A MULTIMODAL LEGO ROBOT

Guillaume Barraud, Priam Pierret and Léon Rothkrantz
Data and Knowledge Systems Group
Department of Information Technology and Systems
Delft University of Technology
Mekelweg 4, 2628 CD Delft, the Netherlands
E-mail: L.J.M.Rothkrantz@cs.tudelft.nl

KEYWORDS

Lego robot, multimodal interaction, artificial intelligence.

ABSTRACT

The goal of the project was to develop a robot and a multimodal user interface. The robot, designed as a digital cat, can show complex behaviours such as move, speak, touch, listen, and read. The input command interface is based on text, icons, and speech. A prototype of the robot is implemented using the Lego Mindstorms™ System. The design and implementation of the robot are presented in this paper.

INTRODUCTION

At Delft University of Technology there is a project on multimodal communication. At the moment research topics focus on automatic recognition and generating of facial expressions, and automatic speech recognition. To develop new ways of human-computer interaction a test environment was created: AMAELIA (A Multimodal Application for Extensible Lego Intelligent Agent). The robot is similar to the well-known Aibo robot developed by Sony, but unlike the Aibo we wanted to create an open-source and open-development environment. To implement the robot we used Lego Mindstorms ™ System.

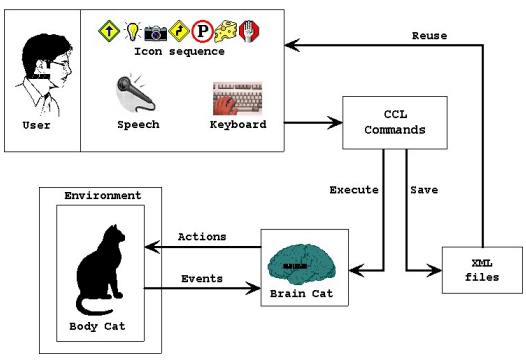

Figure 1: The AMAELIA activity diagram

AI Aspect

AMAELIA is an environment for editing, executing and saving behaviors with a Lego Robot Cat. It can be called an AI environment because the core of the system is designed as an intelligent agent, according to the PAGE definition (Percepts, Actions, Goals, Environment).

Entertaining Aspect

The main use of the application is to interact with a Lego Robot Cat equipped with Lego Camera, which can move, play sounds and music, speak, take pictures and capture videos, but it can also see, watch, touch, listen, read. You can also teach AMAELIA how to react when it is running, and all these things can be done at the same time. After getting used with the Cat Command Language, you can easily edit more complex behaviors, from the funniest to the most useful, from the most stupid to the most intelligent. When your new behaviors are ready for use, you can demonstrate them by using the speech command.

Components Aspect

AMAELIA has a lot of advanced features like infrared communication, image processing for color and movement detection, speech recognition and generation, etc. We used existing components for most of these advanced features. The used components are ActiveX components for Windows operating systems. The AMAELIA program was written in Visual Basic, which is very efficient for ActiveX reuse, graphic user interface design and quick development.

ARCHITECTURE OF AMAELIA

The architecture of AMAELIA is designed according to an object-oriented approach; there are seven main entities, all of them embed one or several existing ActiveX components (see also Figure 2).

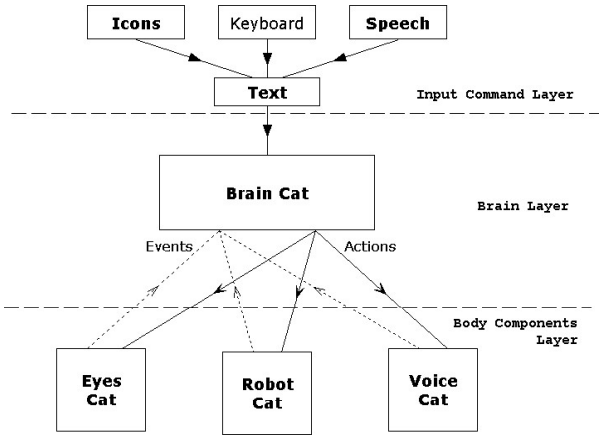

Figure 2: Architecture of AMAELIA

The architecture can be divided into 3 layers:

1. Body components (low layer), which owns three entities.
2. Brain (middle layer), which is one entity.
3. Commands (high layer), which owns three entities.

