" ;i++) {speaker.play(\"bark.wav\"); wait
200;};" +
            "wait 1000;" +
            "}" +
            "}"
    }
}
```

The following function returns a string for an URBI command that makes AIBO wave his head:

```
public static final String NO()
{
    return "headPan.valn = 0.5 sin:1000 ampli:0.2 timeout:1500";
}
```

This command makes use of the **sin** function.

The following code sniffed represents a command for AIBO to screech his ear with the left foot:

```
SCRECHEAR = "legLF2.valn=0.15& legLF1.valn=0.8& legLF3.valn=0.7 &" +
    Head.POSITION(0.25, 0.95)+ "&"+Head.NECKPOSITION(0.30)+
    ";legLF3.valn = 0.7 sin:1000 ampli:0.1 timeout:3000 ";
```

Code reusability is made obvious here by using functions like *Head.POSITION* and *Head.NECKPOSITION* available in the Head class.

Some more complex movements (like walk and turn) have been provided by the developers of URBI and are incorporated on the memory stick.

The package containing all URBI Scripts that have been developed under this project are presented in Appendix E.




## 5 Evaluation and Tests

*“Even with the best of maps and instruments, we can never fully chart our journeys.”*  
**Gail Pool**

This chapter will present the concrete place in development that was reached on the journey to achieving our initial goals. A model for reasoning with emotions was created and implemented and portrayed on AIBO. Unfortunately not a lot of usability tests have been conducted due to lack of time. The author was hoping that the successful implementation of this model will lead to conclusions regarding a lot of issues raised by cognitive science but unfortunately a lot more questions were raised and sadly enough some remained unanswered. Following we summarize tests that we conducted together with recommended tests taking into consideration the domain of the problem itself: at the crossroad of more than a few complex and emerging sciences: cognitive science, psychiatry, artificial intelligence, robotics.

Two tests have been conducted to study perception of the emotions showed by AIBO and they are presented in sub-chapter: “Emotion Perception on AIBO”. The second subchapter will present some tests that the author thinks should be conducted in order to improve the system itself as well as questions that aroused during the implementation. (Should we put vision, values, other kinds of mood, autonomy, how to test interaction, etc)

### *Chapter Overview:*

-  Results
-  Emotion Perception on AIBO
-  Recommended User Tests

## 5.1 Results

“The reward of a thing well done is to have done it.”

**Ralph Waldo Emerson**

The results that this project reveals are both intrinsic and extrinsic. First a model for emotional cognition was developed for AIBO, the model itself has more than one applications. The development of the model itself raises a lot of questions regarding what should be in a personality model that is valid for robots. On the other hand the model that was developed was encapsulated in a working prototype, while the prototype is pretty fragile and far from being complete it helped proof the concept and also brought some more light on the questions that should be addressed and answered by future research. Thirdly an entire gamma of small pieces emerged that can be reused in various ways in different other applications: like the *RobotOutput* package that is a wrapper around a pretty big group of URBI Scripts. Another extrinsic result of the project is regarding the emotions that AIBO can portray and the best way it can do so. The project also answers at least in part the question regarding the types of interactions we can have with an AIBO robot. Looking from a broader perspective the project shows a few possible new applications of this system: like configuration for elderly-care, entertainment for people with disabilities and so on. Some ideas regarding these issues make the content of this chapter.

### 5.1.1 Emotions and Actions

One of the end results of this project is that AIBO shows emotion in a context dependent way. The following images portray the emotions AIBO is able to show :



The above showed emotions are concurrent with some small actions: like wagging tail, head moving backwards, etc. Each emotion is a sequence of about 5-6 seconds of small movements.

Other small actions have been developed in the course of this project including: scratching ear, stretching, waking up, etc. Movies presenting these actions can be found on the website of the author.

These actions become relevant in the context of the project when the remote brain decide the way to combine them to show different complex actions. The same actions and emotions can be used by any other AIBO application that uses URBI and Java.

### 5.1.2 System flexibility

The plus of this system resides in its flexibility. The system is flexible from a lot of points of view: the robot personality is parameterized, since users can customize it according to their desires, the robots mind is customizable since the rules of the two expert systems that make up the brain can be easily changed or maintained. Hence the few rules that are in the system right now are there just for the purpose of giving examples of what can be achieved. In order to write new rules for the two expert systems, the user has to know what are the circumstances in which AIBO can act. Therefore the developer needs to be aware of some files that contain static variables. These files are present in Appendix B.

## 5.2 Emotion Perception on AIBO

*"Test fast, fail fast, adjust fast."*

**Tom Peters**

In order to test the way the system works a lot of tests would be necessary. Just a few of these tests have been conducted but a lot more are required in order to validate the system. In this section we present a user test developed in order to test the way human perceive the emotions showed by AIBO.

### 5.2.1 Test Description

As explained before AIBO at any given time AIBO will shoe one of the 6 possible emotions that have been implemented. A small test has been conducted in order to test if humans find it easy to grasp AIBO's emotion and interpret it as the desired emotion. The experiment was conducted as follows: AIBO in a neutral sit position showed in turns six emotions labeled Emotion1, Emotion2, etc. It is important to mention that AIBO showed these emotions out of any context. The test taker was asked to label each of the 6 emotions with one of the following: happy, sad, angry, fear, disgust and surprise. Also some space was left out for the user to include ideas and inputs in order to make the labeled emotion more realistic. The user had to fill in 6 labels that looked exactly like the following:

Emotion 1	
Please choose one of the above that best matches the emotion AIBO shows	
1	Disgust
2	Surprise

3	Angriness
4	Sadness
5	Fear
6	Happiness
Please give your comments below:	

Emotions were showed in the following order:

1	Happy
2	Sad
3	Angry
4	Fear
5	Surprise
6	Disgust

### 5.2.2 Test Results

A total number of 15 people were interrogated mainly from the area of computer science. The table above mentions the number of people that recognized each emotion from the total number of 15 that took the test.

Happy	Sad	Angry	Fear	Surprise	Disgust
14	11	14	3	13	9

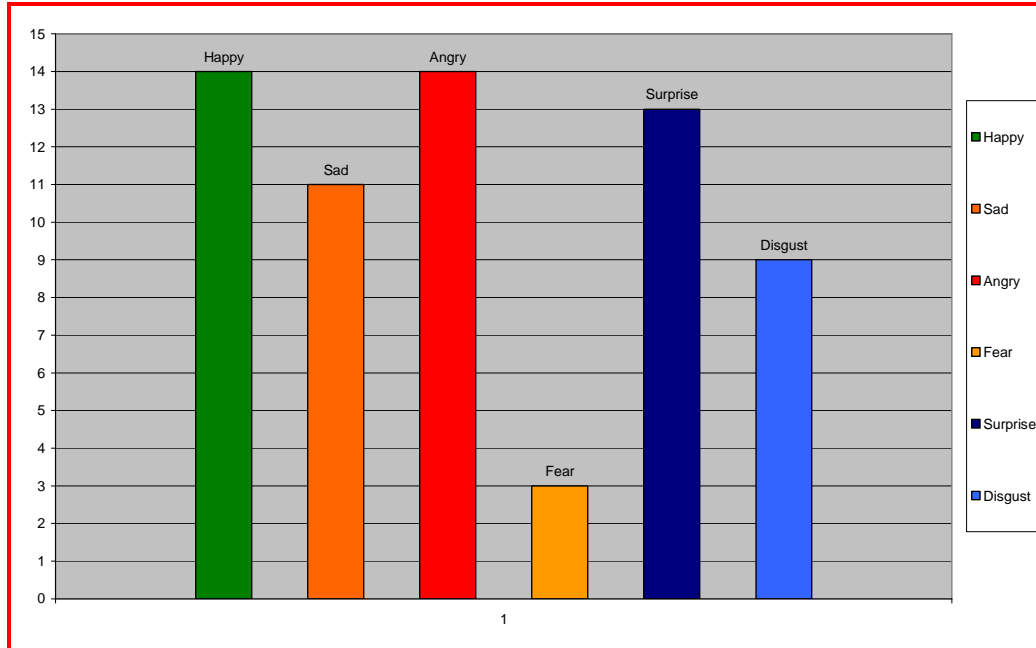


Figure 26: Emotion Perception User Test Chart

As the test proves there are a few emotions that were very clearly identified by the test takers: happiness, anger and surprise. Sadness was identified by more than 2/3 of the test takers and therefore I will consider it realistic enough to be identifiable in a context. On the other hand fear and disgust were wrongly interpreted as other emotions by most test takers and a lot of time confused. Therefore a decision was made to improve the way AIBO shows that emotion in order to become more realistic.

The following table shows the choices test takers made when trying to identify **fear**. Fear was most commonly mistaken by surprise.

Fear	Disgust	Anger	Surprise	Sadness	Happiness
3	2	0	8	0	2

One of the problems concerning the interpretation of this emotion was dependant on the order the emotions were showed. Since **surprise** was presented after **fear** it is very probable that by excluding it people would not have interpret it wrongly. The problem still stays and the “fear” emotion needs to be improved.

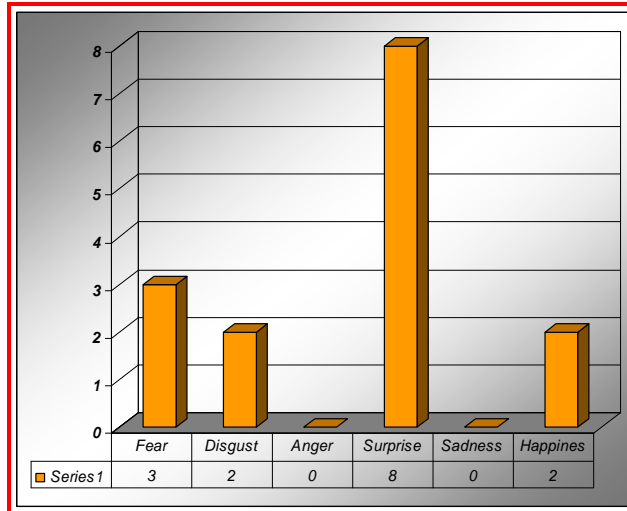


Figure 27: Interpretation of "fear" - Chart

Test subjects trying to interpret emotion **disgust** made the following choices:

Disgust	Fear	Surprise	Angry	Sad	Happy
9	3	1	1	1	0

Disgust was the last emotion to interpret and therefore it should have been very easy to recognize. This justifies the pretty high amount of people that did recognize it correctly. On the other hand more then 1/3 of the test takers were not able to recognize it therefore we consider necessary to improve the way this emotion is displayed.

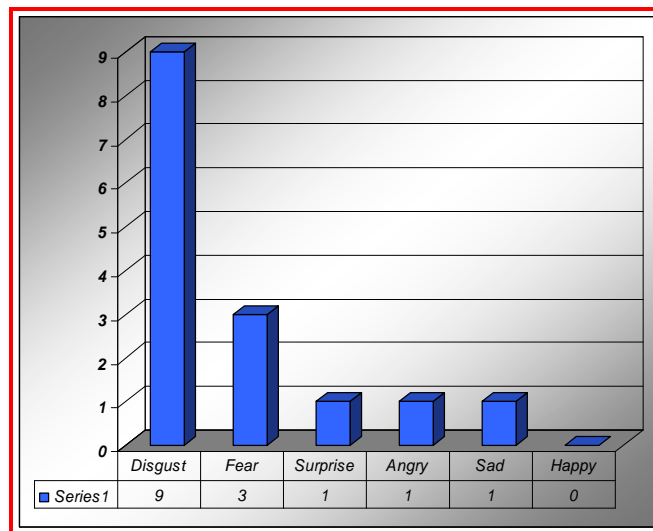


Figure 28: Interpretation of "disgust" - Chart

### 5.2.3 Conclusions

The test that was conducted proved that the emotions that were designed were somewhat distinguishable by humans in a context independent environment. The test proved that emotions like: **happy, sad, angry and surprise** are easy to recognize while **fear and**



**disgust** are not easily perceivable by human actors. Therefore further effort will be put in trying to develop a better way to portray these 2 emotions.

A few problems were indicated in the test itself. First thing that should be taken into account when doing another test like this should be that the order of displaying the emotions matters a lot in the result of the test. Secondly, it would be better to show the test takers either more emotions and ask them to identify the 6 emotions that we are looking for but this kind of test has the disadvantage of having to actually implement those 10-12 choices to choose from. Another alternative would be to display only the 6 emotions that have been implemented but ask the test takers to choose the emotion from a list of 10-12 emotions. A third issue that arises is whether the user should see or not all the possible emotions before taking the test. This also can have an influence in the result of the test.

A second test will be conducted after the two emotions of: fear and disgust will be improved.

### 5.3 Recommended User Test

*“The best thing about the future is that it only comes one day at a time.”*

**Abraham Lincoln**

#### 5.3.1 Psychological Validity

The main test the author feels should be conducted on the system is in order to validate the psychological model developed and proof its realism in the case of robots. While this model is a very simplified model and pretty different from that of a human it resembles in some way human personality model. Therefore psychological tests should be conducted to decide whether the model is valid in the case of robots and/or virtual characters.

I think it is worth mentioning that this is not a trivial test and it might consist of more than one test and I would recommend it to be conducted by somebody with psychological background.

In order to conduct this research the author suggests the following complex method. The entire system has 4 text files configurable by users and that act as parameters for the entire system. Two text files: *traits.txt* and *needs.txt* are simple word files that can be changed by anyone. The other two files: *persRules.clp* and *gpsRules.clp* can be modified only by someone with knowledge in expert system rules and the jess or clips syntax. (The two syntaxes are basically the same. The model that is to be found in the *.clp* files should be followed in writing rules).

The recommended steps for such a test are:

1. Decide upon the rules that make up the inference engines from a psychological point of view. (They should then be translated in jess rules)
2. Start by choosing two or more personalities with significant differences. (ex. A very open and friendly personality). Based on these a decision regarding the absolute values the traits of personality should have is to be made.

3. Design a scenario of a designated time for AIBO. (ex. 4-5 minutes). The scenarios should be made in such a way that some emotional behavior is triggered and personality characteristics are identifiable by humans interacting with AIBO.
4. Start the test by choosing a number of subjects and ask them to interact with AIBO in the given scenario.
5. The questionnaire for the user should include questions to extract their opinion regarding AIBO's personality. The user should not be asked to identify exact values for the 5 traits of personality but rather ask for characterization words for the robot and the interaction with it.
6. The output of the questionnaire should be compared to the traits AIBO was started with. If there is a close enough match than the **psychological validity of the model for robots** was proved.

A trick that can be included is to ask a group of subjects to interact with AIBO and to the questionnaire and ask a separate group of subjects to just watch and observe the interaction and fill in the same questionnaire.

A few questions need to be asked before starting this test: what kind of human interaction is necessary for a human to be able to decide on AIBO's character, how long should the interaction be, how many subjects should the test include, what personalities should be tested, etc.

Please remember this research should be done in order to decide upon the validity of the model in the case of robots, not to proof the correctness of the model.

AIBO can be tested regarding this issue also by taking a standardized personality test. Decisions regarding this should be made by the ones administrating the research.

### 5.3.2 Rule-Base Correctness

The second main question that should be answered regarding this system is weather the rules that make up the central cognitive unit accomplish the desired output. In other words, if we start by trying to develop an AIBO that accomplishes a simple task in a maze like environment than the way this task is accomplished should be tested. On the other hand if the rules that control AIBO's mind (the central processing unit) make him act as a cute dog entertaining humans this aspect should be tested. In order to conduct such a test: decisions should be made regarding the rules that should be contained in the two rule-based systems, then a personality (values for the 5 traits that represent personality in our model) should be chosen considering what personality would be the best that task. Further interaction tests should be conducted with humans in order to decide on the system.

## 6 Conclusion

*“The best way to predict the future is to invent it.”*




**Alan Kay**

When we started this project we wanted to end up with a companion robot-dog acting in different contexts independently and giving emotional response to events that take place. We ended up with a model that allows customization for AIBO to react by conducting small actions and emotions depending on the context.

This chapter will introduce the main results of the project, the reusable components together with possible uses. Later we will proceed in concluding the work that has been done and provide recommendation in case further work will be done on the basis of this system.

I hope this project will prove to be the right combination of ability and effort, lead by the right motivation and done with the right attitude.

### *Chapter Overview:*

-  Conclusions
-  Recommendations
-  Possible applications

## 6.1 Conclusions

*“A man builds a fine house; and now he has a master, and a task for life; he is to furnish, watch, show it, and keep it in repair, the rest of his days.”*

**Ralph Waldo Emerson**

This project tried to accomplish the task of creating a mental model for robots to reason with emotions. As a result of this, a cognitive model was developed, a model of personality that relies on the concepts of: personality as a five dimensional space, mood, needs as in Maslow’s pyramid of needs, and emotions as in the OCC Model. The reasoning is triggered by events that are relevant for the system. This model was then transformed in a working prototype and run on AIBO. In order to test the cognitive module on AIBO some more components were developed and put together in a more complex framework. The result of this study and development process is a system and architecture that runs on AIBO and that is highly flexible and customizable.

