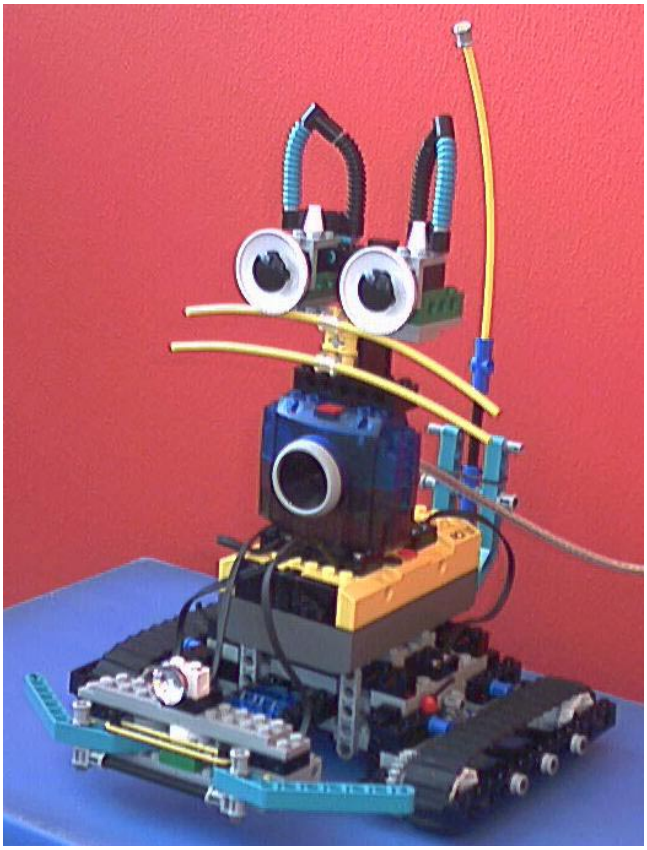
MAELIA

Multimodal Application for Extensible Lego Intelligent Agent



TU DELFT
Project Report, April 1st to August 31st 2002

Guillaume BARRAUD & Priam PIERRET

Supervised by Professor Leon Rothkrantz

0. TABLE OF CONTENTS

1. Acknowledgements & Summary

Above all, we make a point of thanking all researchers and students of KBS department (Knowledge Based Systems) of TU Delft for their reception and their sympathy.

We thank very particularly and very cordially our Master for training course: Leon Rothkrantz for its reception, its availability, and its kindness!

We also thank especially Edwin de Jongh, Matthias Heie, Boi Sletterink, Luca Porzio , Biagio Di Santo, who brought their assistance for the project or our stay to the Netherlands.

Thank you finally to ENSEIRB administration, which helped us for various steps: Mr. Paul Y. Gloess, Mr. Bruno Eymas, … who made possible this training course to Netherlands.

Abstract :

This project deals with Lego robot, communication human-computer, artifical intelligence. The purpose of our project was to built a small robot as a cat agent and to interact with it through different ways of communication.

2. Introduction

We carried out our final project at TU Delft, in the Netherlands, within the framework of the exchange program Socrates-Erasmus. We worked from April 1 to August 30, 2002 within the research laboratory of Knowledge Based System in ITS faculty of TU Delft.

Our mission was at the same time to build a robot according to the model of a cat and to conceive an application making possible to communicate with this robot and also making possible to give him "life".

We have been supervised by Leon Rothkrantz, our professor at TU Delft. This training course had for main objectives :

- To train theoretical and practical knowledge acquired during 3 years at ENSEIRB … in particular skills related to the application design (software engineering).
- To work in an international context: it means to improve our English by using it both for work and everyday life. But also to discover a country and its culture: the Netherlands … but still to meet students coming from the whole world and particularly from the other countries partners of the European Erasmus program (Germany, Spain, Italy, Ireland, Romania, Finland, Belgium …)

These goals were largely reached during these 5 months. This experiment was very instructive and enriching for our future work. This project still accentuated our desire to work in a multicultural and international context … while allowing us to conclude a stimulating project at the technical level.

Note that to complete your knowledge about our work, you can visit this following web site relating all we have done.
http://www.kbs.twi.tudelft.nl/People/Students/G.Barraud/thirdyear/project/project.html?cStyle='kbscolor'

3. Purpose

3.1 Original Subject from Mr Leon ROTHKRANTZ

The goal of the project is to design and implement a Lego robot as a digital animal (pet). There are different ways to communicate with a Lego robot:
-a speech interface, via the connecting PC. On the PC some speech processing tool is implemented. Then you can give some commands as go, stop, go two meters ahead, find the red cookie, find the other robot, explore the labyrinth. When the systems processed the speech correctly, then some commands are via the infrared connector transmitted to the Lego robot. The Lego robot can show some appropriate response.
-some sensors as interface :
When the Logo hits some object, or the infrared sensor is reading some gray value, some behavior response can be triggered.
-digital camera :
The Lego robot has eyes via a simple WEB cam connected to the PC. On the PC some image processing software is running. If some object is recognized, i.e. apply, cup of milk, dog, then again some appropriate behavior patterns are activated.

The Lego robot can show some behavior as driving forward or backward, go left or right, go to some specific place, show some flashlights, turning wheels etc. It is even possible to use speech synthesis on the PC to generate appropriate sounds.

The Lego has some intelligent module to compute the right response based on the right input.
The reasoning module has a limited capacity, but a nice option is to do the reasoning work at the PC and connect with the robot via the infrared sensor.

Once the multimodal interface is designed and implemented two applications can be designed and a prototype of the system can be implemented:

-design the Lego robot as a communicating agent, model the Lego robot as a digital cat. So the user can speak to the cat and the Lego robot shows cat behavior.
-design the Lego robot as a search agent, so that it is able to find his route in an unknown environment and communicate the results to a colleague robot.