The Brain layer has only one entity, which is the core of the system. All the entities in a same layer (1 or 3) are equivalents in terms of role in the system.

Body Components layer

The entities in this layer can access the hardware to produce a physical action: RobotCat is in charge with the Lego Robot, EyesCat with the Lego Camera, and VoiceCat with speech generation. These components are also sources of events (contactPushed, objectSeen, endOfSpeech...). The events are sent to the BrainCat which is the only entity allowed to trigger actions on the Body entities.

Brain layer

The Brain entity organizes the execution of the cat behaviors (structured as a tree) that the user defines with the entities of the Commands layer. With the Body components, the Brain entity calls some actions and receives some events. From this point of view the Brain entity is designed as an agent.

Commands layer

The entities in the Commands layer are the multimodal interfaces for the user to give orders with text, icons, and speech. Icons and speech are translated into text, and then text is parsed to build an StrTree structure, which is the command input for the Brain entity.

ROBOTCAT

A couple of years ago, the Lego Company released a new range of Lego toys called Lego Mindstorms™ System. The goal of these toys was to give children (and adults) a tool to learn developing and building robots. The kit allows you to build a Lego robot and command it from your PC. This new range uses the pieces of the Lego Technics range, but Lego adds some special pieces:

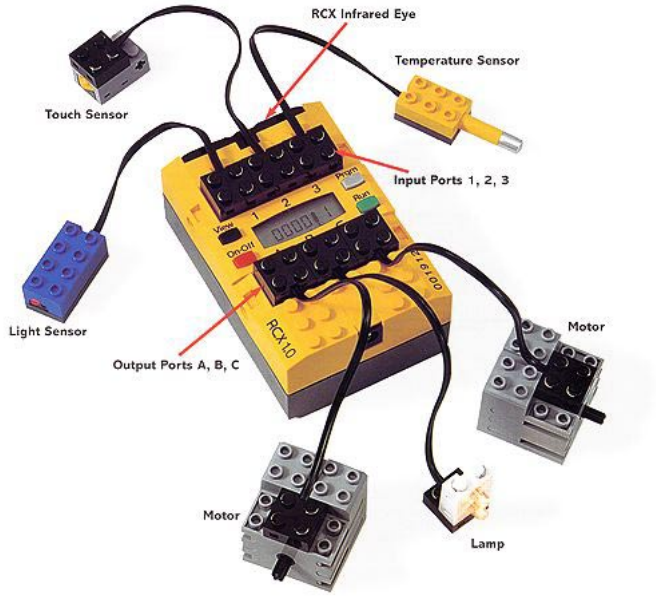

Figure 3: The LEGO Mindstorms system

- The RCX or programmable brick: the main yellow piece in Figure 3 is a big Lego brick with a microprocessor inside and some inputs and outputs on the top. It can communicate with the PC via an infrared port located on the RCX and an infrared tower that should be connected on the COM port of the PC.
- The output bricks: motors, lights.
- The input bricks (sensors): to detect contact, rotation, light and temperature.
- The cable bricks: these are just two small and simple Lego bricks with electrical contacts and are used to connect a RCX port to another special brick (inputs and outputs).

Building

The first stage of the realization of the robot is building it. It could be almost summarized in three words: connecting Lego bricks. Indeed, from its nature, Lego offers us such an easy building way, which gives us a lot of building possibilities. However we were inspired from one of the most basic models (and thus one of the most functional) to build our robot.

One of the major simplifications that we would like to point out that the robot has 'no legs'. Instead we used caterpillars and wheels. Our robot can nevertheless use legs if it is wished (the wheel-caterpillar-legs are interchangeable) but the accuracy is reduced during moving and the control is much more random. Two engines are devoted for moving, using two PBrick outputs. The third output is used to connect the lamp. For the sensors of the

robot, we equipped it with a rotation sensor allowing it to measure the distances covered. We also placed two contact sensors on the front side of the robot on the bumper, which enables it to detect contact with obstacles on the left and right independently.