The exact same software can be used for AIBO to reason in very different contexts. Therefore there are a few variable parameters in the system: values for the traits of personality, initial values for AIBO’s needs and at a higher development level: the rules upon which AIBO reasons are subject for modification and adaptation to whatever context. Thus, considering that we have a well established set of rules for the reasoning component, AIBO can be transformed in a very shy, closed, not-communicative, lazy robot or in a very active, jovial, cute dog. On the other hand with little adaptation in the rule set of the reasoning, AIBO can be adapted to perform various tasks including searching objects, acting as a rescue dog, a companion for elderly or for people with disabilities and special needs. It is needless to say, there’s lot more that can be improved in this framework including: adding a complex history and memory level, adding a learning mechanism to assure that a robot can be thought by an owner. Further a complex set of concepts can be tested including: creativity, instinct, etc.

On the other hand the mental model that was developed is based on existing models of personality that were tested on virtual characters in games or for chat conversations. This model is in many ways adapted to a human environment where the character (that is applied on) is unpredictable. This model could further be tested with virtual characters in chat conversations or game settings.

Considering now the development process and the software implementation perspective, the system is a complex architecture, over a TCP/IP wireless connection and it makes use of an existing framework: URBI. This eased the development process and assured in many ways reusability of code with other robots and on other platforms, therefore knowledge and development effort will not be lost and hopefully can be reused later with humanoid robots. The current model has to face the disadvantage of not being a step towards an intelligent autonomous robot. According to this system, AIBO is a complex hardware structure (therefore lacking any kind of reasoning or knowledge) that is controlled by an intelligent computer. The focus of this project was in assuring that the computer is in fact “intelligent” emotionally. Since the system is practically on a computer Java was used for the development process with several of its features including threads, exception handling, polymorphism, encapsulation, etc. A special part of the “brain” (the software system that was developed) is made off the central cognitive unit. For this part an artificial intelligence

technique needed to be used. Therefore the brain makes use of rule-based systems (expert-systems). A simple expert system could have been used but the author choused to use two expert systems tightly coupled where one assures the fact-base for the second one. The rules of the rule-based systems are statically added before running the application but in this way assure adaptability to the system.

The field of robotics and artificial intelligence are growing at a very fast pace, when you combine it with human-computer interaction it simply explodes. Right now a lot of companies and research institutions are putting a lot of effort in studying the interaction between humans and all other kinds of robots (for home use, for entertainment, for security situations, etc) and even in studying interactions between robots of the same kind or of different types. In the near future robots will be an active part of our lives, studies have revealed that future robots have to be able to show emotional response to humans and bonds between robots and humans are inevitable. Thus working in this project was a continuous self-rewarding job, triggered by the desire to grasp the way robots will function and respond to human actions in the future.

## 6.2 Recommendations

*“Nothing great was ever achieved without enthusiasm.”*

**Ralph Waldo Emerson**

The major challenge of this project resides in the fact that it is part of a field that is changing so fast that by the end of one project the premises of the project have already changed. Even though some premises of this project have changed already, more are likely to change in the near future. Extra effort was put into assuring a certain level of extensibility and flexibility for this project but some things are simply unpredictable.

Following some recommendations can be made in order to improve the overall system, to extend it to a more complex stage and to better respond to future issues. The recommendations I have for future work rely mostly on weaknesses of the current system. Therefore the first thing that should be addressed is to create the memory/history module that in the implementation of this prototype is missing. This will open space for reasoning modules to be developed on top of the history module. Further the system can be enriched with many more possible actions to be done by AIBO and reasoning rules. This will offer strength and also the possibility of various other user and non-user tests to be conducted. A special effort should then be directed on integrating vision and sound recognition modules in the system. (Right now these are being simulated with 2 user-interfaces). Connecting recognition components with the current system will raise a lot of problems including time synchronization, robot connectivity, etc. As a general recommendation I would suggest using URBI scripts as much as possible to incorporate a lot of functionality. For example: a walking script can be enriched to be a walking script where AIBO is also detecting obstacles and is avoiding them.

The second type of possible improvements relies on further development in the field of robotics. Sooner or later the issues of learning including: unknown-word learning, action learning, feedback-based learning, etc and curiosity need to be addressed by this system.

Some effort has been put into these concepts by others and work can be adapted to run within this system.

At a very abstract level further improvements can be brought also to the personality model that has been developed. Since a lot of things in psychology are not yet known regarding humans and the human mind it is hard to spot the exact concepts that will do the job for robots. Stretching the idea I strongly believe that robots can be a way to discover more about humans themselves in two ways: first of all by doing research for robots we find out new things about the human mind, but on the other hand, close models of the human mind on robots can answer questions that only complicated and unreliable psychological tests have answered until now. In this area it is extremely important to know exactly where to stop and use just as much as we need to accomplish our goals. Overdoing can be destructive.

I would also like to stress the idea of autonomy of AIBO. In this project AIBO is not autonomous; anything that will interrupt the wireless connection with the PC will “kill” also AIBO. Until now we have two major examples of AIBO’s autonomy: AIBO playing soccer and AIBO as an entertainment dog in the way Sony developed it. Even though autonomy is hard to achieve in the context of complex processes that need to be run (in the circumstances of the hardware and software limitations) this is a major challenge. Robots of the future are autonomous and responsible; therefore the wireless connectivity is used better for other situations, communicating with other hardware devices for the use of exchanging knowledge.

### 6.3 Possible applications

*“The sign of an intelligent people is their ability to control emotions by the application of reason.”*

**Marya Mannes**

Although the system was developed as a prototype no direct practical applications resulted. With small configurations and adaptations the system can be developed in a wide range of applications including:

- AIBO offering companionship for elderly
- AIBO offering companionship for people with different disabilities and special needs
- AIBO offering entertainment in family and home environments
- AIBO acting as a watch dog.
- AIBO acting as a motivating factor in lab – work environments.

AIBO can play the role of a mediator and game server for a game developed specially for people working in the same environment in order to increase productivity and performance. AIBO is cute and funny, and it has an incredibly big power to become adorable, basing our idea on these observations I think we can stimulate people in work environments. A game can be invented to measure and keep productivity and AIBO can act as an intermediate and server for the game since it has the mobility and the capacity to act as one.

- AIBO acting as a home character in intelligent home environments.

Lots of research is being conducted nowadays to develop intelligent home environments where all electronic devices communicate and synchronise each other. AIBO can act as

intermediate level in-between such as intelligent system and humans since AIBO is proven to have a good level of possible interactions with humans. There has been a continuous conflict whether we need or not a virtual character to welcome us and follow us through our journey in an intelligent home where: TV, PC, PDA, fridge, microwave, washing machine, etc communicate and act intelligently. I strongly believe AIBO can play the role of such a character (and the current model can be adapted to act as one) and further tests could be developed in order to decide if such a character is necessary or not.

- AIBO acting as a rescue dog in simulated environments

Even though due to its size and current physical limitations AIBO cannot act as a rescue dog in real situations, research can be conducted in simulated environments to prepare a model for the future when humanoid robots will be able to operate rescue missions.

- AIBO as a game or play situations

AIBO can have a great entertainment value in home environments and a new series of games can be adapted or invented to transform AIBO in a play-mate. Currently we have seen AIBO play X-and-O in an experiment conducted by another university. There's no reason to believe AIBO will be able to play many other games in the future, maybe even chess. The future of gaming industry is currently evolving, and lots of accent is on games that combine virtual reality and physical reality. I strongly believe AIBO can be a nice interface between the two.





## 7 Bibliography

- [1] Baillie, J.C., "URBI: A UNIVERSAL LANGUAGE FOR ROBOTIC CONTROL", International Journal of Humanoid Robotics, 2004, World Scientific Publishing Company
- [2] Baillie, J.C., "URBI Language Specification", v. 1.0, 2005.
- [3] Baillie, J.C., "Grounding symbols in Perception with two interacting Autonomous Robots"
- [4] Bartneck, C., "Integrating the OCC Model of Emotions in Embodied Characters", *Workshop on Virtual Conversational Characters*, 2002
- [5] Beaumont, R., L., "Five Factor Constellations and Popular Personality Types", *Psychology* 106, 2003
- [6] Costa, P.T., McCrae, R.R., "Normal personality assessment in clinical practice: The NEO personality inventory". *Psychological Assessment*, 1992
- [7] Dobai, I., Rothkrantz, L., van der Mast, C. "Personality Model for a Companion AIBO", *ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, Broadway New York, June 2005, pp. 438-441.
- [8] Eckel, B., "Thinking in Java", Prentice-Hall Inc, 2000, 2<sup>nd</sup> edition.
- [9] Ekman, P., Friesen, W.V., "The argument and Evidence About Universals in Facial Expressions of Emotion". In *Handbook of Social Psychophysiology*. New York: John Wiley and Sons, Ltd, 1989
- [10] Fogg, B.J., "Persuasive Technology. Using Computers to change what we think and do", Morgan Kaufmann Publishers, 2003, p.23-29.
- [11] Friedman-Hill, E., "Jess in Action. Rule-Based Systems in Java", Manning Publications Co., 2003
- [12] Fujita M., "On Activating Human Communication With Pet-Type Robot AIBO", *Proceedings of the IEEE*, Vol.92, NO.11, November 2004
- [13] Fujita, M, Kageyama, K, "An open architecture for Robot Entertainment", In *Proceedings of the First International Conference on Autonomous Agents*, ACM Press, 1997, pp. 435-442.
- [14] Gamma, E., Helm, R., Johnson, R., Vlissides, J., "Elements of reusable Object-Orientated Software", Design Patterns CD
- [15] Hopgood, A. A., "Intelligent Systems for Engineers and Scientists", CRC Press LLC, 2001
- [16] Kaplan, F, "Talking AIBO: First Experiment of Verbal Interactions with an Autonomous Four-legged Robot", In *Proceedings of the CELE-Twente workshop on interacting agents*.
- [17] Kaplan, F, Oudeyer, P.Y., "Maximizing learning progress: an internal reward system for development"
- [18] Kaplan, F, Oudeyer, P.Y., "Motivational principles for visual know-how development"
- [19] Kaplan, F, Hafner, V.V., "The challenges of joint attention"
- [20] Ksirsagar, S and N. Magnenat-Thalmann, "A Multilayer Personality Model", In: *Proceedings of 2nd International Symposium on Smart Graphics*, 2002
- [21] Lang, P., J., "The Emotion Probe: studies of motivation and attention", A study in the Neuroscience of Love and Hate. Hillside, NJ: Lawrence Erlbaum Associates, Publishers, 1995.
- [22] Larman, C., "Applying UML and Patterns"
- [23] Lee, C.H.A., "Emotional AIBO", Internal Report, 2004.
- [24] Maslow, A.H., *Motivation and Personality*, 2nd. ed., New York, Harper & Row, 1970
- [25] Ortony, A., Clore, G.L., Collins, A., *The Cognitive Structure of Emotions*, Cambridge University Press, 1988
- [26] Oudeyer, P.Y., Kaplan, F, Hafner, V.V., Whyte, A, "The playground experiment: Task-Independent Development of Curious Robot", In the *Proceedings of the AAAI Spring Symposium on Developmental Robotics*, 2005
- [27] Shalloway, A., Trott, J, "Design Patterns Explained"
- [28] Silva, R.D., Siebra, A.C., Valadares, L.J, Almeida, A.L., Frery, C.A., Falcao, J.R., Ramalho, L.G., "A synthetic actor model for Long-Term Computer Games".
- [29] <http://www.sony.net/Products/aibo/> - Sony AIBO Global web site.
- [30] <http://www.eu.aibo.com/> - Sony AIBO European web site
- [31] [https://openr.aibo.com/openr/eng/perm/main\\_menu.php4](https://openr.aibo.com/openr/eng/perm/main_menu.php4) - Sony AIBO development tools
- [32] <http://www.urbiforge.com/eng/index.html> URBI official web page.

- [33] <http://cogrob.ensta.fr/>
- [34] <http://www.csl.sony.fr/Research/Topics/DevelopmentalRobotics/index.html>
- [35] <http://www-2.cs.cmu.edu/~tekkotsu/>
- [36] <http://mmi.tudelft.nl/~iulia/aibo/> – Author’s web site on AIBO
- [37] <http://www.mmi.tudelft.nl/~siska/aibo/index.php> - MMI AIBO Team
- [38] <http://aibo.cs.uu.nl/> - Dutch AIBO Team

Note: All Java API files for the entire system are to be found on the CD accompanying this report and on the author’s web-site.

## Appendix A: AIBO from a hardware perspective

AIBO presents itself as a complex piece of hardware enriched by sensors and actuators. Following you will find a description of various components that make up AIBO:

- **CPU** 64-bit RISC Processor
- **CPU clock speed** 576 MHz
- **RAM** 64 MB
- **Program media**
  - Dedicated AIBO robot "Memory Stick™" media
- **Moveable parts** (Total 20 degrees of freedom)
  - Head - 3 DOF
  - Mouth - 1 DOF
  - Legs - 3 DOF x 4 (legs)
  - Ears - 1 DOF x 2 (ears)
  - Tail - 2 DOF
- **Input section**
  - Charging contacts
- **Setting switches**
  - Volume control switch
  - Wireless LAN switch
- **Image input** 350.000-pixel CMOS image sensor
- **Audio input** Stereo microphones
- **Audio output** Speaker 20.8mm, 500mW
- **Integrated sensors**
  - Infrared distance sensors x 2
  - Acceleration sensor
  - Vibration sensor
- **Input sensors**
  - Head sensor
  - Back sensor
  - Chin sensor
  - Paw sensors (\* 4)
- **Power consumption** Approx. 7 W (in standard mode)
- **Operating time** Approx. 1,5 hours (with fully charged ERA-7B1, in standard mode)
- **Dimensions** Approx. 180 (w) x 278 (h) x 319 (d) mm
- **Weight** Approx. 1.65 kg (including battery pack and "Memory Stick™" media)
- **Wireless LAN function** Wireless LAN module (Wi-Fi certified) Internal standard compatibility: IEEE 802.11b/IEEE 802.11 Frequency band: 2,4 GHz  
Wireless channels: 1 – 11 Modulation : DS-SS (IEEE 802.11 – compliant)  
Encryption : WEP 64 (40 bits), WEP 128 (104 bits)

The following two pictures (Figure 2 and 3) present the position of sensors and actuators on AIBO from a front view and a rear view.

Figure 29: AIBO Sensors and Actuators - Front View

► Features-front

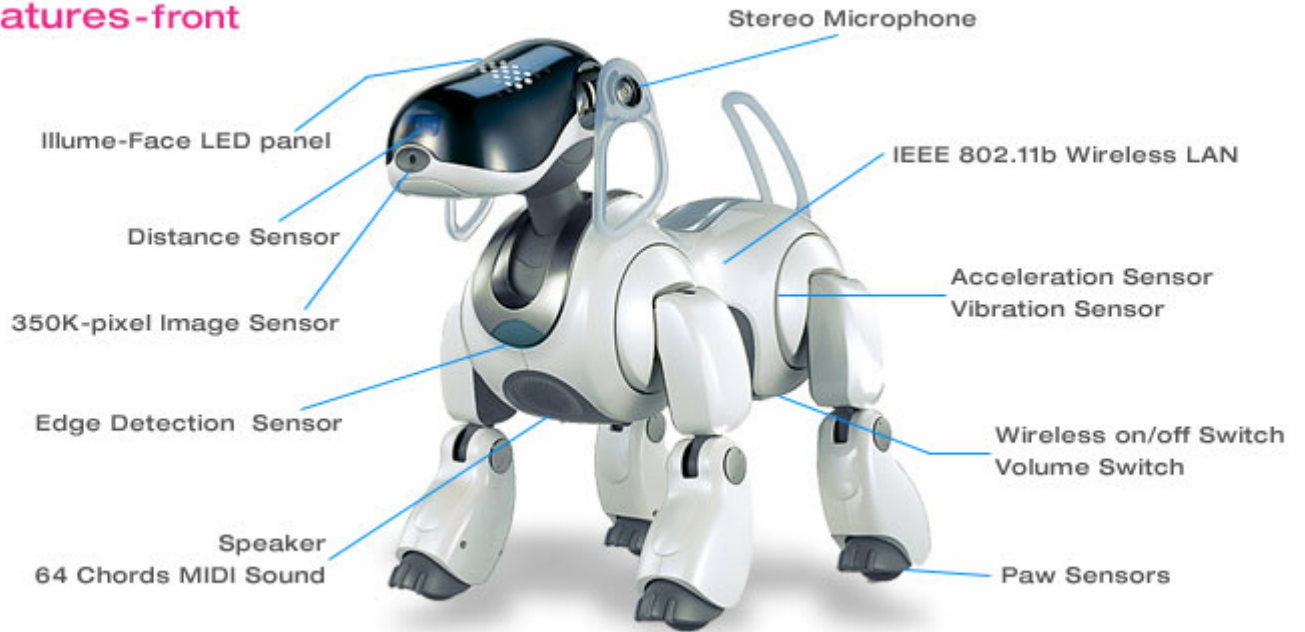
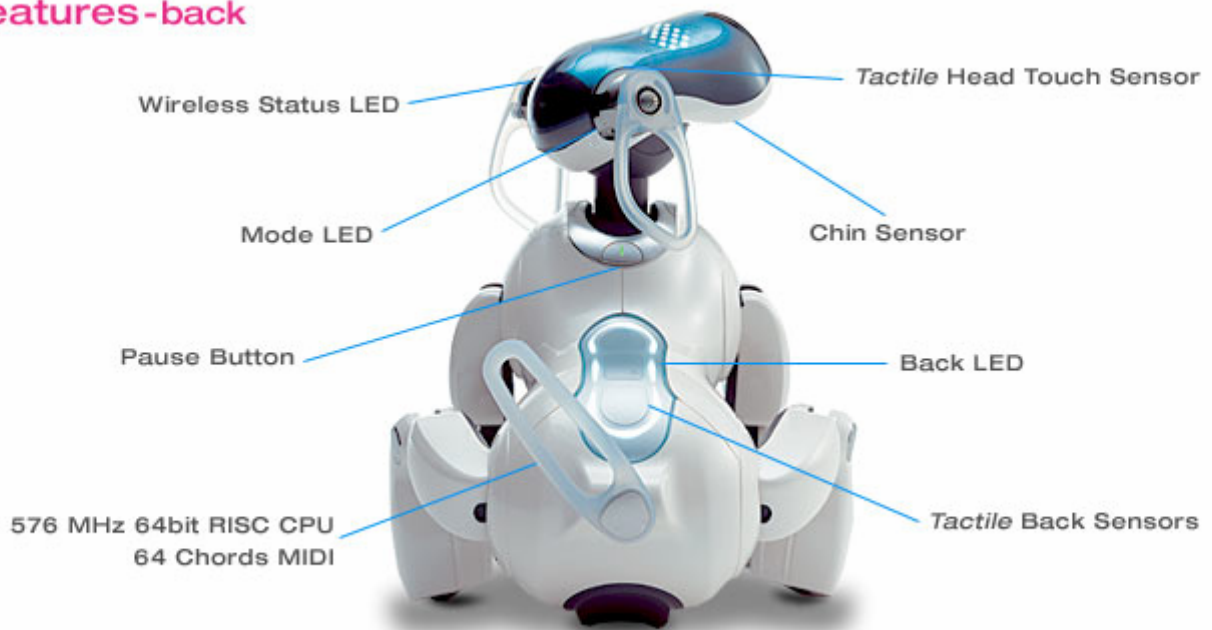


Figure 30: AIBO Sensors and Actuators - Back View

► Features-back



## Appendix B: Implementation