3.2 Understanding the problem

This was the subject we read and we chose before arriving in Delft. The first month, while testing separately every possible feature for our project, we understood that it's quite quick to implement an intelligent behavior on the robot cat, but when it works we always want to implement something more intelligent, so how make a project with that? We thought with Mr Rothkrantz that it can be powerful to create an application which offers to the user the possibility of executing intelligent behaviors with the robot, but also which allows to create new behaviors using a text language and an icons language; a kind of AI development studio with a Lego Robot Cat. We enjoyed this idea of creating a whole application, especially because of our specialization in software engineering.

3.3 Specifications of MAELIA

3.3.1 Hardware needed

- Lego RCX Programmable Brick (PBrick)
- Lego output bricks
  - 2 motors for driving
  - 1 lamp for light
- Lego sensors
  - Contact sensor - one in front-left and one in the front-right for collision control
  - Rotation sensor - measurement of distance
- IR transmitter - transmission of IR signal
- Microphone - speech recognition
- Speaker - audio signal (speech, sound) output
- Lego camera - vision for the cat
- Approx. 120 LEGO bricks
- Computer – with both COM and USB ports.

3.3.2 Software specifications

With our supervisor, we outlined the requirements of the application to be designed.

3.3.2.1 Functional Requirements

- Execute behaviors on the robot cat
- Use sensors to have reactive behaviors
- Create a language of behaviors description
- Edit behaviors with text
- Edit behaviors with an icons language
- Save user-defined behaviors for reuse

3.3.2.2 Non-Functional Requirements

- Use the Lego Camera
- Use speech recognition to input commands
- Use speech generation
- Displaying language reference when editing behaviors
- Make the icons input easier to use than the text input

3.4 Constraints

- Robotics Invention system works only with Windows 98.
- Windows NT does not support (by default) USB port, which is used for the Lego camera.

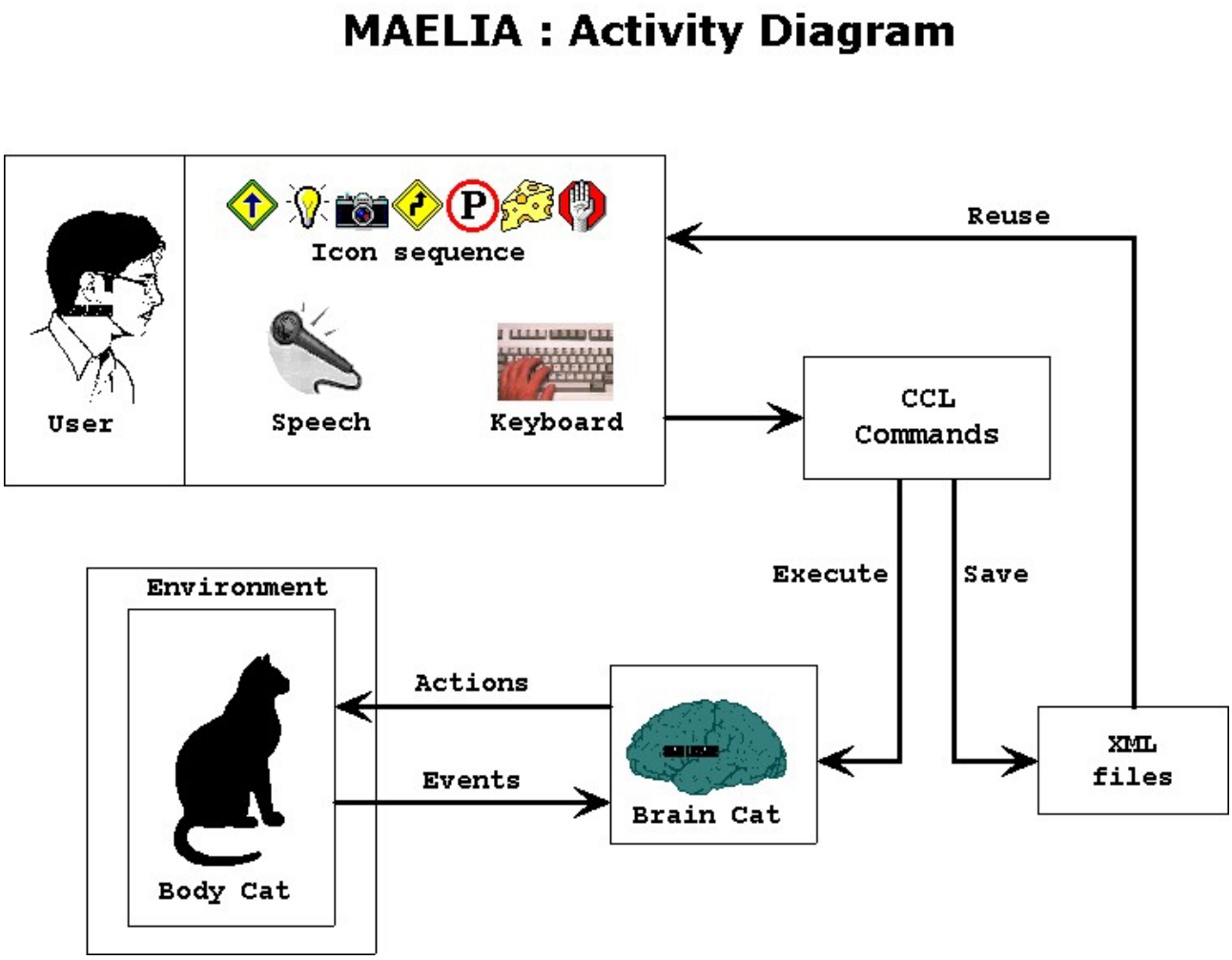
3.5 Anticipated Project Schedule

- Week 1-2: study the Lego tool box, how to build a Lego robot with the tool box, read some manuals, read some application reports
- Week 3-4: study the report on the speech processing, how to design the speech interface
- Week 5-6: study the report on the Lego cam camera, how to recognize objects
- Week 7-8: design a Lego cat,
- Week 9-12: implement the Lego cat (basic behaviors, simple interface)
- Week 13-14: test the Lego cat and make some improvements
- Week 15-17: implement more advanced behaviors and interface
- Week 18-19: test the advanced prototype and make improvements
- Week 20: write the final report

4. Conception

4.1 MAELIA: Global presentation

4.1.1 Activity diagram



4.1.2 AI Aspect

The Multimodal Application for Extensible Lego Intelligent Agent (MAELIA) 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).