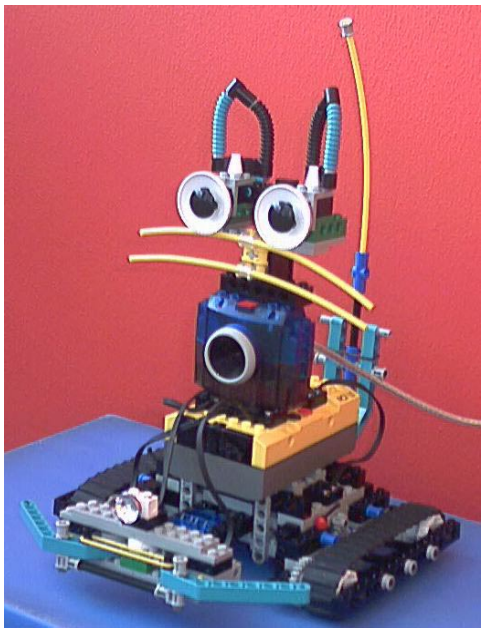

Figure 4: The LEGO cat

Software

There is software included in the Lego box. The program is called RIS (Robotics Invention System). RIS allows you to program simple behaviors using the RCX brick by visually connecting procedures. However, it is too much limited for our needs, in particular for extensibility and the possibility of adding external components such as the Speech API.

The Lego software includes the Spirit.ocx component. The Spirit component is an ActiveX control with access to the COM port and the infrared tower connected on it to communicate with the RCX programmable brick (Pbrick). You can control the PBrick in two ways, and most of the Spirit methods are available for both ways:

1. Direct commands: the action is done on the PBrick when the method is called on the PC.
2. Downloadable commands: the command is downloaded when the method is called on the PC, the command is executed in the PBrick when the program has been started.

To store downloadable commands, the PBrick has 5 programs slot, each of them can contains 10 tasks and 8 subroutines. The ActiveX control can only be accessed from a programming language, which provides ActiveX dynamic linking. Common user languages for this are C++ and Visual Basic. We chose Visual Basic for its advantages of quick development.

We can group the actions of the Lego robot into four categories:

1. Moving actions: driveForward, rotate Left
2. Sounds and music actions: play Sound, play Music.
3. Light actions: setLightOn, set LightOff.
4. Systems actions: setPowerMotor, set PowerDownTime.

The RobotCat cab fire four events from the contact sensors, they can be left or right contact which is pushed or released.

EyesCat

Another kit in the Lego Mindstorms System line is the Vision Command kit, which provides the Lego Camera and the Vision Command software. The Vision Command software allows the user to command the PBrick according to some camera events fired from colour and movement detection. The Lego Camera is actually a simple web cam using the standard QuickCam drivers. Logitech, the QuickCam provider, offers the QuickCam SDK, which is a set of ActiveX libraries and controls. EzVidCap is another free ActiveX control to preview and capture pictures and videos; it uses the QuickCam SDK. The Lego Camera Control (LCC) is an ActiveX control which uses EzVidCap and which makes colour and movement detections according to a layout of detection zones that the developer can define.

LCC Detection

The Lego Camera Control provides an efficient way to do colour and movement detection. We can define up to 64 layers, each of them can contain up to 64 detection zones for colour or movement. The detection is not done on the real image but on 16-colours version of this image, this increase the reliability of colour detection (movement detection is actually colour-changes detection).

Actions and Events

We had to specify how the system would use the camera, which means what are the layers that can be useful for the Robot Cat. The first idea is to put the camera on the Robot Cat, instead of his eyes, so the camera is looking horizontally. It would have been nice to put a motor with the camera to make it rotating up and down, but we are limited in using motors outputs on the Pbrick; there are only three outputs available, two are used for driving and one for the light. So the camera can only move from left to right, using the rotate driving commands of the whole robot.

Because only one layer is active at the same time, the actions for Eyes Cat are to set a particular layer, or to set the inactive layer to prevent the system to receive events from the camera. Different events are raised according to the layer.

VoiceCat

In order to give a little more presence to our robot cat, we equipped it with a voice. For this purpose, we had to use some classes of the Speech API to develop this small module of speech generation. This module is based on the same model as the other cat commands, as will be discussed

in the section iCatBeh. It means that the order "say" includes ICatBeh interface.