```

package engines.events;
import java.io.Serializable;
import AIBOExceptions.*;

/**
 * Project: Companion AIBO
 * @author Iulia Dobai
 * Apr 14, 2005
 */
public class InternalEvent extends Event implements Serializable
{
    /**
     * possible event generators are the battery and the internal timer of aibo
     */
    public static final int BATTERY = 1;
    public static final int TIMER = 2;

    /**
     * possible internal event names
     */
    public static final int NOEVENT = 0;

    public static final int MORNING = 1;
    public static final int EVENING = 2;
    public static final int LUNCH = 4;
    public static final int THISYEAR = 5;

    /**
     * This field practically represents the device that generated the internalEvent
     * For a beginning the generators will be just the Battery and the Internal Timer
     */
    private int eventGenerator;

    public synchronized int getEventGenerator() [...]

    public synchronized void setEventGenerator(int eventGenerator)
    {
        int tmp = this.eventGenerator;
        this.eventGenerator = eventGenerator;
        getPcs().firePropertyChange("evtGen", tmp, eventGenerator);
    }

    public String toString() [...]

    public InternalEvent(int eventGenerator, int name) throws InvalidEventGenerator,
    InvalidEventName
    {
        super(name);
        if(eventGenerator <1 || eventGenerator >2)
        {
            throw new InvalidEventGenerator("invalidEventGenerator");
        }
        else
        {
            this.eventGenerator = eventGenerator;
        }
    }
}

```

```

package engines.events;
import java.io.Serializable;
import engines.gps.*;
import AIBOExceptions.InvalidEventGenerator;
import AIBOExceptions.InvalidEventName;

/**
 * Project: Companion AIBO
 * @author Iulia Dobai
 * Apr 14, 2005
 */
public class ExternalEvent extends Event implements Serializable
{
    /**
     * possible external event names are Actions defined for Standards
     */
    /**
     * possible event generators are the battery and the internal timer of aibo
     */
    public static final int AUDIO = 1;
    public static final int VIDEO = 2;
    public static final int TOUCH = 3;
    public static final int DISTANCE = 4;
    public static final int ACCELERATION = 5;

    private int eventGenerator;

    public ExternalEvent(int name, int eventGenerator) throws
    InvalidEventGenerator, InvalidEventName
    {
        super(name);
        if(eventGenerator >=1 && eventGenerator <=5)
        {
            this.eventGenerator = eventGenerator;
        }
        else
        {
            throw new InvalidEventGenerator("invalidEventGenerator");
        }
    }

    /**
     * @return Returns the eventGenerator.
     */
    public int getEventGenerator()
    {
        return eventGenerator;
    }

    /**
     * @param eventGenerator The eventGenerator to set.
     */
    public void setEventGenerator(int eventGenerator)
    {
        int tmp = this.eventGenerator;
        this.eventGenerator = eventGenerator;
        getPcs().firePropertyChange("evtGen", tmp, eventGenerator);
    }

    public String toString() [...]
}

```

```

/*
 * Last Revision May 19, 2005
 */
package engines.events;
import java.io.Serializable;
import AIBOExceptions.InvalidEventName;
/**
 * Project: Companion AIBO
 * @author Iulia Dobai
 * May 19, 2005
 */
public class Characterization extends Event implements Serializable
{
    /**
     * possible characterization for AIBO
     */
    public static final int STUPIDDOG = 103;
    public static final int SMARTDOG = 104;
    public static final int GOODDOG = 101;
    public static final int BADDOG = 102;

    public Characterization(int name) throws InvalidEventName
    {
        super(name);
    }

    public String toString() [...]
}

```

```

package engines.gps;
public class PreferenceCategories
{
    public static final int ALLHUMANS = 10;
    public static final int OWNER = 11;
    public static final int HUMANFRIENDS = 12;
    public static final int HUMANENEMIES = 13;
    public static final int VICTIMS = 14;
    public static final int UNKNOWNHUMAN = 15;

    public static final int ALLAIBO = 20;
    public static final int AIBOMATE = 21;
    public static final int AIBOFRIENDS = 22;
    public static final int AIBOENEMIES = 23;
    public static final int UNKNOWNNAIBO = 24;

    public static final int ALLTOYS = 30;
    public static final int AIBOBONE = 31;
    public static final int AIBOBALL = 32;
    public static final int UNKNOWNTOY = 33;
    public static final int WEAPON = 34;

    public static final int SELF = 1;

    public static final int NOCATEGORY = 0;
}

```

```
package engines.gps;
public class Action
{
    public static final int TOUCH = 10;
    public static final int PETTING = 11;
    public static final int HITBACK = 12;
    public static final int TOUCHCHIN = 13;
    public static final int TOUCHHEAD = 14;
    public static final int TOUCHPAWLF = 15;
    public static final int TOUCHPAWLB= 16;
    public static final int TOUCHPAWRF = 17;
    public static final int TOUCHPAWRB = 18;
    public static final int TOUCHBACK = 19;

    public static final int HITBALL = 20;
    public static final int MISSBALL = 21;
    public static final int STEALBALL = 22;
    public static final int PLAYBALL = 23;

    public static final int PICKBONE = 30;
    public static final int MISSBONE = 31;
    public static final int STEALBONE = 32;
    public static final int PLAYBONE = 33;

    public static final int SPEAK = 40;
    public static final int YELL = 41;
    public static final int WISPER =42;

    public static final int PLAY = 50;
    public static final int HELP = 60;
    public static final int HARM = 70;

    public static final int GOODDOG = 81;
    public static final int BADDOG = 82;
    public static final int SMARTDOG = 83;
    public static final int STUPIDDOG = 84;

    public static final int NOACTION = 0;
}
```



```
package engines.events;
import java.io.Serializable;
import AIBOExceptions.*;

/**
 * Project: Companion AIBO
 * @author Iulia Dobai
 * Apr 14, 2005
 */

public class DirectCommand extends Event implements Serializable
{
    /**
     * possible direct commands event names
     */
    //play
    public static final int PLAY = 5;
    //Ball
    public static final int PLAYBALL = 20;
    public static final int FINDBALL = 3;
    public static final int BRINGBALL = 15;
    //Bone
    public static final int PLAYBONE = 19;
    public static final int BRINGBONE = 14;
    public static final int FINDBONE = 4;

    //follow
    public static final int FOLLOWME = 6;
    public static final int FOLLOWAIBO = 7;
    public static final int FOLLOWHUMAN = 8;
    public static final int GOAWAY = 9;

    //sound
    public static final int SHUTUP = 10;
    public static final int BARK = 16;
    public static final int SING = 1;

    //movements
    public static final int WALK = 11;
    public static final int CROWL = 12;
    public static final int TURN = 13;

    public static final int DANCE = 2;

    //fight
    public static final int DEFENDME = 17;
    public static final int JUMPENEMY = 18;

    public DirectCommand(int name) throws InvalidEventName
    {
        super(name);
    }
    public String toString() [...]
}
}
```

## Appendix C: expert system files

Please take these files as example rule files for the two expert systems. These files can be configured and adapted to reason in different contexts and circumstances:

### ***persRules.clp***

```
(defmodule PERS)
(defclass personality engines.Personality )
(defclass needs engines.Needs )

(deffacts idle-fact
  (idle))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;Adding the 2 facts
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(bind ?personality (call engines.Personality Instance))
(definstance personality ?personality dynamic)

(bind ?needs (call engines.Needs Instance))
(definstance needs ?needs dynamic)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;Functions
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(deffunction runGPS()
  (reset)
  (call-on-engine
    (get-member engines.GPSThread rete)
    (printout t "Starting GPS Engine ..." crlf)
    ;(facts)
    (focus GPS)
    (run-until-halt)
  )
)

(deffunction makePreference(?category ?value)
  (call-on-engine
    (get-member engines.GPSThread rete)
    (bind ?pref (new engines.gps.Preference))
    (definstance preference ?pref )
    (call ?pref setValue ?value)
    (call ?pref setCategory ?category)
    (printout t "Generated preference is " ?category " with value " ?value
  ." crlf)
    ;(facts)
    (reset)
  )
)

(deffunction changeMood(?value)
  (call-on-engine
    (get-member engines.GPSThread rete)
    (bind ?mood (call engines.Mood Instance))
    (call ?mood setValence ?value)
    (printout t "mood was changed to " ?value "." crlf)
    (reset)
  )
)

(deffunction makeStandard(?action ?agent ?val)
  (call-on-engine
    (get-member engines.GPSThread rete)
    (bind ?stand (new engines.gps.Standard))
    (definstance standard ?stand )
  )
)
```

```

        (call ?stand setAction ?action)
        (call ?stand setAgent ?agent)
            (call ?stand setValue ?val)
            (printout t "AIBO approves/dissapproves " ?action " done by other agent"
?agent " with value " ?val "." crlf)
        (reset)
    )
)

(defun makeGoal(?category ?value)
  (call-on-engine
    (get-member engines.GPSThread rete)
    (bind ?goal (new engines.gps.Goal))
    (definstance goal ?goal )
    (call ?goal setValue ?value)
    (call ?goal setCategory ?category)
    (printout t "Generated goal is " ?category " with value " ?value ".")
  crlf)
  (reset)
)

)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;Rules that take into consideration one trait of personality
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(facts)
(defrule HighOpeness
  ; IF OPENESS is high (>=70)
  (personality(OPENESS ?open&: (>= ?open 70)))
  =>
  (printout t "creative, imaginative, philosophical" crlf)
  (makePreference (get-member engines.gps.PreferenceCategories UNKNOWNTYOY) 100)
  (makePreference (get-member engines.gps.PreferenceCategories UNKNOWNHUMAN) 80)
  (makePreference (get-member engines.gps.PreferenceCategories ALLTOYS) 80)
  (makeStandard (get-member engines.gps.Action PLAY) (get-member
engines.gps.PreferenceCategories ALLHUMANS) 90)
  (makeStandard (get-member engines.gps.Action PLAY) (get-member
engines.gps.PreferenceCategories ALLAIBO) 90)
  (undefrule HighOpeness)
)