4.1.3 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 and you can teach it how to react it is running, and finally everything 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.

4.1.4 Components Aspect

MAELIA has a lot of advanced features like infrared communication, image processing for color and movement detection, speech recognition and generation, etc. To make the development last only five months, we used existing components for most of these advanced features. These components are ActiveX components for Windows operating systems, so we need a programming language, which provided ActiveX dynamic linking. Visual Basic is very efficient for ActiveX reuse, graphic user interface design and quick development, so we used it.

4.2 Application architecture

4.2.1 Description

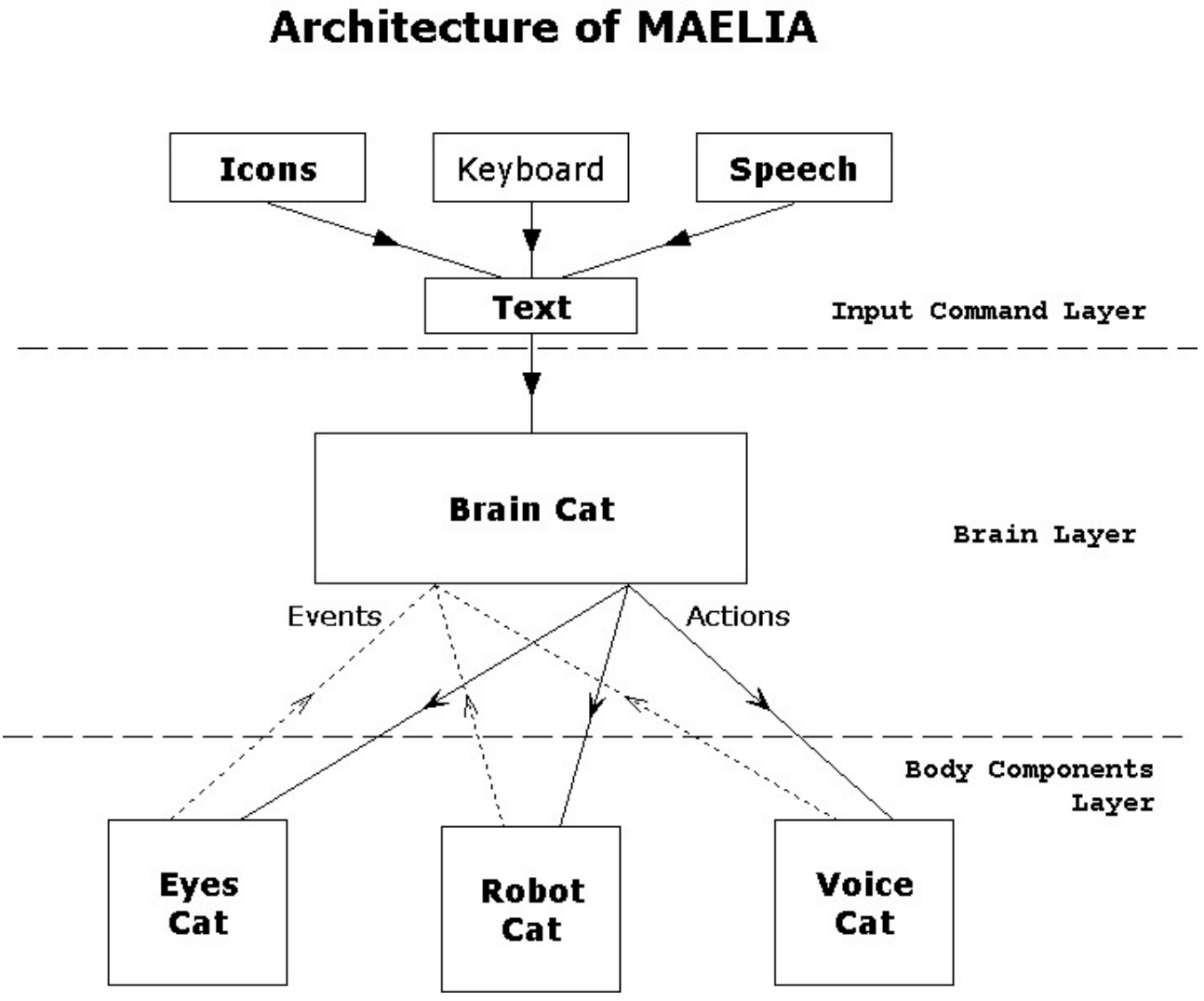
The architecture of MAELIA is designed according to an object-oriented approach; there are seven main entities, all of them embed one or several existing ActiveX components.

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.

4.2.2 Diagram



4.2.3 Responsibilities and Collaboration

4.2.3.1 Body Components layer

The entities in this layer can access to 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...), these events are sent to the BrainCat which is the only entity allowed to trigger actions on the Body entities.

4.2.3.2 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 design as an agent, in the sense of an artificial intelligence context.

4.2.3.3 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.