The actions and events module adds the action “say” which takes a string parameter. It introduces also the concept of events of beginning and end of word or phrases, but these events are not available to the AMAELIA user. They are just used to synchronize the execution (when we want to do something after saying something).

BRAINCAT

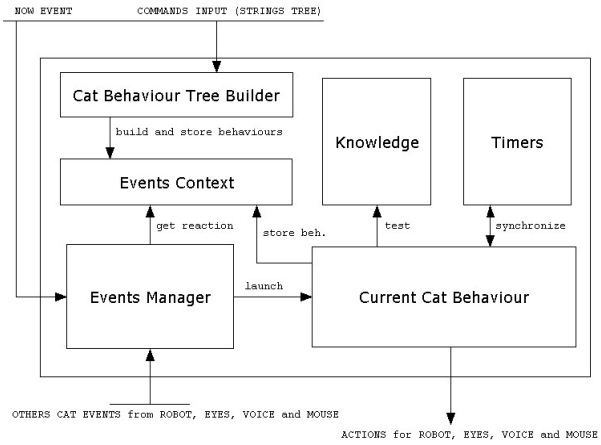

Figure 5: Information flow diagram of BrainCat

Component responsibilities

The Events Manager object listen to events from the Body entities and also to the doNow event that the Commands entities use to start execution.

- The ICatBeh interface is implemented by the current behavior tree, once the execution has been started, it has to organize its own internal execution of its nodes. The implementation of the ’Action()’ method should call specific actions on Body components.
- The Events Context stores all the event-reaction couples. While executing, the user can change these couples using a special control command.
- The Knowledge object is the knowledge base of the system, inside are stored variables of different types which reflect the mind state of the cat (combination of Booleans), and also some integer and string values used for events and actions.
- The CatBehTreeBuilder manages input commands from the Commands layer to build an ICatBeh tree, which will be stored in the Events Context. The input command is already organized into an StrTree.
- The Timers form uses some Timer controls to make a countdown when an action is executing, in order to stop it if the end of task event has not been received (which can happen when the robot goes too far and loses infrared contact).

ICatBeh interface as a composite model

In order to structure our language, we take as a starting point a design pattern called Composite. It is a structural design pattern, which enables us to organize the words of the language in sentences.

The composite model offers to us a common interface for all the words of the language, the ICatBeh interface. But also a structure for the spoken and executed sentences.

In a second part, we introduced some control commands to enrich the language.

- Three conditional commands: if, while, doWhile.
- An event reaction is possible with the command "when".
- A sequential command which is the natural alignment of the words in a sentence.
- A synchronous command: "doBoth".
- A loop command: "repeat"

From a certain point of view, all the orders are equivalent but at the same time, some control commands may contain other basic orders or control itself. The customer would like to process all orders in the same way. For that reason we consider the basic orders like leaves of the tree of the sentence and the nodes of the tree are control icons.

In summary, it gives: basic, or complex orders on the leaves of the tree and the control commands, such as if, while, doWhile, when, doBoth or a sequence command, on the nodes of the tree. We note that in the case of some control icons, the leaves can also be tests (as "if", "while", or "doWhile" children) or events (as "when" children). Finally we note that we are also inspired by the Interpreter design pattern to build the control icons and integrate them in the tree of language.

Discussion about the execution model

The execution model defines how a tree of ICatBeh objects is executed (to execute means to call the method ’Action()’ of the interface ICatBeh). There are two categories of ICatBeh objects as leaves of the tree:

1. Immediate actions: like setLightOn, watchTarget, stopAll.
2. During actions: like driveForward 10 cm, say “Hello“, playMusic A5, A5, A6.

The internal nodes of the tree of ICatBeh objects are necessary controls command like sequence, if, when, while etc. An internal node can be an immediate or during type depending of the children of the nodes. Notice one exception: the when control is always an immediate command because this control command tells the cat how to react when an event occurs. The reaction (the child of the when node) is not executed when the ’ when control’ is executed, but when the event occurs.

We want to build an execution model, which can execute a tree of ICatBeh objects regardless of the category of each

object from the sequence (the sequence is a subsequence of the sequence of all the nodes of the tree in prefix order).