(defrule LowOpeness
  ; IF OPENESS is low (<30)
  (personality(OPENESS ?open&: (< ?open 30)))
  =>
  (printout t "uncreative, unintelectual, unintelligent" crlf)
  (makePreference (get-member engines.gps.PreferenceCategories UNKNOWNHUMAN) 20)
  (makePreference (get-member engines.gps.PreferenceCategories UNKNOWNTYOY) 0)
  (makeStandard (get-member engines.gps.Action PLAY) (get-member
engines.gps.PreferenceCategories ALLHUMANS) 20)
  (makeStandard (get-member engines.gps.Action PLAY) (get-member
engines.gps.PreferenceCategories ALLAIBO) 20)
  (undefrule LowOpeness)
)

)

;(defrule t03
  ; IF OPENESS is medium-low (>=30 and <50)
  ; (personality (OPENESS ?open&: (and (>= ?open 30) (< ?open 50))))
  ; =>
  ;)

;(defrule t04
  ; IF OPENESS is medium-high (>=50 and <70)
  ; (personality (OPENESS ?open&: (and (>= ?open 50) (< ?open 70))))
  ; =>
  ;)

(defrule HighConscientiousness
  ; IF CONSCIENTIOUSNESS is high (>=70)
  (personality(CONSCIENTIOUSNESS ?con&: (>= ?con 70)))
  =>

```

```

    (printout t "through, steady, consistent" crlf)
    (undefrule HighConscientiousness)
  )

(defrule LowConscientiousness
  ; IF CONSCIENTIOUSNESS is low (<30)
  (personality(CONSCIENTIOUSNESS ?com&: (< ?com 30)))
  =>
  (printout t "inconsistent, scatterbrained, unstable" crlf)
  (undefrule LowConscientiousness)
  ;more related to goals
)

;(defrule tC3
  ; IF CONSCIENTIOUSNESS is medium-low (>=30 and <50)
  ; (personality (CONSCIENTIOUSNESS ?con&: (and (>= ?con 30) (< ?con 50))))
  ; =>
  ;nothing
  ;more related to goals
;)

;(defrule tC4
  ; IF CONSCIENTIOUSNESS is medium-high (>=50 and <70)
  ; (personality (CONSCIENTIOUSNESS ?con&: (and (>= ?con 50) (< ?con 70))))
  ; =>
  ;nothing
  ;more related to goals
;)

(defrule HighExtraversion
  ; IF EXTRAVERSION is high (>=70)
  ;High on Extraversion means that we have an AIBO high in need for stimulation
  (personality(EXTRAVERSION ?extra&: (>= ?extra 70)))
  ?needs <-(needs(love ?love)(belonging ?belonging) (achievement ?achievement)
(recognition ?recognition ))
  =>
  (printout t "talkative, extraverted, aggressive" crlf)
  ;make preferences
  (makePreference (get-member engines.gps.PreferenceCategories AIBOFRIENDS) 100)
  (makePreference (get-member engines.gps.PreferenceCategories HUMANFRIENDS) 100)
  (makePreference (get-member engines.gps.PreferenceCategories ALLAIBO) 80)
  (makePreference (get-member engines.gps.PreferenceCategories ALLHUMANS) 80)
  (makePreference (get-member engines.gps.PreferenceCategories HUMANENEMIES) 60)
  (makePreference (get-member engines.gps.PreferenceCategories AIBOENEMIES) 60)
  (makePreference (get-member engines.gps.PreferenceCategories OWNER) 100)
  ;change need so that need for belonging and love is bigger
  (modify ?needs (love (+ ?love 20)))
  (modify ?needs (belonging (+ ?belonging 20)))
  (modify ?needs (achievement (+ ?achievement 20)))
  (modify ?needs (recognition (+ ?recognition 20)))
  ;make standards - AIBO aproves on ALLHUMANS to TOUCH, SPEAK, PLAY
  (makeStandard (get-member engines.gps.Action TOUCHHEAD) (get-member
engines.gps.PreferenceCategories ALLHUMANS) 100)
  (makeStandard (get-member engines.gps.Action TOUCHCHIN) (get-member
engines.gps.PreferenceCategories ALLHUMANS) 100)
  (makeStandard (get-member engines.gps.Action PETTING) (get-member
engines.gps.PreferenceCategories ALLHUMANS) 100)
  (makeStandard (get-member engines.gps.Action SPEAK) (get-member
engines.gps.PreferenceCategories ALLHUMANS) 100)
  (makeStandard (get-member engines.gps.Action HITBACK) (get-member
engines.gps.PreferenceCategories ALLHUMANS) 0)
  (makeStandard (get-member engines.gps.Action SPEAK) (get-member
engines.gps.PreferenceCategories SELF) 100)
  (makeStandard (get-member engines.gps.Action SPEAK) (get-member
engines.gps.PreferenceCategories ALLAIBO) 100)
  (makeStandard (get-member engines.gps.Action PLAY) (get-member
engines.gps.PreferenceCategories SELF) 100)
  ;modify mood - switch to a positive mood
  (changeMood 50)
  (undefrule HighExtraversion)
)

```

```

(defrule LowExtraversion
  ; IF EXTRAVERSION is low (<30)
  (personality(EXTRAVERSION ?extra&: (< ?extra 30)))
  ?needs <-(needs(love ?love)(belonging ?belonging)(achievement ?achievement)(recognition
?recognition) )
  =>
  (printout t "shy, quiet, introverted" crlf)
  ;make preferences
  (makePreference (get-member engines.gps.PreferenceCategories AIBOFRIENDS) 30)
  (makePreference (get-member engines.gps.PreferenceCategories HUMANFRIENDS) 30)
  ;change need so that need for belonging and love is bigger
  (modify ?needs (love (- ?love 20)))
  (modify ?needs (belonging (- ?belonging 20)))
  (modify ?needs (recognition(+ ?recognition 20)))
  ;make standards - AIBO down'st like talking,
  (makeStandard (get-member engines.gps.Action SPEAK) (get-member
engines.gps.PreferenceCategories SELF) 0)
  (makeStandard (get-member engines.gps.Action SPEAK) (get-member
engines.gps.PreferenceCategories ALLAIBO) 0)
  (makeStandard (get-member engines.gps.Action SPEAK) (get-member
engines.gps.PreferenceCategories ALLHUMANS) 30)
  ;modify mood - switch to a positive mood
  (changeMood 10)
  (undefrule LowExtraversion)
)

(defrule MediumLowExtraversion
  ; IF EXTRAVERSION is medium-low (>=30 and <50)
  ;Medium-low extraversion changes need for love and belonging (decrease) and for
recognition (decrease)
  (personality (EXTRAVERSION ?extra&: (and (>= ?extra 30) (< ?extra 50))))
  ?needs <-(needs(love ?love)(belonging ?belonging)(achievement ?achievement)(recognition
?recognition) )
  =>
  (makePreference (get-member engines.gps.PreferenceCategories ALLAIBO) 30)
  (makePreference (get-member engines.gps.PreferenceCategories ALLHUMANS) 30)
  (makePreference (get-member engines.gps.PreferenceCategories OWNER) 50)
  (makePreference (get-member engines.gps.PreferenceCategories AIBOFRIENDS) 40)
  (makePreference (get-member engines.gps.PreferenceCategories HUMANFRIENDS) 40)
  (modify ?needs (love (- ?love 10)))
  (modify ?needs (belonging (- ?belonging 10)))
  (modify ?needs (recognition(+ ?recognition 10)))
  ;make standards - AIBO down'st like talking,
  (makeStandard (get-member engines.gps.Action SPEAK) (get-member
engines.gps.PreferenceCategories SELF) 30)
  (makeStandard (get-member engines.gps.Action SPEAK) (get-member
engines.gps.PreferenceCategories ALLAIBO) 30)
  (makeStandard (get-member engines.gps.Action SPEAK) (get-member
engines.gps.PreferenceCategories ALLHUMANS) 50)
  (undefrule MediumLowExtraversion)
)

(defrule MediumHighExtraversion
  ; IF EXTRAVERSION is medium-high (>=50 and <70)
  ;Medium-high on Extraversion means that we have an AIBO high in need for stimulation
  (personality (EXTRAVERSION ?extra&: (and (>= ?extra 50) (< ?extra 70))))
  ?needs <-(needs(love ?love)(belonging ?belonging)(achievement ?achievement)(recognition
?recognition) )
  =>
  (makePreference (get-member engines.gps.PreferenceCategories AIBOFRIENDS) 60)
  (makePreference (get-member engines.gps.PreferenceCategories HUMANFRIENDS) 60)
  (makePreference (get-member engines.gps.PreferenceCategories OWNER) 70)
  (makePreference (get-member engines.gps.PreferenceCategories HUMANENEMIES) 50)
  (makePreference (get-member engines.gps.PreferenceCategories AIBOENEMIES) 50)
  ;change need so that need for belonging and love is bigger
  (modify ?needs (love (+ ?love 10)))
  (modify ?needs (belonging (+ ?belonging 10)))
  (modify ?needs (achievement (+ ?achievement 10)))
  (modify ?needs (recognition (+ ?recognition 10)))
  ;make standards - AIBO aproves on ALLHUMANS to TOUCH, SPEAK, PLAY

```

```

    (makeStandard (get-member engines.gps.Action TOUCHHEAD) (get-member
engines.gps.PreferenceCategories ALLHUMANS) 70)
    (makeStandard (get-member engines.gps.Action TOUCHCHIN) (get-member
engines.gps.PreferenceCategories ALLHUMANS) 70)
    (makeStandard (get-member engines.gps.Action PETTING) (get-member
engines.gps.PreferenceCategories ALLHUMANS) 70)
    (makeStandard (get-member engines.gps.Action SPEAK) (get-member
engines.gps.PreferenceCategories ALLHUMANS) 70)
    (makeStandard (get-member engines.gps.Action HITBACK) (get-member
engines.gps.PreferenceCategories ALLHUMANS) 0)
    (makeStandard (get-member engines.gps.Action SPEAK) (get-member
engines.gps.PreferenceCategories SELF) 70)
    (makeStandard (get-member engines.gps.Action SPEAK) (get-member
engines.gps.PreferenceCategories ALLAIBO) 70)
    (makeStandard (get-member engines.gps.Action PLAY) (get-member
engines.gps.PreferenceCategories SELF) 70)
    ;modify mood - switch to a positive mood
    (changeMood 25)
    (undefrule MediumHighExtraversion)
)

(defrule HighAgreeableness
; IF AGREEABLENESS is high (>=70)
(personality(AGREEABLENESS ?agr&: (>= ?agr 70)))
?needs <-(needs(achievement ?achievement))
=>
(printout t "sympathetic, kind, warm" crlf)
;preferences
    (makePreference (get-member engines.gps.PreferenceCategories ALLAIBO) 80)
    (makePreference (get-member engines.gps.PreferenceCategories ALLHUMANS) 80)
    (makePreference (get-member engines.gps.PreferenceCategories ALLTOYS) 80)
    (makePreference (get-member engines.gps.PreferenceCategories VICTIMS) 100)
    ;need for helping others, achievement is big = Symphaty
    (modify ?needs (achievement (+ ?achievement 30)))
;standards
;altruism - doing things for others is self-fulfilment. They like.
    (makeStandard (get-member engines.gps.Action HELP) (get-member
engines.gps.PreferenceCategories ALLHUMANS) 100)
    (makeStandard (get-member engines.gps.Action HELP) (get-member
engines.gps.PreferenceCategories ALLAIBO) 100)
    (undefrule HighAgreeableness)
)

(defrule LowAgreeableness
; IF AGREEABLENESS is low (<30)
(personality(AGREEABLENESS ?agr&: (< ?agr 30)))
?needs <-(needs(achievement ?achievement))
=>
(printout t "unsympathetic, unkind, harsh" crlf)
;preferences
    (makePreference (get-member engines.gps.PreferenceCategories ALLAIBO) 40)
    (makePreference (get-member engines.gps.PreferenceCategories ALLHUMANS) 40)
    (makePreference (get-member engines.gps.PreferenceCategories SELF) 100)
    (makePreference (get-member engines.gps.PreferenceCategories VICTIMS) 0)
;needs
    (modify ?needs (achievement (- ?achievement 30)))
;I don't know what to say about standards
    (undefrule LowAgreeableness)
)

(defrule MediumLowAgreeableness
; IF AGREEABLENESS is medium-low (>=30 and <50)
(personality (AGREEABLENESS ?agr&: (and (>= ?agr 30) (< ?agr 50))))
?needs <-(needs(achievement ?achievement))
=>
;preferences
    (makePreference (get-member engines.gps.PreferenceCategories ALLAIBO) 60)
    (makePreference (get-member engines.gps.PreferenceCategories ALLHUMANS) 60)
    (makePreference (get-member engines.gps.PreferenceCategories SELF) 80)
    (makePreference (get-member engines.gps.PreferenceCategories VICTIMS) 20)
;needs

```

```

(modify ?needs (achievement (- ?achievement 10)))
(undefrule MediumLowAgreeableness)
)

(defrule MediumHighAgreeableness
; IF AGREEABLENESS is medium-high (>=50 and <70)
(personality (AGREEABLENESS ?agr&: (and (>= ?agr 50) (< ?agr 70))))
?needs <-(needs(achievement ?achievement))
=>
;preferences
(makePreference (get-member engines.gps.PreferenceCategories ALLAIBO) 60)
(makePreference (get-member engines.gps.PreferenceCategories ALLHUMANS) 60)
(makePreference (get-member engines.gps.PreferenceCategories SELF) 60)
(makePreference (get-member engines.gps.PreferenceCategories VICTIMS) 80)
;need for helping others, achievement is big = Symphaty
(modify ?needs (achievement (+ ?achievement 10)))
;standards
(makeStandard (get-member engines.gps.Action HELP) (get-member
engines.gps.PreferenceCategories ALLHUMANS) 70)
(makeStandard (get-member engines.gps.Action HELP) (get-member
engines.gps.PreferenceCategories ALLAIBO) 70)
(undefrule MediumHighAgreeableness)
)

(defrule HighNeuroticism
; IF NEUROTICISM is high (>=70)
(personality (NEUROTICISM ?neur&: (>= ?neur 70)))
?needs <-(needs(safety ?safety)(recognition ?recognition))
=>
(printout t "moody, jealous, possessive" crlf)
;preferences for dangerous situations is very low
(makePreference (get-member engines.gps.PreferenceCategories AIBOENEMIES) 60)
(makePreference (get-member engines.gps.PreferenceCategories HUMANENEMIES) 60)
(makePreference (get-member engines.gps.PreferenceCategories WEAPON) 60)
(makePreference (get-member engines.gps.PreferenceCategories ALLHUMANS) 40)
;modify mood - switch to a negative mood
(changeMood -50)
(modify ?needs (safety 70))
(modify ?needs (recognition 60))
(makeStandard (get-member engines.gps.Action HELP) (get-member
engines.gps.PreferenceCategories SELF) 70)
(undefrule HighNeuroticism)
)

(defrule LowNeuroticism
; IF NEUROTICISM is low (<30)
(personality (NEUROTICISM ?neur&: (< ?neur 30)))
?needs <-(needs(safety ?safety)(recognition ?recognition))
=>
(printout t "unenvious" crlf)
(makePreference (get-member engines.gps.PreferenceCategories AIBOENEMIES) 80)
(makePreference (get-member engines.gps.PreferenceCategories HUMANENEMIES) 80)
(makePreference (get-member engines.gps.PreferenceCategories WEAPON) 80)
;modify mood - switch to a negative mood
(changeMood -10)
(modify ?needs (safety 20))
(modify ?needs (recognition 40))
(undefrule LowNeuroticism)
)

;(defrule tN3
; IF NEUROTICISM is medium-low (>=30 and <50)
; (personality (NEUROTICISM ?neur&: (and (>= ?neur 30) (< ?neur 50))))
; =>
;nothing
;)