4.3 Body components

4.3.1 RobotCat

4.3.1.1 Lego Mindstorms System

Few years ago, the Lego Company released a new range of Lego stuff called Lego Mindstorms System. The goal of this new range was to give a practical (Lego) support for robotics development. With this kit you can 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:

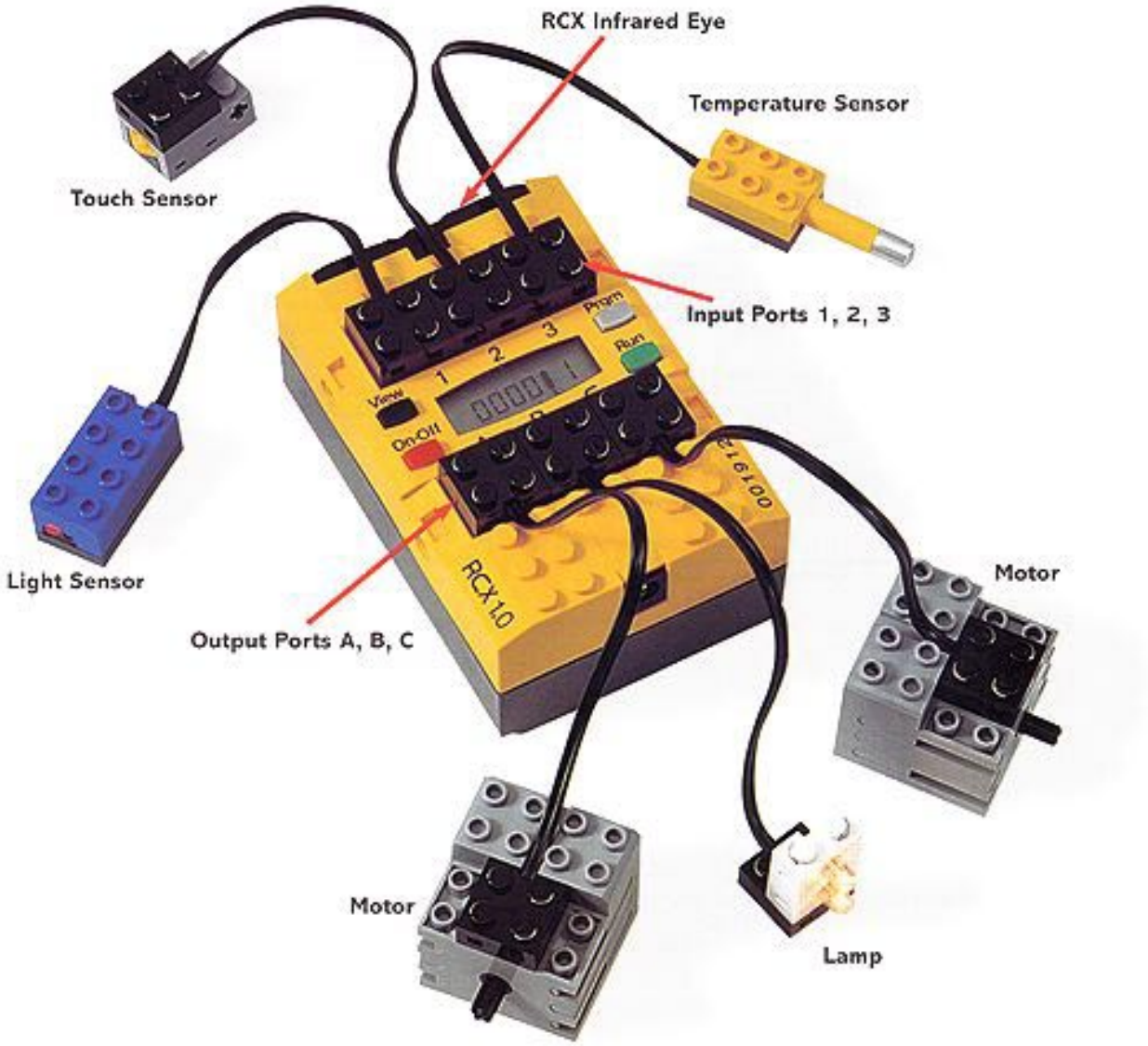
- The PBrick: the main piece, it is a big Lego Brick, with a microprocessor inside, and some inputs and outputs on the top. It can communicate with the PC via an infrared port on the PBrick and infrared tower connected on the COM port of the PC.

- The output bricks: motors, lights.

- The input bricks (sensors): to detect contact, rotation, light and temperature.

- The cable bricks: it is just two small and simple Lego bricks with electrical contacts; it is used to connect a PBrick port to another special brick, for both inputs and outputs.

The box of RIS Lego Mindstorms System



The new bricks of the Lego Mindstorms System.

Infrared communication between PBrick and PC.

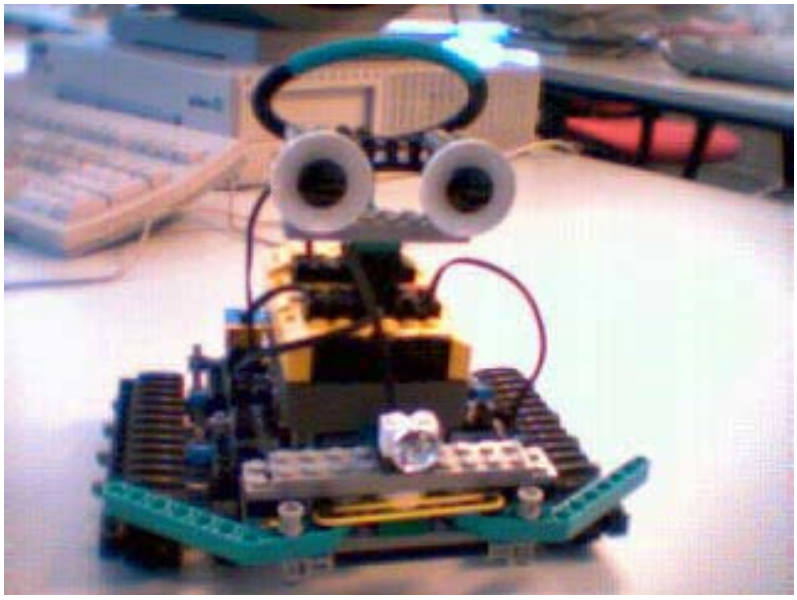
4.3.1.2 Robot Building

First stage of the realization of the robot : its building. It could be almost summarized in only one word: Lego. 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 will be able to note initially is 'no legs' (replaces by caterpillars or wheels). Our robot can nevertheless wear its 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 thus 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 him to measure the distances covered and with two contact sensors placed on the front side of the robot on the bumper which enables him to detect contact with obstacles on the left and on the right independently.