The execution model should satisfy the following two requirements:

1. Execute an action not before the previous action in the sequence is finished.
2. Execute a new sequence when an event occurs.

In the same way, we separate events into two categories:

1. 'end of task' event, which notify that the current action is just finished, so we can now trigger the next action in the current behavior tree, these events are not available at the user level (in the Cat Command Language)
2. 'environmental events', which notify a "Cat Event" from the environment (ex: contactPushed, object seen, speech recognized...), so we need to stop the current execution, to change the current tree and to execute this new tree. These events are available at the user level (in the Cat Command Language), it means that the user can define the behavior to execute when an event occurs.

Our incremental approach of the problem resulted in a design three different execution models. The features provided with the two first models are equivalent, but the first one makes the system call stack growing without control. This problem is corrected with model 2, which needs an "executioner" (an entity different from the "Events Manager" which can run and cancel the actions of the ICatBeh objects). The third model gives the feature of multitasking, which obviously increases the interactivity of the robot. Moreover model 3 does not need any "executioner".

THE CAT COMMANDS LANGUAGE (CCL)

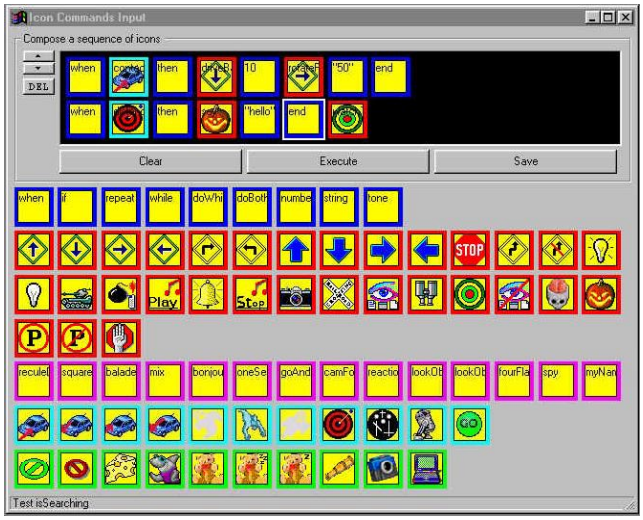

Figure 6: Icon interface of AMAELIA

The cat command language makes it possible to order the robot cat quite simply. The user just has to compose a sentence and to validate it.

This sentence can be written either with the words of the language, or with command icons. These two languages (text and icons) are completely equivalent, that is to say each word of the language text matches an icon.

Basic commands

The basic commands form the primary bricks of the language of the cat. They are the basic orders of the robot, which match to its basic functionalities such as going straight, lighting the lamp or playing some music, taking a picture or saying something.

Some of these basic commands are followed by some parameters, not optional for most of them.

State icons

The state icons or test icons represent the states of the robot cat. They will be used with the keywords "if", "while" and "doWhile" in order to compose conditional or repetitive behaviors. When these icons are used, they call upon the database of the cat (its base of knowledge, its brain) to know the value of the test and thus to decide the continuation of the behaviour.

The icons of tests are: isHungry, isSleeping, withCam.

Event icons

Event icons stand for events for which the robot cat is sensitive, that is to say all the events it can detect and which are available for the user. These icons will be used with the keyword "when", thus the user will be able to set up specific reaction on some event.

Control icons

The control icons make it possible to structure the cat language in a logical way. It makes it possible to combine basic icons between them, but also to introduce the events and the tests in the language.

CONCLUSIONS

AMAELIA is designed as an extensible application, it is possible to add new actions and events on the existing body components, but it is also possible to add new body components, which have their own actions and events. The input layer is also extensible while the new input can be translated in CCL.

A very good extension for our cat could have been the icons recognition by the Lego camera. Indeed we designed an icon language and we designed a module able to recognize some icons belonging to CCL. In this way we could have a fourth way to command the cat.

REFERENCES

[1] Pascal Vaillant: Sémotique des languages d' icons. Honoré Champion éditeur. 1999

[2] http://neuron.eng.wayne.edu:80/LEGO_ROBOTICS/lego_robotics.html