;(defrule tN4
; IF NEUROTICISM is medium-high (>=50 and <70)
; (personality (NEUROTICISM ?neur&: (and (>= ?neur 50) (< ?neur 70))))

```

```

;    =>
;        ;nothing
;)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;Rules that take into consideration two traits of personality
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule HExtraversionHAgreeableness
  (declare (salience -50))
  (personality(EXTRAVERSION ?extra&: (>= ?extra 70)))
  (personality(AGREEABLENESS ?agr&: (>= ?agr 50)))
  =>
  (printout t "socialble, social, enthusiastic" crlf)
  (undefrule HExtraversionHAgreeableness)
)

(defrule HExtraversionLAgreeableness
  (declare (salience -50))
  (personality(EXTRAVERSION ?extra&: (>= ?extra 70)))
  (personality(AGREEABLENESS ?agr&: (< ?agr 50)))
  =>
  (printout t "dominant, domineering, forcefull" crlf)
  (undefrule HExtraversionLAgreeableness)
)

(defrule HExtraversionHConscientiousness
  (declare (salience -50))
  (personality(EXTRAVERSION ?extra&: (>= ?extra 70)))
  (personality(CONSCIENTIOUSNESS ?con&: (>= ?con 50)))
  =>
  (printout t "active, competitive, persistent" crlf)
  (undefrule HExtraversionHConscientiousness)
)

(defrule HExtraversionHOpeness
  (declare (salience -50))
  (personality(EXTRAVERSION ?extra&: (>= ?extra 70)))
  (personality(OPENESS ?open&: (>= ?open 50)))
  =>
  (printout t "expressive, adventurous, dramatic" crlf)
  (undefrule HExtraversionHOpeness)
)

(defrule LExtraversionLAgreeableness
  (declare (salience -50))
  (personality(EXTRAVERSION ?extra&: (<= ?extra 30)))
  (personality(AGREEABLENESS ?agr&: (< ?agr 50)))
  =>
  (printout t "unsociable, uncommunicative, seclusive" crlf)
  (undefrule LExtraversionLAgreeableness)
)

(defrule LExtraversionHNeuroticism
  (declare (salience -50))
  (personality(EXTRAVERSION ?extra&: (<= ?extra 30)))
  (personality(NEUROTICISM ?neur&: (< ?neur 50)))
  =>
  (printout t "lonely, weak, cowardly" crlf)
  (undefrule LExtraversionHNeuroticism)
)

(defrule LExtraversionLOpeness
  (declare (salience -50))
  (personality(EXTRAVERSION ?extra&: (<= ?extra 30)))
  (personality(OPENESS ?open&: (< ?open 50)))
  =>
  (printout t "passive, meek, dull" crlf)
  (undefrule LExtraversionLOpeness)
)

(defrule HAgreeablenessHExtraversion

```



```

(declare (saliency -50))
  (personality(AGREEABLENESS ?agr&: (>= ?agr 70)))
  (personality(EXTRAVERSION ?extr&: (>= ?extr 50)))
=>
(printout t "merry, cheerfull, happy" crlf)
(undefrule HAgreablenessHExtraversion)
)

(defrule HAgreablenessHConscientiousness
  (declare (saliency -50))
  (personality(AGREEABLENESS ?agr&: (>= ?agr 70)))
  (personality(CONSCIENTIOUSNESS ?con&: (>= ?con 50)))
=>
  (printout t "helpful, cooperative, considerate" crlf)
  (undefrule HAgreablenessHConscientiousness)
)

(defrule HAgreablenessHNeuroticism
  (declare (saliency -50))
  (personality(AGREEABLENESS ?agr&: (>= ?agr 70)))
  (personality(NEUROTICISM ?neur&: (>= ?neur 50)))
=>
  (printout t "sentimental, affectionate, sensitive" crlf)
  (undefrule HAgreablenessHNeuroticism)
)

(defrule HAgreablenessLConscientiousness
  (declare (saliency -50))
  (personality(AGREEABLENESS ?agr&: (>= ?agr 70)))
  (personality(CONSCIENTIOUSNESS ?con&: (< ?con 50)))
=>
  (printout t "inconsiderate, rude, impolite" crlf)
  (undefrule HAgreablenessLConscientiousness)
)

(defrule HAgreablenessLNeuroticism
  (declare (saliency -50))
  (personality(AGREEABLENESS ?agr&: (>= ?agr 70)))
  (personality(NEUROTICISM ?neur&: (< ?neur 50)))
=>
  (printout t "trustfull, pleasant, tolerant" crlf)
  (undefrule HAgreablenessLNeuroticism)
)

(defrule LAgreablenessLNeuroticism
  (declare (saliency -50))
  (personality(AGREEABLENESS ?agr&: (<= ?agr 30)))
  (personality(NEUROTICISM ?neur&: (< ?neur 50)))
=>
  (printout t "insensitive, unaffectionate, passionless" crlf)
  (undefrule LAgreablenessLNeuroticism)
)

(defrule LAgreablenessHNeuroticism
  (declare (saliency -50))
  (personality(AGREEABLENESS ?agr&: (<= ?agr 30)))
  (personality(NEUROTICISM ?neur&: (> ?neur 50)))
=>
  (printout t "demanding, selfish, ill-tempered" crlf)
  (undefrule LAgreablenessHNeuroticism)
)

(defrule HConscientiousnessHExtraversion
  (declare (saliency -50))
  (personality(CONSCIENTIOUSNESS ?con&: (>= ?con 70)))
  (personality(EXTRAVERSION ?extra&: (>= ?extra 50)))
=>
  (printout t "alert, ambitious, firm" crlf)
  (undefrule HConscientiousnessHExtraversion)
)

```

```
(defrule HConscientiousnessHAgreeableness
  (declare (salience -50))
  (personality(CONSCIENTIOUSNESS ?con&: (>= ?con 70)))
  (personality(AGREEABLENESS ?agr&: (>= ?agr 50)))
  =>
  (printout t "responsible, dependable, reliable" crlf)
  (undefrule HConscientiousnessHAgreeableness )
)

(defrule HConscientiousnessLNeuroticism
  (declare (salience -50))
  (personality(CONSCIENTIOUSNESS ?con&: (>= ?con 70)))
  (personality(NEUROTICISM ?neur&: (< ?neur 50)))
  =>
  (printout t "through, steady, consistent" crlf)
  (undefrule HConscientiousnessLNeuroticism)
)

(defrule HConscientiousnessLExtraversion
  (declare (salience -50))
  (personality(CONSCIENTIOUSNESS ?con&: (<= ?con 30)))
  (personality(EXTRAVERSION ?extr&: (< ?extr 50)))
  =>
  (printout t "inefficient, lazy, indecisive" crlf)
  (undefrule HConscientiousnessLExtraversion )
)

(defrule LConscientiousnessHNeuroticism
  (declare (salience -50))
  (personality(CONSCIENTIOUSNESS ?con&: (< ?con 30)))
  (personality(NEUROTICISM ?neur&: (> ?neur 50)))
  =>
  (printout t "inconsistent, scatterbreined, unstable" crlf)
  (undefrule LConscientiousnessHNeuroticism)
)

(defrule LConscientiousnessLOpeness
  (declare (salience -50))
  (personality(CONSCIENTIOUSNESS ?con&: (<= ?con 30)))
  (personality(OPENESS ?open&: (< ?open 50)))
  =>
  (printout t "haphazard, illogical, immature" crlf)
  (undefrule LConscientiousnessLOpeness)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;Rules that relate to personality and needs together
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule HighExtraversionLove1
  (declare (salience -100))
  ;High on extraversion and need for love is small...then I need to increase the need for
  love
  (personality(EXTRAVERSION ?extra&: (>= ?extra 70)))
  ?needs <-(needs(love ?love&: (< ?love 50))(belonging ?belonging ))
  =>
  (modify ?needs (love (+ ?love 20)))
  (modify ?needs (belonging (+ ?belonging 20)))
)

(defrule MediumHighExtraversionLove2
  (declare (salience -100))
  ;Medium-High on extraversion and need for love is small...then I need to increase the
  need for love
  (personality(EXTRAVERSION ?extra&: (and (< ?extra 70)(>= ?extra 50))))
  ?needs <-(needs(love ?love&: (< ?love 50)))
  =>
  (modify ?needs (love (+ ?love 10)))
  (modify ?needs (belonging (+ ?belonging 10)))
)

```

```

(defrule HighExtraversionBelonging1
  (declare (saliency -100))
  ;High on extraversion and need for belonging is small...then I need to increase the
  need for belonging
  (personality(EXTRAVERSION ?extra&: (>= ?extra 70)))
  ?needs <-(needs(belonging ?belonging&: (< ?belonging 50)))
  =>
  (modify ?needs (love (+ ?love 20)))
  (modify ?needs (belonging (+ ?belonging 20)))
)

(defrule MediumHighExtraversionBelonging2
  (declare (saliency -100))
  ;Medium-High on extraversion and need for belonging is small...then I need to
  increase the need for belonging
  (personality(EXTRAVERSION ?extra&: (and (< ?extra 70) (>= ?extra 50))))
  ?needs <-(needs(belonging ?belonging&: (< ?belonging 50)))
  =>
  (modify ?needs (love (+ ?love 10)))
  (modify ?needs (belonging (+ ?belonging 10)))
)

(defrule HAgreeablenessHAchievement
  (declare (saliency -100))
  (personality (AGREEABLENESS ?agr&: (>= ?agr 70)))
  ?needs <-(needs(achievement ?ach&: (< ?ach 30))(recognition ?rec&: (< ?rec 50)))
  =>
  (printout t "immodesty, arrogant" crlf)
)

(defrule HighNeuroticismNSafety
  (declare (saliency -100))
  ; IF NEUROTICISM is high (>=70)
  (personality(NEUROTICISM ?neur&: (>= ?neur 70)))
  ?needs <-(needs(safety ?safety&: (> ?safety 50)))
  =>
  (printout t "AIBO is afraid even in normal situations" crlf)
  (makePreference (get-member engines.gps.PreferenceCategories AIBOENEMIES) 0)
  (makePreference (get-member engines.gps.PreferenceCategories HUMANENEMIES) 0)
  (makePreference (get-member engines.gps.PreferenceCategories WEAPON) 0)
  (changeMood -20)
  (makeStandard (get-member engines.gps.Action HELP) (get-member
engines.gps.PreferenceCategories SELF) 100)
)

(defrule highNeedRecognition
  (declare (saliency -120))
  ?needs <-(needs(recognition ?recognition&: (> ?recognition 50)))
  =>
  (makeStandard (get-member engines.events.Characterization GOODDOG)(get-member
engines.gps.PreferenceCategories OWNER) ?recognition)
  (makeStandard (get-member engines.events.Characterization SMARTDOG) (get-member
engines.gps.PreferenceCategories OWNER) ?recognition)
)

(defrule highNeedAchievement
  (declare (saliency -120))
  ?needs <-(needs(achievement ?achievement&: (> ?achievement 50)))
  =>
  (makeStandard (get-member engines.events.Characterization GOODDOG) (get-member
engines.gps.PreferenceCategories OWNER) ?achievement)
  (makeStandard (get-member engines.events.Characterization SMARTDOG) (get-member
engines.gps.PreferenceCategories OWNER) ?achievement)
)

(defrule always
  (declare (saliency 100))
  =>
  (makeStandard (get-member engines.events.Characterization BADDOG) (get-member
engines.gps.PreferenceCategories OWNER) 100)
)

```

```

    (makeStandard (get-member engines.events.Characterization STUPIDDOG) (get-member
engines.gps.PreferenceCategories OWNER) 100)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;This rule is necessary to give back the handle to the GPS engine
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule nothing
    ;This is the lowest salience rule. It is called when Personality Engine have
finished firing all active rules
    (declare (salience -200))
        =>
        (runGPS)
)

```

### ***gpsRules.clp***

```

(clear)
(defmodule GPS)
(defclass mood engines.Mood)
(defclass preference engines.gps.Preference )
(defclass intEvent engines.events.InternalEvent)
(defclass extEvent engines.events.ExternalEvent)
(defclass directCom engines.events.DirectCommand)
(defclass caract engines.events.Characterization)
(defclass emotion engines.Emotion)
(defclass standard engines.gps.Standard )
(defclass goal engines.gps.Goal)

(deffacts idle-fact
    (idle))

(defglobal ?*needs* = nil)
(bind ?*needs* (call engines.Needs Instance))

(defquery emotionSearch
    (emotion (type ?t) (intensity ?i))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;Adding the "mood" fact
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(bind ?mood (call engines.Mood Instance))
(definstance mood ?mood dynamic)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;Functions
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(deffunction makeEmotion (?type ?intensity)
    (bind ?emotion (new engines.Emotion))
    (definstance emotion ?emotion )
    (call ?emotion setType ?type)
    (call ?emotion setIntensity ?intensity)
    (printout t "Generated emotions is " ?type " with intensity " ?intensity "." crlf)
    (return ?emotion)
)

(deffunction intrepretEmotion(?type ?intensity)
    (bind ?eInterpreter (new interpreters.EmotionInterpreter))
    (call ?eInterpreter makeEmotion ?type ?intensity)
    (call ?eInterpreter start)
)

(deffunction intrepretAction(?name)
    (bind ?aInterpreter (new interpreters.ReActionInterpreter))
    (call ?aInterpreter makeReAction ?name)
    (call ?aInterpreter start)
)

```

```

)

(deffunction runPERS()
  (reset)
  (call-on-engine
    (get-member engines.PersThread rete)
    (printout t "Starting Personality Engine... " crlf)
    (focus PERS)
    (run-until-halt)
  )
)

(deffunction waitEvent()
  (focus GPS)
  ((engine) waitForActivations)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;Rules that evaluate the preferences upon occurrence of events
;;All rules that change the needs need to call the PERSONALITY ENGINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;This is how a rule in this section is structured. Please follow this structure
closely!!!
;(defrule eventNamedRule_no
  ;(preference(category ?cat&: (? ?cat ?)) (value ?val&: (? ?val ?)))
  ;?id<-(intEvent(eventName ?name&: (? ?name ?)))
  ;?m<-(mood (valence ?valence&: (? ?valence ?)))
  ;=>
  ;the event that generated the emotion is retracted because it is no further needed
  ;(retract ?id)
  ;change mood. Mood decreases with ? units.
  ;(modify ?m (valence (- ?valence 10)))
  ;AIBO's emotion is generated and the EmotionInterpreter is called
  ;(intrepretEmotion (get-member engines.Emotion SAD 40))
  ;change needs. setNeed -is the name of a setter in the needs class and ? should be
replaced with a value for that need
  ;(call ?*needs* setNeed ?)
  ;(printout t "If you want to print out something do it here" crlf)
  ;Call the Personality Engine...since change in needs might effect the GPS.
  ;(runPERS)
;)
;;;;;;;;;;;;;;;;;END OF COMMENT representing an example of a rule in this section

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;INTERNAL EVENTS;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule boredomRule1
  ;event that fires this rule is internal event: NOEVENT
  ;preference towards ALLHUMANS is bigger then 50% and mood is positive
  (preference(category ?cat&: (and (>= ?cat 10) (< ?cat 20))) (value ?val&: (> ?val
50)))
  ?id<-(intEvent(eventName ?name&: (eq ?name 0)))
  ?mood<-(mood (valence ?valence&: (> ?valence 0)))
  =>
  (retract ?id)
  (modify ?mood (valence (- ?valence 10)))
  (intrepretEmotion (get-member engines.Emotion SAD) ?val)
  (call ?*needs* setLove 20)
  (printout t "boredomRule1" crlf)
  (runPERS)
)