The first robot.

4.3.1.3 Robotics Invention System

A software is provided with the Lego box, it is called RIS (Robotics Invention System). It makes it possible to program simple behaviors with the robots using the RCX brick. This is due to building blocs of language matching the basics behaviors such as starting engine A forward, starting engine B backward...

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 for example. Let us note that this software uses also the Spirit.ocx component, which is described in the next section.

4.3.1.4 Spirit & VB

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 PBrick. You can control the PBrick in two ways, and most of the Spirit methods are available for both ways:

- Direct commands: the action is done on PBrick when the methods is called on the PC
- Downloadable commands: the command is downloaded when the method is called on PC, the command is executed in the PBrick when the program is started

To store downloadable commands, the PBrick has 5 programs slot, each of them can contains 10 tasks and 8 subroutines. Without having the PC, you can choose the slot and run the program on it. We use this mode to make the Mouse running (the Mouse is another Lego robot used only to trigger events on the Cat).

As an ActiveX control, it can be accessed (used) only from a programming language, which provides ActiveX dynamic linking. Common user languages for this are C++ and Visual Basic. We choosed Visual Basic for its advantages of quick development.

4.3.1.5 NQC

An alternative to Visual Basic could have been the NQC (Not Quite C), a programming language especially designed for the RCX based on the syntax of C (from where its name). From a performance point of view, that doubtless would have been the best choice. But this language is compiled into native RCX code executable by the PBrick, and we cannot access to PC resources from it, it is why we did not use it.

4.3.1.6 Java

We could also very well have used Java technology in order to make the interface of control of our robot. Indeed, when researching at the beginning of the project, we found various APIs written in Java, allowing to control the RCX PBrick. These different APIs have the same role as the Spirit component; it plays the role of interface of communication between the RCX and the PC.

Java could have brought a better programming support to us, particularly for object-oriented features and also for safe errors management. However, we did not find any Java API for the Lego Camera. Another significant point is the design of the user interface, so easy with Visual Basic; it would have become really huge with Java.

4.3.1.7 SCRAP

SCRAP was the previous project in the KBS department, which uses a Lego Robot. Mathias Heie, a student from Norway, designed it. It deals with the speech recognition by allowing the user to command movements to the robot from speech. We were inspired by this project during our conception in order to provide a speech interface for our application.

4.3.1.8 Actions and Events

We group the actions into four categories, for details see the CCL reference.

1. Moving actions : driveForward, rotateLeft ...
2. Sounds and music actions : playSound, playMusic.
3. Light actions : setLightOn, setLightOff.
4. System actions : setPowerMotor, set PowerDownTime...

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

4.3.2 EyesCat

The Vision Command box

4.3.2.1 LegoCam & Vision Command

Another part of the Lego Mindstorms System 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 color and movement detection.

The Lego Camera

4.3.2.2 QuickCam SDK, EzVidCap & LCC

The Lego Camera is actually a simple web cam using the standard QuickCam drivers. Logitech (QuickCam provider) propose for free the QuickCam SDK, which is a set of ActiveX libraries and controls. EzVidCap (Ez? Video Capture) 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 color and movement detections according to a layout of detection zones that the developer can define.

4.3.2.3 Hit and Run

Hit and Run was the previous project in the KBS department, which uses the Lego Camera and the LCC ActiveX control. There are two robots moving on a flat square. One robot is moving autonomously, and the other communicates with the PC on which the Lego Camera is connected. The camera is above the square, looking at it. The goal is to make the second robot hit the first one, the camera being the only percept of the second robot. Some colored panels are on each robot, so by using LCC, we can know where is each robot and what are their orientation.

4.3.2.4 LCC Detection

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

Two events can be fired: color and motion, which happen when color or motion is detected (or no longer detected) in a particular zone of the active layer (only one layer can be active at the same time).

4.3.2.5 Actions and Events

We need to specify what how the system will use the camera, which means what are the layers that can be useful for the Robot Cat. The natural 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 missed motors outputs on the PBrick, there are only three, two are used for driving and one for the light. So the camera can only move from left and 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. See section 5.10 for details.

4.3.3 VoiceCat

4.3.3.1 Microsoft Speech API and TTS

In order to give a little more presence to our robot cat, we equip it with the voice. For this purpose, we have to use some classes of the Speech API to develop this small module of speech generation. This module is base on the same model that the other cat commands, it means that the order "say" implements ICatBeh interface.

4.3.3.2 Actions and Events

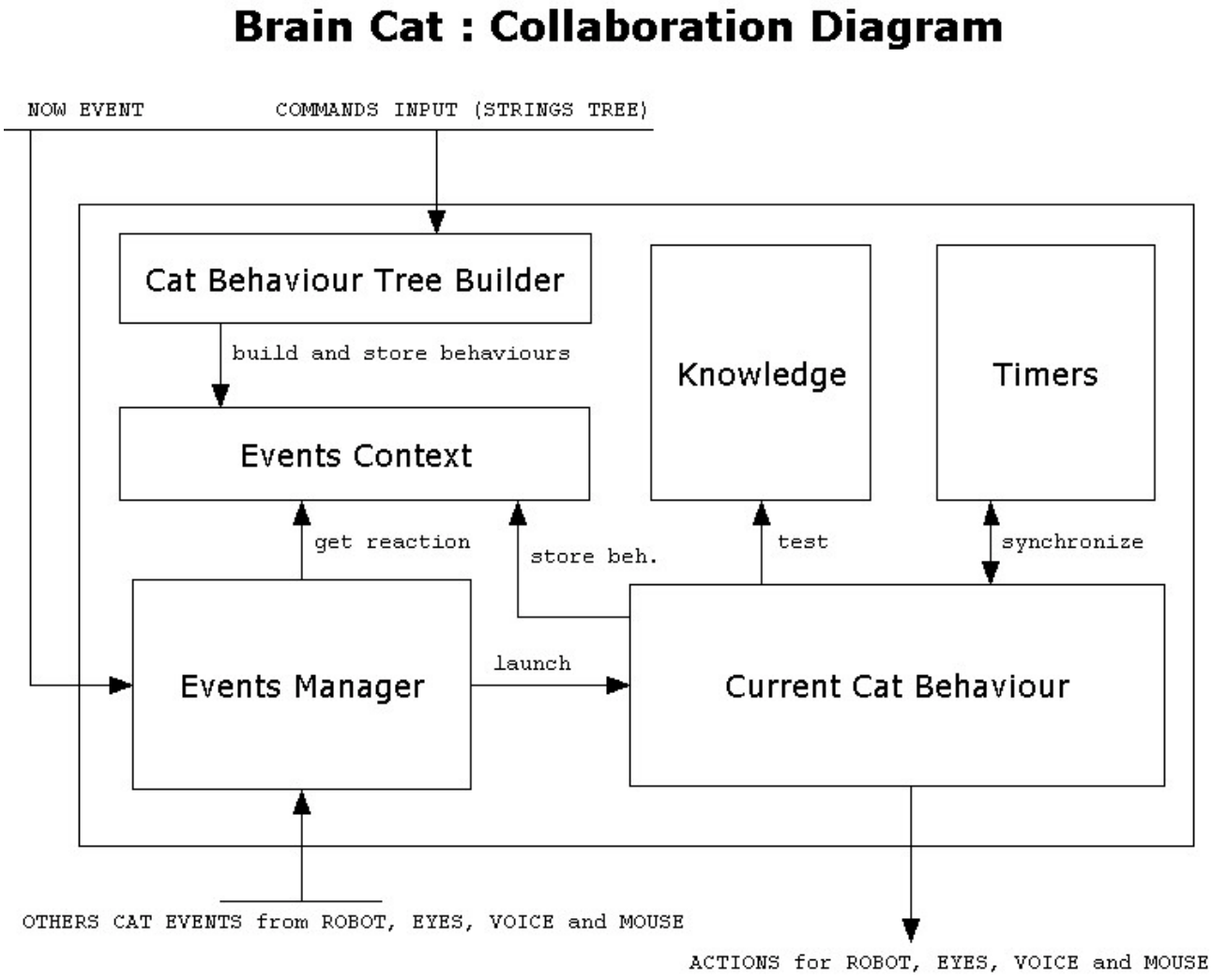
This module adds the action “say” which takes a string parameter. It introduces also the concept of events of beginning and end of word or phrases. But these events are not available for the MAELIA user; they are just used to synchronize the execution (when we want to do something after saying something).

4.4 Brain

4.4.1 Purpose

The Brain is the core of the system, it is in charge of the execution of the commands and it should react when a Cat Event occurs. The Brain is designed as an agent according to the PAGE definition (Percepts, Actions, Goals, Environment).

4.4.2 Collaboration Diagram



4.4.3 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 is the current behavior tree, once the execution started, it has to organize its own internal execution of its nodes. The implementation of the 'Action()' method from should calls specific actions on Body components.

- The Events Context stores all the event-reaction couples. While executing, the user can change these couples using special control command.

- The Knowledge object is the knowledge base of the system, inside are stored variables of different types which reflect the spirit state of the cat (combination of booleans), and also some integer and string values used for events and actions.

- The CatBehTreeBuilder manages the input command 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, which is discussed in section 4.7.4.

- 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 is not receive (which can happens when the robot goes too far and loses infrared contact.

4.4.4 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 thus offers to us both a common interface for all the words of the language, the ICatBeh interface. But also a structure for the uttered then executed sentences.

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

- 3 conditional commands: if, while, doWhile.
- A 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"

And however, from a certain point of view, all the orders are equivalent and in same time, some control commands may contain other basic orders or control itself.

And the customer would like to process all orders in the same way.

Then we consider the basic orders like leaves of the tree of the sentence and the nodes of the tree are the icons controls.

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.

Let us 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).

Also let us note that we are also inspired by the Interpreter design pattern to build the control icons and integrate them in the tree of language.

4.4.5 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 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 to 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 these two requirements:

- R1 : Execute an action not before the previous action in the sequence is finished.

- R2 : Execute a new sequence when an event occurs.

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

I. '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 user level (in the Cat Command Language)
II. '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 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 made us 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 the model 3 does not need any “executioner“.

4.4.5.1 Model 1 : synchronous, reentrant, monotask

This model is said to be synchronous because we use function calls to synchronize the execution. In this way, we consider that the concrete action is finished when the call of the method ’Action()’ is done, so if we have a control command like ’sequence’ , we just have to call Action() on every child in the prefix order, and requirement R1. will be completed.

For immediate command, this model is the natural way of implementation, but for during commands, we need the method ’Action()’ to last the time of the physical action, so we need to know when the physical action is done.

For example, to implement the driveForward command with a distance parameter, the Action() method does the following:

1. Send the parameter from the PC to the PBrick.
2. Start the moving task on the PBrick.
3. Wait until we receive the ’end of moving task’ event from the PBrick
4. Finish the call to trigger the following actions.

Question: How to listen to events if the execution flow is waiting (sleeping) in the call of Action() ?

First idea: using two threads; the first thread for events listening is running the event loop and when an event occurs, a new thread is launched with the corresponding Action() method. When the end of task event arrives, the event loop (first thread) notifies the second thread to finish this action and to run the following one.

Problem: the Visual Basic standard API does not support any threads implementation. The only way to use threads is to make dynamic linking with the Win32 library, but there are some restrictions to respect the OLE model (shared memory between thread is not possible).