(defrule boredomRule2
  ;event that fires this rule is internal event: NOEVENT
  ;preference towards ALLHUMANS is bigger then 50% and mood is negative
  (preference(category ?cat&: (and (>= ?cat 10) (< ?cat 20))) (value ?val&: (> ?val
50)))
  ?id<-(intEvent(eventName ?name&: (eq ?name 0)))
  ?mood <-(mood (valence ?valence&: (<= ?valence 0)))
  =>

```

```

(retract ?id)
  (modify ?mood (valence (- ?valence 20)))
  (interpretEmotion (get-member engines.Emotion SAD) 100)
  (call ?*needs* setLove 100)
  (printout t "boredomRule2" crlf)
  (runPERS)
)

(defrule boredomRule3
  ;event that fires this rule is internal event: NOEVENT
  ;preference towards ALLHUMANS is less then 50% and mood is negative
  (preference(category ?cat&: (and (>= ?cat 10) (< ?cat 20))) (value ?val&: (< ?val
50)))
  ?id<-(intEvent(eventName ?name&: (eq ?name 0)))
  ?mood<-(mood (valence ?valence&: (< ?valence 0)))
  =>
  (retract ?id)
  (modify ?mood (valence (+ ?valence 10)))
  (printout t "boredomRule3" crlf)
  (runPERS)
)

(defrule boredomRule4
  ;event that fires this rule is internal event: NOEVENT
  ;preference towards ALLHUMANS is less then 50% and mood is positive
  (preference(category ?cat&: (and (>= ?cat 10) (< ?cat 20))) (value ?val&: (< ?val
50)))
  ?id<-(intEvent(eventName ?name&: (eq ?name 3)))
  ?mood<-(mood (valence ?valence&: (> ?valence 0)))
  =>
  (retract ?id)
  (printout t "boredomRule4" crlf)
  (runPERS)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;DIRECT COMMANDS EVENTS;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule findBone1
  ?dc<-(directCom(eventName ?name&: (eq ?name (get-member engines.gps.DirectCommand
FINDBONE))))
  ?mood<-(mood (valence ?valence&: (> ?valence 0)))
  =>
  (printout t "findBone1 command was caught" crlf)
  (retract ?dc)
  (modify ?mood (valence (+ ?valence 10)))
  (interpretEmotion (get-member engines.Emotion FEAR) 60)
  (call ?*needs* setLove 40)
  (runPERS)
)

(defrule bringBone1
  ?dc<-(directCom(eventName ?name&: (eq ?name (get-member engines.gps.DirectCommand
BRINGBONE))))
  ?mood<-(mood (valence ?valence&: (> ?valence 0)))
  =>
  (printout t "bringBone1 command was caught" crlf)
  (retract ?dc)
  (modify ?mood (valence (+ ?valence 10)))
  (interpretEmotion (get-member engines.Emotion FEAR) 60)
  (call ?*needs* setLove 40)
  (runPERS)
)

(defrule playBone1
  ?dc<-(directCom(eventName ?name&: (eq ?name (get-member engines.gps.DirectCommand
PLAYBONE))))
  ?mood<-(mood (valence ?valence&: (> ?valence 0)))
  =>
  (printout t "playBone1 command event was caught" crlf)
  (retract ?dc)

```

```

(modify ?mood (valence (+ ?valence 10)))
  (intrepretEmotion (get-member engines.Emotion FEAR) 60)
  (call ?*needs* setLove 40)
  (runPERS)
)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;CHARACTERIZATION EVENTS;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule testCaractGOODDOG1
  ?caract<-(caract(eventName ?name&: (eq ?name 101)))
  ?mood<-(mood (valence ?valence&: (> ?valence 0)))
  =>
  (printout t "GOODDOG event was caught" crlf)
  (retract ?caract)
  (modify ?mood (valence (+ ?valence 20)))
  (intrepretAction (call companionAIBO.Head NO))
  (intrepretAction (call companionAIBO.Head NO))
  (intrepretEmotion (get-member engines.Emotion HAPPY) (+ ?valence 50))
  (call ?*needs* setRecognition (+ (call ?*needs* getRecognition) 20))
  (runPERS)
)

(defrule testCaractGOODDOG2
  ?caract<-(caract(eventName ?name&: (eq ?name 101)))
  ?mood<-(mood (valence ?valence&: (< ?valence 0)))
  =>
  (printout t "GOODDOG event was caught" crlf)
  (retract ?caract)
  (modify ?mood (valence (+ ?valence 10)))
  (intrepretEmotion (get-member engines.Emotion HAPPY) (+ ?valence 50))
  (call ?*needs* setRecognition (+ (call ?*needs* getRecognition) 20))
  (runPERS)
)

(defrule testCaractBADDOG1
  ?caract<-(caract(eventName ?name&: (eq ?name 102)))
  ?mood<-(mood (valence ?valence&: (< ?valence 0)))
  =>
  (printout t "BADDOG1 event was caught" crlf)
  (retract ?caract)
  (modify ?mood (valence (- ?valence 10)))
  (intrepretEmotion (get-member engines.Emotion ANGRY) 80)
  (call ?*needs* setRecognition (+ (call ?*needs* getRecognition) 20))
  (runPERS)
)

(defrule testCaractBADDOG2
  ?caract<-(caract(eventName ?name&: (eq ?name 102)))
  ?mood<-(mood (valence ?valence&: (>= ?valence 0)))
  =>
  (printout t "BADDOG2 event was caught" crlf)
  (retract ?caract)
  (modify ?mood (valence (- ?valence 10)))
  (intrepretEmotion (get-member engines.Emotion SURPRISE) (+ ?valence 50))
  (call ?*needs* setRecognition (+ (call ?*needs* getRecognition) 20))
  (runPERS)
)

(defrule testCaractSTUPIDDOG1
  ?caract<-(caract(eventName ?name&: (eq ?name 103)))
  ?mood<-(mood (valence ?valence&: (< ?valence 0)))
  =>
  (printout t "STUPIDDOG1 event was caught" crlf)
  (retract ?caract)
  (modify ?mood (valence (- ?valence 20)))
  (intrepretEmotion (get-member engines.Emotion SURPRISE) 100)
  (intrepretAction (call companionAIBO.Head NO))
  (intrepretEmotion (get-member engines.Emotion SAD) 60)
  (call ?*needs* setRecognition (+ (call ?*needs* getRecognition) 40))
  (runPERS)
)

```

```

)

(defrule testCaractSTUPIDDOG2
  ?caract<-(caract(eventName ?name&: (eq ?name 103)))
  ?mood<-(mood (valence ?valence&: (>= ?valence 0)))
  =>
  (printout t "STUPIDDOG2 event was caught" crlf)
  (retract ?caract)
  (modify ?mood (valence (- ?valence 20)))
  (intrepretEmotion (get-member engines.Emotion SURPRISE) 100)
  (intrepretAction (call companionAIBO.Head NO))
  ;(intrepretEmotion (get-member engines.Emotion SAD) (- 50 ?valence))
  (intrepretEmotion (get-member engines.Emotion SAD) 40)
  (call ?*needs* setRecognition (+ (call ?*needs* getRecognition) 40))
  (runPERS)
)

(defrule testCaractSMARTDOG1
  ?caract<-(caract(eventName ?name&: (eq ?name 104)))
  ?mood<-(mood (valence ?valence&: (> ?valence 0)))
  =>
  (printout t "SMARTDOG event was caught" crlf)
  (retract ?caract)
  ;(modify ?m (valence (+ ?valence 10)))
  (intrepretAction (call companionAIBO.Head YES))
  (intrepretEmotion (get-member engines.Emotion HAPPY) 100)
  (call ?*needs* setRecognition (+ (call ?*needs* getRecognition) 20))
  (call ?*needs* setAchievement (+ (call ?*needs* getAchievement) 10))
  (runPERS)
)

(defrule testCaractSMARTDOG2
  ?caract<-(caract(eventName ?name&: (eq ?name 104)))
  ?mood<-(mood (valence ?valence&: (< ?valence 0)))
  =>
  (printout t "SMARTDOG event was caught" crlf)
  (retract ?caract)
  ;(modify ?m (valence (+ ?valence 10)))
  (intrepretAction (call companionAIBO.Head YES))
  (intrepretEmotion (get-member engines.Emotion SURPRISE) (- 50 ?valence))
  (call ?*needs* setRecognition (+ (call ?*needs* getRecognition) 20))
  (call ?*needs* setAchievement (+ (call ?*needs* getAchievement) 10))
  (runPERS)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;EXTERNAL EVENTS;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule headTouch1
  ?extE<-(extEvent(eventName ?n&: (eq ?n (get-member engines.gps.Action TOUCHHEAD))))
  ;(preference(category ?cat&: (eq ?cat 1)) (value ?val&: (< ?val 50)))
  ?m<-(mood (valence ?valence&: (> ?valence 0)))
  =>
  (printout t "HeadTouch1 event was caught" crlf)
  (retract ?extE)
  (intrepretEmotion (get-member engines.Emotion HAPPY) ?val)
  (modify ?m (valence (+ ?valence 5)))
  (call ?*needs* setBelonging (- (call ?*needs* getBelonging) 10))
  (runPERS)
)

(defrule headTouch2
  ?extE<-(extEvent(eventName ?n&: (eq ?n (get-member engines.gps.Action TOUCHHEAD))))
  ;(preference(category ?cat&: (eq ?cat 1)) (value ?val&: (< ?val 50)))
  ?m<-(mood (valence ?valence&: (< ?valence 0)))
  =>
  (printout t "HeadTouch2 event was caught" crlf)
  (retract ?extE)
  (intrepretEmotion (get-member engines.Emotion ANGRY) 60)
  (modify ?m (valence (- ?valence 10)))
)

```



```

    (call ?*needs* setBelonging (- (call ?*needs* getBelonging) 10))
    (runPERS)
  )

(defrule headTouch3
  ?extE<-(extEvent(eventName ?n&: (eq ?n (get-member engines.gps.Action TOUCHHEAD))))
  (preference(category 1|4|5|6|15) (value ?val&: (< ?val 50)))
  ?m<-(mood (valence ?valence&: (> ?valence 30)))
  =>
  (printout t "HeadTouch3 event was caught" crlf)
  (retract ?extE)
  (interpretEmotion(get-member engines.Emotion HAPPY) 60)
  (modify ?m (valence (- ?valence 10)))
  (call ?*needs* setBelonging (- (call ?*needs* getBelonging) 10))
  (runPERS)
)

(defrule headTouch4
  ?extE<-(extEvent(eventName ?n&: (eq ?n (get-member engines.gps.Action TOUCHHEAD))))
  (preference(category ?cat&: (eq ?cat (get-member engines.gps.PreferenceCategories
ALLHUMANS))) (value ?val&: (< ?val 50)))
  ?m<-(mood (valence ?valence&: (> ?valence 0)))
  =>
  (printout t "HeadTouch4 event was caught" crlf)
  (retract ?extE)
  (interpretEmotion (get-member engines.Emotion ANGRY) 60)
  (modify ?m (valence (- ?valence 10)))
  (call ?*needs* setBelonging (- (call ?*needs* getBelonging) 10))
  (runPERS)
)

(defrule chinTouch1
  ?extE<-(extEvent(eventName ?n&: (eq ?n (get-member engines.gps.Action TOUCHCHIN))))
  ;(preference(category ?cat&: (eq ?cat 1)) (value ?val&: (< ?val 50)))
  ?m<-(mood (valence ?valence&: (> ?valence 0)))
  =>
  (printout t "ChinTouch1 event was caught" crlf)
  (retract ?extE)
  (interpretEmotion (get-member engines.Emotion HAPPY) ?val)
  (modify ?m (valence (+ ?valence 5)))
  (call ?*needs* setBelonging (- (call ?*needs* getBelonging) 10))
  (runPERS)
)

(defrule chinTouch2
  ?extE<-(extEvent(eventName ?n&: (eq ?n (get-member engines.gps.Action TOUCHCHIN))))
  ;(preference(category ?cat&: (eq ?cat 1)) (value ?val&: (< ?val 50)))
  ?m<-(mood (valence ?valence&: (< ?valence 0)))
  =>
  (printout t "ChinTouch2 event was caught" crlf)
  (retract ?extE)
  (interpretEmotion (get-member engines.Emotion ANGRY) 60)
  (modify ?m (valence (- ?valence 10)))
  (call ?*needs* setBelonging (- (call ?*needs* getBelonging) 10))
  (runPERS)
)

(defrule chinTouch3
  ?extE<-(extEvent(eventName ?n&: (eq ?n (get-member engines.gps.Action TOUCHCHIN))))
  (preference(category 1|4|5|6|15) (value ?val&: (< ?val 50)))
  ?m<-(mood (valence ?valence&: (> ?valence 30)))
  =>
  (printout t "ChinTouch3 event was caught" crlf)
  (retract ?extE)
  (interpretEmotion(get-member engines.Emotion HAPPY) 60)
  (modify ?m (valence (- ?valence 10)))
  (call ?*needs* setBelonging (- (call ?*needs* getBelonging) 10))
  (runPERS)
)

(defrule chinTouch4

```

```

    ?extE<-(extEvent(eventName ?n&: (eq ?n (get-member engines.gps.Action TOUCHCHIN))))
;(preference(category ?cat&: (eq ?cat 1)) (value ?val&: (< ?val 50)))
?m<-(mood (valence ?valence&: (> ?valence 0)))
=>
  (printout t "ChinTouch4 event was caught" crlf)
  (retract ?extE)
  (intrepretEmotion (get-member engines.Emotion ANGRY) 60)
  (modify ?m (valence (- ?valence 10)))
  (call ?*needs* setBelonging (- (call ?*needs* getBelonging) 10))
  (runPERS)
)

(defrule BackTouch1
  ?extE<-(extEvent(eventName ?n&: (eq ?n (get-member engines.gps.Action TOUCHBACK))))
;(preference(category ?cat&: (eq ?cat 1)) (value ?val&: (< ?val 50)))
?m<-(mood (valence ?valence&: (> ?valence 0)))
=>
  (printout t "BackTouch1 event was caught" crlf)
  (retract ?extE)
  (intrepretEmotion (get-member engines.Emotion ANGRY) 60)
  (modify ?m (valence (- ?valence 10)))
  (call ?*needs* setBelonging (- (call ?*needs* getBelonging) 10))
  (runPERS)
)

(defrule BackTouch2
  ?extE<-(extEvent(eventName ?n&: (eq ?n (get-member engines.gps.Action TOUCHBACK))))
;(preference(category ?cat&: (eq ?cat 1)) (value ?val&: (< ?val 50)))
?m<-(mood (valence ?valence&: (> ?valence 0)))
=>
  (printout t "BackTouch2 event was caught" crlf)
  (retract ?extE)
  (intrepretEmotion (get-member engines.Emotion ANGRY) 60)
  (modify ?m (valence (- ?valence 10)))
  (call ?*needs* setBelonging (- (call ?*needs* getBelonging) 10))
  (runPERS)
)

(defrule BackTouch3
  ?extE<-(extEvent(eventName ?n&: (eq ?n (get-member engines.gps.Action TOUCHBACK))))
;(preference(category ?cat&: (eq ?cat 1)) (value ?val&: (< ?val 50)))
?m<-(mood (valence ?valence&: (> ?valence 0)))
=>
  (printout t "BackTouch3 event was caught" crlf)
  (retract ?extE)
  (intrepretEmotion (get-member engines.Emotion ANGRY) 60)
  (modify ?m (valence (- ?valence 10)))
  (call ?*needs* setBelonging (- (call ?*needs* getBelonging) 10))
  (runPERS)
)