Fortunately, Visual Basic offers alternatives for threads, these features are not common in classic programming, but they can be very powerful.

VB programming is event based programming, it means that the VB compiler takes care of the events loop, the developer has just to add some code for the events he wants the application to react. When an event callback function is running, the event loop cannot run because the application is monothread, but VB provides a way to manage events while running the code of the developer: there exists a function called DoEvents which manages all the events waiting in the events queue before give the execution flow back to the code of the developer. This is useful to not freeze the user graphic interface while running a code, which need several seconds of execution.

Thus, for the model 1 and 2, every during action is calling DoEvents while waiting for ’end of task’ notification. In this way the requirement R1 is completed.

Model 1 is said reentrant because when an event occurs, the method ’Action()’ of the reaction is called inside (via the DoEvents call) the call of the method ’Action()’ of the previous action, which is interrupted and cancelled. In this way requirement R2 is completed.

From a system point of view, this is a real problem because the system call stack is growing without control. It is why we change the model to correct this problem.

4.4.5.2 Model 2 : synchronous, organized, monotask

To complete requirement R1, model 2 is the same than model 1. The difference is in the way to launch a reaction when a 'Cat Event' occurs.

Model 2 is not reentrant, the call of 'Action()' for the reaction is done outside the call of 'Action()' of the behavior that was running when the event occurred. Model 2 is said to be organized because an entity (the 'Executioner') is responsible to launch the reaction after an event occurred. In model 1, the event notification launch the reaction, in model 2, the event notification sets the next reaction to execute and cancel the current behavior, then the execution flow comes back to the Executioner which can now launch the reaction at the same level (in the call stack) than the previous behavior. To make the model uniform, we launch the initial behavior using an artificial event: the DoNow event.

4.4.5.3 Model 3: asynchronous, multitask

Once model 2 has been implemented and tested, we could begin to define some intelligent behaviors with reaction on events, the first example was the "stroll", a behavior that makes the robot driving forward and avoiding obstacles. It was working well, but a question comes: how to make the robot playing music while it is driving forward? In another way, how to make the robot doing two (or more) things at the same time? In the models 1 and 2, because the actions are called synchronously, it is not possible to have two actions running at the same time (with only one execution flow). So we needed to change the model again.

We want to keep the idea that when a tree is executed, it is responsible of its internal execution; more exactly a parent node is responsible to execute its children. So now in the model 3, when we call 'Action()' for the first time, it launches the physical action on a body component, and then the call returns. This is enough for immediate commands, but for during commands, when the 'end of task' event occurred, the action() method is called again to make the command notifying its parent node that its action has just finished, in order to the parent to call the next child.

With this asynchronous model, all actions calls are very quick, so the application is more reactive (the event loop is running most of the time). Moreover, we introduced a new control command 'doBoth' which takes two behaviors to be executed at the same time. And finally when an event occurs, the reaction is always launched but the previous behavior is not stopped and cancelled, we just keep it running if it is possible. For example, if we want the robot to react to contact while it is driving, for music reaction both actions will be executes at the same time, but for another driving reaction, the reaction will overload the previous action. In this ways we define several exclusives categories of actions (see Implementation section for details).

4.5 The Cat Commands Language (CCL)

The cat command language makes it possible to order the robot cat quite simply. You just have 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 that to each word of the language text matches an icon, but we will explain this part a bit further, see chapter 4.6.2.

4.5.1 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 the most part of them.

Available basicCmd list :

Id	Image	Source	Version	lastModified	Param (optionnal)	Unit
driveForward		robotCat	1.0	10-05-02	distanceInCm	centimeters
driveBack		robotCat	1.0	11-05-02	distanceInCm	centimeters
rotateRight		robotCat	1.0	11-05-02	rotationInDegrees	degrees
rotateLeft		robotCat	1.0	11-05-02	rotationInDegrees	degrees
turnRight		robotCat	1.0	29-05-02	none	none
turnLeft		robotCat	1.0	29-05-02	none	none
driveForwardDuring		robotCat	1.0	25-05-02	timeInSeconds	seconds
driveBackDuring		robotCat	1.0	25-05-02	timeInSeconds	seconds
rotateRightDuring		robotCat	1.0	25-05-02	timeInSeconds	seconds
rotateLeftDuring		robotCat	1.0	25-05-02	timeInSeconds	seconds
stopMoving		robotCat	1.0	12-05-02	none	none
startStroll		robotCat	1.0	29-06-02	none	none

endStroll		robotCat	1.0	29-06-02	none	none
setLightOn		robotCat	1.0	15-05-02	none	none
setLightOff		robotCat	1.0	15-05-02	none	none
setPowerMotor		robotCat	1.0	31-05-02	powerMotor	none
setPowerDownTime		robotCat	1.0	01-06-02	powerDownTimeInSeconds	seconds
playMusic		robotCat	1.0	02-06-02	toneSequence	none
playSound		robotCat	1.0	05-06-02	soundName	none
stopMusic		robotCat	1.0	02-06-02	none	none
takePicture		eyesCat	1.0	04-07-02	none	none
doNothing		pcCat	1.0	19-06-02	none	none
watchLeftMiddleRight		eyesCat	1.0	11-06-02	none	none
watchSymbol		eyesCat	1.0	12-06-02	none	none
watchTarget		eyesCat	1.0	13-06-02	none	none
stopWatch		eyesCat	1.0	13-06-02	none	none
autonomous		pcCat	1.0	08-08-02	none	none
say		pcCat	1.0	14-06-02	stringToSay	none
wait		pcCat	1.0	08-06-02	timeInSeconds	seconds
stopWait		pcCat	1.0	08-06-02	none	none
stopAll		pcCat	1.0	16-06-02	none	none

4.5.2 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 data base of the cat (its base of knowledge, its brain) to know the value of the test and thus to decide the continuation of the behavior.

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

Available test list.