(defrule BackTouch4
  ?extE<-(extEvent(eventName ?n&: (eq ?n (get-member engines.gps.Action TOUCHBACK))))
;(preference(category ?cat&: (eq ?cat 1)) (value ?val&: (< ?val 50)))
?m<-(mood (valence ?valence&: (> ?valence 0)))
=>
  (printout t "BackTouch4 event was caught" crlf)
  (retract ?extE)
  (intrepretEmotion (get-member engines.Emotion ANGRY) 60)
  (modify ?m (valence (- ?valence 10)))
  (call ?*needs* setBelonging (- (call ?*needs* getBelonging) 10))
  (runPERS)
)

(defrule BackHit1
  ?extE<-(extEvent(eventName ?n&: (eq ?n (get-member engines.gps.Action HITBACK))))
;(preference(category ?cat&: (eq ?cat 1)) (value ?val&: (< ?val 50)))
?m<-(mood (valence ?valence&: (> ?valence 0)))
=>
  (printout t "BackHit1 event was caught" crlf)
  (retract ?extE)
  (intrepretEmotion (get-member engines.Emotion ANGRY) 60)
  (modify ?m (valence (- ?valence 10)))
  (call ?*needs* setBelonging (- (call ?*needs* getBelonging) 10))
)

```

```

    (runPERS)
  )

(defrule BackHit1
  ?extE<-(extEvent(eventName ?n&: (eq ?n (get-member engines.gps.Action HITBACK))))
  ;(preference(category ?cat&: (eq ?cat 1)) (value ?val&: (< ?val 50)))
  ?m<-(mood (valence ?valence&: (> ?valence 0)))
  =>
  (printout t "BackHit1 event was caught" crlf)
  (retract ?extE)
  (intrepretEmotion (get-member engines.Emotion ANGRY) 60)
  (modify ?m (valence (- ?valence 10)))
  (call ?*needs* setBelonging (- (call ?*needs* getBelonging) 10))
  (runPERS)
)

(defrule BackHit2
  ?extE<-(extEvent(eventName ?n&: (eq ?n (get-member engines.gps.Action HITBACK))))
  ;(preference(category ?cat&: (eq ?cat 1)) (value ?val&: (< ?val 50)))
  ?m<-(mood (valence ?valence&: (> ?valence 0)))
  =>
  (printout t "BackHit2 event was caught" crlf)
  (retract ?extE)
  (intrepretEmotion (get-member engines.Emotion ANGRY) 60)
  (modify ?m (valence (- ?valence 10)))
  (call ?*needs* setBelonging (- (call ?*needs* getBelonging) 10))
  (runPERS)
)

(defrule BackHit3
  ?extE<-(extEvent(eventName ?n&: (eq ?n (get-member engines.gps.Action HITBACK))))
  ;(preference(category ?cat&: (eq ?cat 1)) (value ?val&: (< ?val 50)))
  ?m<-(mood (valence ?valence&: (> ?valence 0)))
  =>
  (printout t "BackHit3 event was caught" crlf)
  (retract ?extE)
  (intrepretEmotion (get-member engines.Emotion ANGRY) 60)
  (modify ?m (valence (- ?valence 10)))
  (call ?*needs* setBelonging (- (call ?*needs* getBelonging) 10))
  (runPERS)
)

(defrule BackHit4
  ?extE<-(extEvent(eventName ?n&: (eq ?n (get-member engines.gps.Action HITBACK))))
  ;(preference(category ?cat&: (eq ?cat 1)) (value ?val&: (< ?val 50)))
  ?m<-(mood (valence ?valence&: (> ?valence 0)))
  =>
  (printout t "BackHit4 event was caught" crlf)
  (retract ?extE)
  (intrepretEmotion (get-member engines.Emotion ANGRY) 60)
  (modify ?m (valence (- ?valence 10)))
  (call ?*needs* setBelonging (- (call ?*needs* getBelonging) 10))
  (runPERS)
)

;YES
(defrule PawLF
  ?extE<-(extEvent(eventName ?n&: (eq ?n (get-member engines.gps.Action TOUCHPAWLF))))
  ?m<-(mood (valence ?valence&: (> ?valence 0)))
  =>
  (printout t "PawLF event was caught" crlf)
  (retract ?extE)
  (intrepretAction (call companionAIBO.Head NO))
  ;(intrepretEmotion (get-member engines.Emotion ANGRY) 60)
  (modify ?m (valence (+ ?valence 10)))
  (call ?*needs* setBelonging (- (call ?*needs* getBelonging) 10))
  (runPERS)
)

;NO
(defrule PawRF

```

```

    ?extE<-(extEvent(eventName ?n&: (eq ?n (get-member engines.gps.Action TOUCHPAWRF))))
    ?m<-(mood (valence ?valence&: (> ?valence 0)))
    =>
      (printout t "PawRF event was caught" crlf)
      (retract ?extE)
      (interpretAction (call companionAIBO.Head YES))
      (printout t "cucu" crlf)
      (modify ?m (valence (- ?valence 10)))
      (call ?*needs* setBelonging (- (call ?*needs* getBelonging) 10))
      (runPERS)
  )
(defrule PawLB
  ?extE<-(extEvent(eventName ?n&: (eq ?n (get-member engines.gps.Action TOUCHPAWLB))))
  ?m<-(mood (valence ?valence&: (> ?valence 0)))
  =>
    (printout t "PawLB event was caught" crlf)
    (retract ?extE)
    (interpretEmotion (get-member engines.Emotion ANGRY) 60)
    (modify ?m (valence (- ?valence 10)))
    (call ?*needs* setBelonging (- (call ?*needs* getBelonging) 10))
    (runPERS)
)
(defrule PawRB
  ?extE<-(extEvent(eventName ?n&: (eq ?n (get-member engines.gps.Action TOUCHPAWRB))))
  ?m<-(mood (valence ?valence&: (> ?valence 0)))
  =>
    (printout t "PawRB event was caught" crlf)
    (retract ?extE)
    (interpretEmotion (get-member engines.Emotion ANGRY) 60)
    (modify ?m (valence (- ?valence 10)))
    (call ?*needs* setBelonging (- (call ?*needs* getBelonging) 10))
    (runPERS)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;Rules that evaluate the standards upon occurrence of events
;;All rules that change the needs need to call the PERSONALITY ENGINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;This is how a rule in this section is structured. Please follow this structure
closely!!!
;(defrule eventNamedRule_no
  ;(standard(agent ?agent&: (? ?agent ?)) (action ?action&: (? ?action ?)) (value
?val&: (? ?val ?)))
  ;?id<-(intEvent(eventName ?name&: (? ?name ?)))
  ;?m<-(mood (valence ?valence&: (? ?valence ?)))
  ;=>
    ;the event that generated the emotion is retracted because it is no further needed
    ;(retract ?id)
    ;change mood. Mood decreases with ? units.
    ;(modify ?m (valence (- ?valence 10)))
    ;AIBO's emotion is generated and the EmotionInterpreter is called
    ;(interpretEmotion (get-member engines.Emotion SAD 40))
    ;change needs. setNeed -is the name of a setter in the needs class and ? should be
replaced with a value for that need
    ;(call ?*needs* setNeed ?)
    ;(printout t "If you want to print out something do it here" crlf)
    ;Call the Personality Engine...since change in needs might effect the GPS.
    ;(runPERS)
;)
;;;;;;;;;;;;;;;;;END OF COMMENT representing an example of a rule in this section

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;Rules that evaluate the goals upon occurrence of events
;;All rules that change the needs need to call the PERSONALITY ENGINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;This is how a rule in this section is structured. Please follow this structure
closely!!!
;(defrule eventNamedRule_no
  ;(goal ())

```

```

;?id<-(intEvent(eventName ?name&: (? ?name ?)))
;?m<-(mood (valence ?valence&: (? ?valence ?)))
;=>
    ;the event that generated the emotion is retracted because it is no further needed
;(retract ?id)
;change mood. Mood decreases with ? units.
;(modify ?m (valence (- ?valence 10)))
;AIBO's emotion is generated and the EmotionInterpreter is called
;(intrepretEmotion (get-member engines.Emotion SAD 40))
;change needs. setNeed -is the name of a setter in the needs class and ? should be
replaced with a value for that need
;(call ?*needs* setNeed ?)
;(printout t "If you want to print out something do it here" crlf)
;Call the Personality Engine...since change in needs might effect the GPS.
;(runPERS)
;)
;;;;;;;;;END OF COMMENT representing an example of a rule in this section

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;The rule that sets the engine waiting untill events are generated
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defrule waitEvents
    ;This is the rule with the lowest salience...it has to be called after all other rules
have fired.
    (declare (salience -200))
    =>
    (printout t "GPS Engine is waiting for events to fire ... " crlf)
    (waitEvent)
)

```

## Appendix D: RobotOutput Package

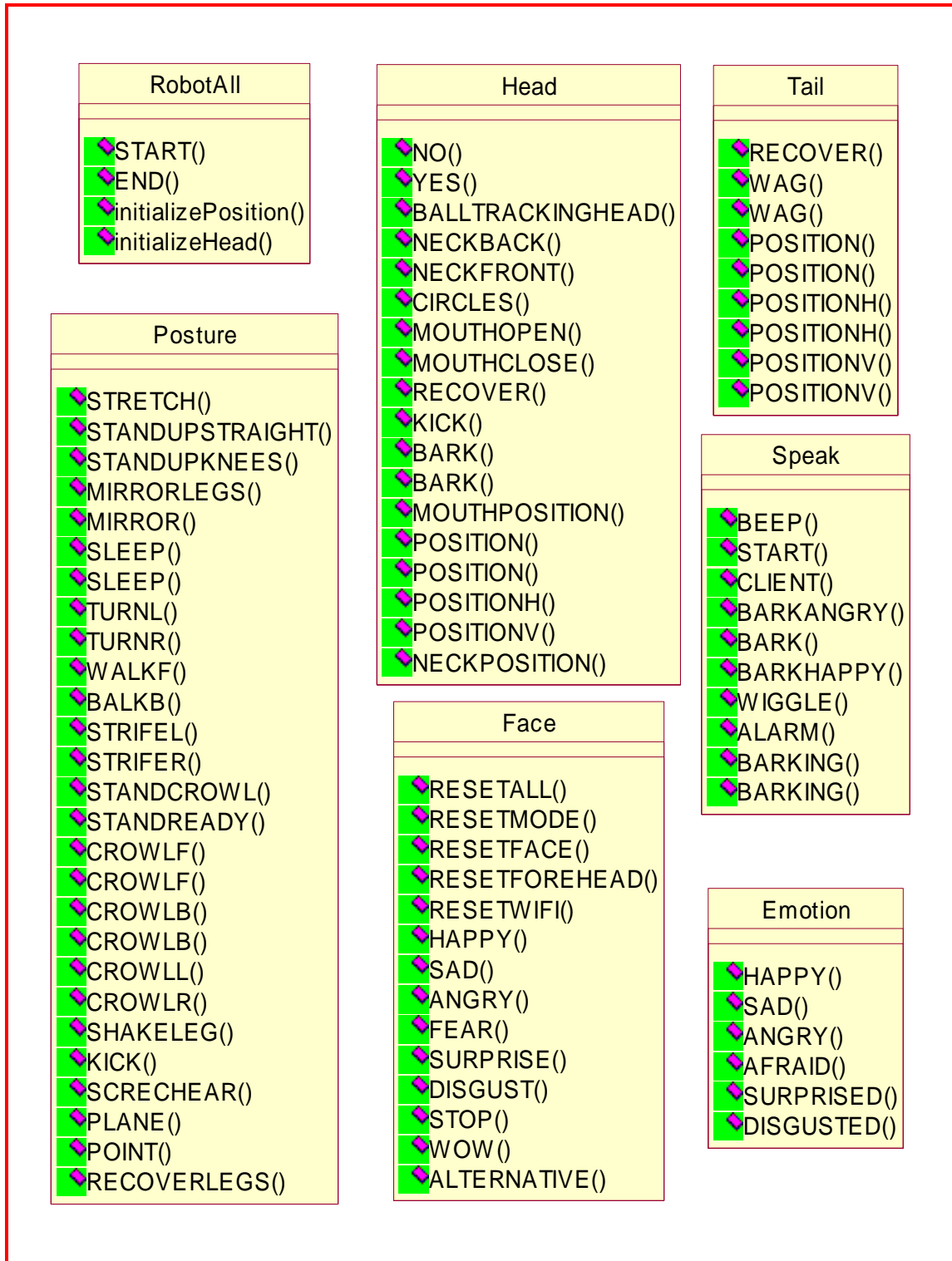


Figure 31: RobotOutput class diagram

More details on all these functions and their roles are to be found in the Java API.

## Appendix E: ACE Paper

### Personality model for a companion AIBO

Iulia Dobai

Delft University of Technology  
Mekelweg 4, Delft, 2628CD

The Netherlands  
+31 610700219

[i.dobai@ewi.tudelft.nl](mailto:i.dobai@ewi.tudelft.nl)

Leon Rothkrantz

Delft University of Technology  
Mekelweg 4, Delft, 2628CD

The Netherlands  
+31 15 2787504

[l.j.m.rothkrantz@ewi.tudelft.nl](mailto:l.j.m.rothkrantz@ewi.tudelft.nl)

[Charles van der Mast](#)

Delft University of Technology  
Mekelweg 4, Delft, 2628CD

The Netherlands  
+31 152787504

[c.a.p.g.vandermast@ewi.tudelft.nl](mailto:c.a.p.g.vandermast@ewi.tudelft.nl)

#### ABSTRACT

In this paper we describe the architecture that allows the modeling of an emotionally intelligent robot. We chose to implement these architectures on AIBO, which is a quadruped autonomous robot, developed by Sony. AIBO was developed as an entertainment robot with its “mind” resident on a memory stick. Our goal is to develop a new mind for a companion AIBO with a more complex personality model. The focus of this paper is on a model of personality. Taking into consideration the realities concerning an emotionally intelligent AIBO, that acts in an unpredictable and changing environment, the existing models of personality need improvement, modifications and adaptations to the current situation. Most existing personality models that are used in virtual humans and agents take into consideration three layers: personality, mood and emotions or just two: personality and emotions. We, on the other hand will try to add a new set of parameters: needs that will not constitute a new layer but together with the personality layer, a pillar for the top two layers of mood and emotions. This paper introduces the

architecture that renders valuable the personality model discussed above.

#### Categories and Subject Descriptors

D.2.11 [Software Engineering]: Software Architectures - *domain-specific architectures*

#### General Terms

Design, Human Factors.

#### Keywords

AIBO-robot, human-computer interaction, companion robot, personality modeling.

## 8 INTRODUCTION

Studies conducted by Masahiro Fujita assess that AIBO seems to be a good partner with users, having a positive effect on their emotional state [5]. Very recent experiments show that AIBO is useful for mental therapy from a medical point of view. According to the same author the current implementation of AIBO software provided by Sony uses behaviors that come from a “manually designed database”. Next steps would be an *open-ended system* or an *ever-evolving system* by which new behaviors are emerged through the interaction with human and environment. While some efforts are being put by others into developing *unknown-word technologies* and *unknown-object learning technologies* we thought about improving the personality model that judges upon the interaction with human or environment.

Work has been done until now in developing AIBOs that can perform well different tasks (entertainment, watch-dog, etc.). All these task-focused behaviors have in common the physical/hardware components of an AIBO. While some might argue that a watch dog for example does not need personality, moods or emotions it is our believe that all these task orientated AIBOs need to have in common also a complex personality model. The division on tasks should be made at a higher cognitive level, while the personality model has to act as a ground field.

This paper is organized as follows: first we will introduce the personality model that we have developed which will be followed by the architecture that we used to implement it on an AIBO.

## 9 The nPME Model

Until now in the design of virtual humans, agents or game characters, two categories of personality models were mostly used: PME models that take into consideration personality(P), mood(M) and emotions(E) and PE models that are based solely on personality(P) and emotions(E).

In designing our nPME model we started with a layered PME architecture as described in [6]. Here mood is seen as an intermediate layer between personality and emotions and therefore is influenced by both. Since we want AIBO to act independently in a changing environment we have to take into consideration his individual needs, therefore we included in the layered architecture the needs parameter (nPME). By adding needs we also had to make some major changes in the existing architecture and therefore mood is directly influenced by neither emotions nor personality, but it is indirectly influenced by them as a consequence of the evaluation of the goals, preferences and standards in regards with the events that take place.

## 9.1 Concepts

### 9.1.1 Personality

The first important factor of the personality model is personality itself. Of all the existing psychological models of personality the one most widely accepted is the Five Factor Model. This model is based on the idea that personality can be described and measured on five broad dimensions: Openness, Conscientiousness, Extraversion, Agreeableness and Neuroticism. (Known as the OCEAN Model)[2]. Conventional wisdom has long set that our personality is genetic and therefore set in stone but according to some scientist those five key personality characteristics change throughout our lives. In the case of robots that don't really have a life span, it is acceptable to have a persistent personality that does not change due to external influences. We based our decision on Paul Costa Jr. whose work revealed that we don't see major personality changes but we see "nuanced" changes [2] that we consider to be insignificant in the case of robots. Therefore in the model we introduce, personality is not changed through time and the values for the 5 dimension of personality are constant through the application.

### 9.1.2 Needs

It is generally agreed that the simpler emotions, whose expression and recognition Ekman (1989) has shown to be universal are driven by the basic need of organisms such as mating, defense or avoidance of predators, and social affiliation [3]. Therefore here is the theoretical framework we were looking for. In order to develop a realistic AIBO companion dog that shows emotions in a changing and unpredictable environment we have to take into consideration his needs. We chose to use the highly appreciated theory of needs introduced by Maslow [8] (see Figure 1). The use of Maslow Pyramid of Needs is not accidental but based on very solid reasons: it seems the one most widely accepted and it is a model simple enough to be implemented and sufficiently complex to show realistic results. The needs model proposed by Maslow is based on a



structure (the pyramid), a well defined set of rules and principles and uses priorities that will be in our case a way to support a priority management system in AIBO.

The pyramid of needs that Maslow introduced as a base for motivation theory contains 5 categories of needs: physiological, safety, love and belonging, self esteem and self actualization. The first 4 categories of needs have been introduced by Maslow as “deficit needs” while the last category of needs is known as “being needs”. In our nPME model we will take into consideration only the “deficit needs” since these needs can be met fully while self actualization needs are seen as growing (a continuous driving force). Maslow supports our decision by explaining that not everyone ultimately seeks self-actualization [8]. The hierarchy of needs states that we must satisfy each need in turn, starting with the first, which deals with the most obvious needs for survival itself. Only when the lower order needs of physical and emotional well-being are satisfied we are concerned with the higher order needs of influence and personal development. Conversely, if the things that satisfy our lower order needs are swept away, we are no longer concerned about the maintenance of our higher order needs. In practice we translated the different categories of needs into AIBO specific needs (e.g. physiological needs are represented by the battery

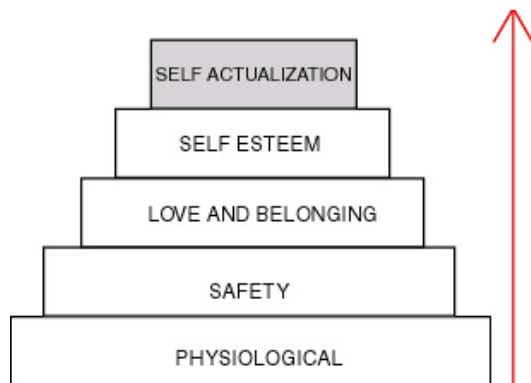


Figure 1. Maslow Pyramid of Needs

level, etc.). Needs have priorities and act as thermometers, once critical values have been reached by different categories of needs depending on their priorities they lead to different

structures of goals, preferences and standards. (e.g. once the battery reaches a critical value of 10 % a new goal with the highest priority is being generated that states: recharge, etc.).

### 9.1.3 Mood

Mood is a conscious and prolonged state of mind that directly controls the emotions. While emotions are instantaneous, mood is constant for longer time spans [6]. The presence of mood in a well established personality model is doubtless, without mood we can not properly link personality which is in a sense “eternal” to emotional expressions that are instantaneous. The mood model we have used has its fundamentals in the mood model proposed by Lang [7] that basically plots mood on a two dimensional system with *valence* and *arousal* as axes. Valence refers to whether the mood is positive or negative and arousal refers to the intensity of the mood.

### 9.1.4 Emotional States and Expressions

By emotional state we understand a particular state of mind that is reflected visually by means of an emotional expression. We use the emotion categories proposed by Orthony, Clore and Collins, commonly known as the OCC model [9]. The model categorizes 22 different emotions based on positive or negative reactions to events, actions and objects. We will not use the cognitive processing proposed by the OCC model directly but all entities in the model are developed based on ideas included in the OCC model. We based our decision regarding this on [1]. To date cognitive science does not seem to have provided any crucial tests to decide between competing models of the mind. What does seem well established in the light of cross-cultural research is that a number of emotions have inter-translatable names and universally recognizable expressions [3]. According to Ekman and Friesen [4] these are happiness, sadness, fear, anger, surprise and disgust. Another idea in the field of emotions that is becoming widely accepted is that emotions, are capable of

being not only explained but also justified - they are closely related to the reasons that give rise to them. Emotions can be categorized in terms of their associated cognitions [3].

### 9.1.5 Goals, Preferences and Standards

The goals, preferences and standards parameters will be set dynamically based on needs and personality. Every time changes have occurred in the parameters of needs the goals, preferences and standards (GPS system) will also be updated.

By goals we understand tasks orientated objectives that are SMART (simple,

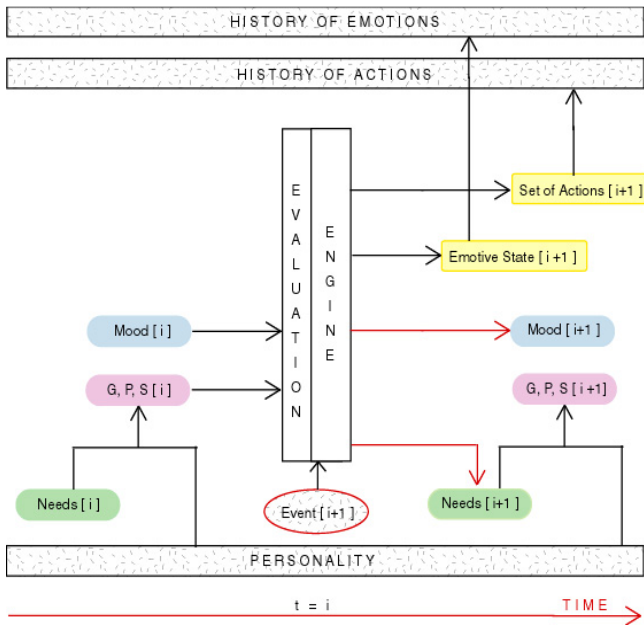


Figure 2. Time based nPME model

measurable, acceptable, realistic, time frame) (e.g.: aibo find ball). By preference we understand appealingness to aspects of objects (e.g. like/dislike of ball). By standards we understand approval/disapproval regarding actions of agents. Standards can focus on self agent or other agents (e.g. approval of being touched on back).

## 9.2 Personality model overview

### 9.2.1 A schematic approach

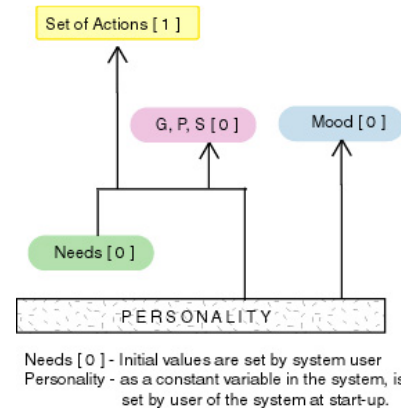


Figure 3. Personality model initialization.

Figure 2 introduces the time based nPME model developed by us that has as ultimate goal the development of a companion AIBO that acts independently in a changing environment. According to this design AIBO shows an emotional reaction and will follow a specific set of actions only if some kind of event triggers the engine that will generate them.

Initial set-up requires values for all traits of personality (that will remain constant) and an initial set of needs. Every time there is a new set of needs present in the system a small engine will be activated to update values for goals, preferences and standards. If an event occurs (by event we understand a registered happening that is detected by AIBO) the response engine is triggered and based on the evaluation of goals, preferences and standards and on the current mood a new emotional state and a set of actions results.

Figure 3 presents the initialization of the entire system at moment  $t=0$ . At this moment we have to take into consideration the possibility that no event happens in a certain amount of time, therefore based on the personality of AIBO and its needs we decide on a few actions randomly chosen from the list of possible actions. It is important to mention that even if no event occurs in a specified amount of time an event will be generated. (e.g. the absence of any human for prolonged amounts of time in the vicinity of an AIBO represents an event that will lead to an increase of the need for love and belonging and a

switch of the mood from positive to negative with different degrees).

## 10 DESIGN CONCEPT FOR COMPANION AIBO

### 10.1 A modular architecture

Figure 4 presents the architecture we designed to integrate the personality model described above and the division of logical parts into system components.

#### 10.1.1 Input translation

AIBO has the capacity of giving input information through a multimodal set of sensors: camera for vision, microphone for sound, touch sensors for tactile and a set of other useful sensors like distance, vibration, etc. The existence in AIBO of so many ways to receive input and information concerning the context has opened up space for complex modules to analyze and convert this information into the entities the next component needs. This component is an AIBO dependent component and its major roles are to read sensor data, to do preprocessing (pattern recognition, sound recognition, etc) and to convert it into events that are seen as triggers by the next component. Events that are thrown by this component are beforehand registered with the next component.

AIBO receives information from the outside world through his sensors: video camera (a 350,000 CMOS image sensor), stereo microphones in its ears, two distance sensor, and various touch sensors on head, back, chin, paws, acceleration sensor, and a vibration sensor.

#### 10.1.2 Processing

This component does the central processing for the entire system, is not robot dependent and is basically the implementation of the personality model described above. This component is designed to act standalone and was implemented with a constant desire for flexibility. The output of this component at moment  $t=i$  is an emotional state and a possible set of actions.

#### 10.1.3 Mapping

This component will interpret the output of the processing component and will translate the somewhat abstract notions like: happiness and play with ball into procedures for accomplishing them. The output of this component is a complex set of data structures containing information that can be directly transformed into robot specific commands.

#### 10.1.4 Output translation

This component takes the output of the mapping component and handles the connection to AIBO and the specific robot commands AIBO needs to execute in order to have the desired output. In order to give output responses AIBO is equipped with speaker on its chest, LED Lights (on face, ears and back) and a series of movable parts: head (3 DOF<sup>2</sup>), mouth (1 DOF), legs (4\*3 DOF), ears (2\*1 DOF) and tail (2 DOF). Apart from this AIBO is also equipped with a wireless card and a blue LED to show its status.

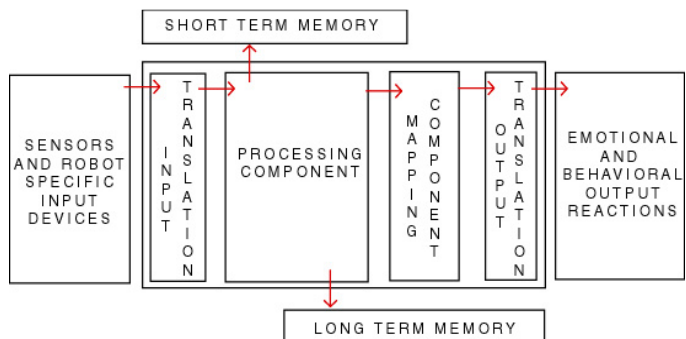


Figure 4. System Architecture

### 10.2 Development perspective

The modular architecture we introduced is based on the existence of another architecture developed in the URBI Project at ENSTA Laboratories. **URBI** (Universal Robotic Body Interface) [10] is a scripted language designed to work over a client-server architecture in order to remotely control a robot. URBI is released under the GNU General Public License. Our present prototype runs the server on AIBO (the original URBI server) and the client on a PC (developed

<sup>2</sup> DOF = Degrees of Freedom

by us), therefore the mind is resident on the PC and AIBO is in charge of sending raw data information from the sensors and executing complex commands written in the URBI scripting language. Our long term goal is to have both client and server running on AIBO and in this way having a perfectly autonomous robot.

### 10.2.1 Implementation

A prototype has been developed to test the personality model using two expert systems: one generating the data for the second one. Practically the first expert system has as facts values for personality and needs and rules that generate preferences, standards and goals. The second expert system is generating emotions and actions while also updating the values for needs and mood. The facts of the second expert system are: preferences, goals, standards, mood and events. Events are added dynamically to the expert system when their occurrence is signaled through the sensors of AIBO. Once an event was evaluated in regards with the goals, preferences and standards it is retracted from the input set. The complete architecture presented in Figure 4 is still under development due to the high complexity of some modules like the Input Translation Module which does the difficult job of analyzing all sensor data. Sound and vision recognition in the context of this framework are the subject of other research projects.

## 11 CONCLUSIONS

A personality model to be implemented in robots acting in a changing and unpredictable environment has to take into account the needs of the robot in that specific environment. A robot that can not evaluate his needs is not capable of autonomously acting in a dynamic environment. We make sure that we respect the integrity of the systems proposed by transforming all input information into relevant events for the system. Further work will include integrating the implementation with complex vision and sound recognition that will generate an extra effort in developing a complex event management system. Further work should also include integrating

learning modules and also more elaborated history management systems.

## 12 ACKNOWLEDGMENTS

Our thanks to “The URBI Project”, an excellent alternative to the Sony development platform for AIBO.

## 13 REFERENCES

- [39] Bartneck, C., “Integrating the OCC Model of Emotions in Embodied Characters”, *Workshop on Virtual Conversational Characters*, 2002
- [40] Costa, P.T., McCrae, R.R., “Normal personality assessment in clinical practice: The NEO personality inventory”. *Psychological Assessment*, 1992
- [41] de Sousa, R., "Emotion", *Stanford Encyclopedia of Philosophy (Spring 2003 Edition)*, Edward N. Zalta (ed.), <http://plato.stanford.edu/archives/spr2003/entries/emotion>
- [42] Ekman, P., Friesen, W.V., “The argument and Evidence About Universals in Facial Expressions of Emotion”. In *Handbook of Social Psychophysiology*. New York: John Wiley and Sons, Ltd, 1989
- [43] Fujita, M. “On activating Human Communication with Pet-Type Robot AIBO”. In *Proceedings of the IEEE, Vol. 92, No. 11*, 2004, 1804-1813
- [44] Ksirsagar, S., Magnenat-Thalmann, N., “A Multilayer Personality Model”, In *Proceedings of 2nd International Symposium on Smart Graphics*, 2002, 107-115
- [45] Lang, P.J., *The Emotion Probe: studies of motivation and attention*, A study in the Neuroscience of Love and Hate. Hillside, NJ: Lawrence Erlbaum Associates, Publishers, 1995.
- [46] Maslow, A.H., *Motivation and Personality*, 2nd. ed., New York, Harper & Row, 1970
- [47] Ortony, A., Clore, G.L., Collins, A., *The Cognitive Structure of Emotions*, Cambridge University Press, 1988
- [48] [www.urbiforge.com](http://www.urbiforge.com)