Id	Image	Source	Version	lastModified
true		pcCat	1.0	25-05-02
false		pcCat	1.0	25-05-02
isHungry		pcCat	1.0	04-06-02
isHappy		pcCat	1.0	04-06-02
isTired		pcCat	1.0	05-06-02
isSleeping		pcCat	1.0	05-06-02
isSleepy		pcCat	1.0	05-06-02
isSearching		pcCat	1.0	05-06-02
withCam		pcCat	1.0	07-06-02
withPC		pcCat	1.0	07-06-02

4.5.3 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 you will be able to set up specific reaction on some event.

Available event list :

Id	Image	Source	Version	lastModified
contactLeftPushed		robotCat	1.0	08-06-02
contactLeftReleased		robotCat	1.0	09-06-02
contactRightPushed		robotCat	1.0	10-06-02
contactRightReleased		robotCat	1.0	11-06-02
seeSmthOnLeft		eyesCat	1.0	08-06-02
seeSmthOnMiddle		eyesCat	1.0	08-06-02
seeSmthOnRight		eyesCat	1.0	09-06-02
distanceTarget		eyesCat	1.0	19-07-02
symbolRead		eyesCat	1.0	15-07-02
RCXMouseMessage		pcCat	1.0	21-07-02
doNow		pcCat	1.0	09-06-02

4.5.4 Control icons

The control icons make it possible to structure the cat language in a logical way. It make it possible to combine basic icons between them, but also to introduce the events and the tests in the language.
To have a look at the exact syntax of this grammar, see the gold grammar, chapter 5.4.

4.5.4.1 « repeat » icon

The "repeat" keyword makes it possible the cat to make several times the specified action in this sentence.

Its syntax :

```
<repeatCmd> ::= repeat <numTimes> <sequenceCmd> end
```

An example :

```
repeat four times driveForward 20 cm, rotateLeft 90 end
```

4.5.4.2 « if » icon

the « if » keyword makes it possible to have conditionnal orders.

Its syntax :

```
<ifCmd> ::= if <booleanTest> <thenToken> <sequenceCmd> end
          | if <booleanTest> <thenToken> <sequenceCmd> else
<sequenceCmd> end
```

An example :

```
if isHungry then startStroll end
```

4.5.4.3 « while » and « doWhile » icons

With the « while » keyword, you can order the robot cat to repeat the specified action as long as the test is not true.

Their syntax:

```
<whileCmd> ::= while <booleanTest> <sequenceCmd> end

<doWhileCmd> ::= do <sequenceCmd> while <booleanTest> end
```

An example :

```
while isHappy then rotateRight 10 end
```

4.5.4.4 « when » icon

The key word "when" makes it possible to specify reactions to him. A sentence beginning with the key word when does not start actions. It does nothing but set up reactions.

Its syntax:

```
<whenCmd> ::= when <eventToken> then <sequenceCmd> end
```

An example :

```
when contactLeftPushed then driveBack 20 end
```

4.5.4.5 « doBoth » icon

The "doBoth" keyword makes it possible the cat to make two actions at the same time.
Its syntax :

```
<doBothCmd>    ::= doBoth <sequenceCmd> and <sequenceCmd> end
```

An example:

```
doBoth driveForward 120 and playSound 2 end
```

4.5.5 Complex commands

Because of the save function commands sequence, any sentence stated with the cat can be recorded and re-used thereafter. We will call complex order, any sequence of orders saved by our application and then being able to be recalled with its identifier (its name).

- A complex order is thus a combination of basic orders, control commands and possibly others complex orders defined beforehand.
- A complex order is considered in grammar as a basic order, it can for example be used within control commands.
- A complex order is entirely defined by its ID (its name), which was given to him at the time of its save. Because of this name, it could be recalled in the language.

4.6 Commands Input Interfaces

4.6.1 Text Input Interface

The Text Input Interface is responsible of the interpretation of the Cat Command Language. It owns a text parser compliant with the CCL grammar, which builds an StrTree from a CCL phrase. It presents a text input window with commands for executing, saving and displaying help.

4.6.1.1 Parsing text into a behaviors tree

Parsing text is not an easy feature to implement from scratch, fortunately there exists a lot of existing freeware parsers available as programs, static library, dynamic library, and also ActiveX components, which are easy to use in Visual Basic. We chose the GOLD Parser because it is more than a parser, it is also a grammar checker and compiler. The parser is an ActiveX Dynamic Linking Library, which loads a compiled grammar at runtime, and the GOLD Parser Builder is a program to edit in clear text your grammar, to check it, and to compile it for the GOLD Parser component.

4.6.1.2 Need of language reference

Obviously it can be useful to have permanently access to language reference. It is done with the Help Browser, which displays the user manual. The reference is organized according to the type of the elements of the CCL: controls, commands, events and tests. Notice that the help is dynamic, it means that the user-defined behaviors are documented as soon as they are saved, this is done by generating at runtime HTML documentation from XML Cat Identity files using XSL transformations.

4.6.2 Icons Input Interface

4.6.2.1 Icons, Symbols, Pictograms...

In order to complete our multimodal interface, we decided to associate a pictogram for each brick of behavior and thus to create a second language similar to the language of the cat (CCL) but using icons instead of words.

The book of Pascal Vaillant, "Semiotique des langages d'icones", informs us in detail about this subject. We learned not to confuse all the symbols: icon, ideogram, letter or pictogram and also that the concept of icon is based on the resemblance to the represented object. In this direction, it is different from ideogram which is a sign belonging to a writing system. The ideogram is defined especially by opposition to the letter like writing symbol. The ideogram does not suppose anything more, and in particular no iconicity. An ideogram that is also iconic is called a pictogram.

We thus can separate the symbols in three categories:

- The icons (tables, signs, statues)
- The ideograms (hieroglyphs, Chinese ideograms, symbols of the electric schemas)
- The pictograms (which form also part of the two last categories)

Our language uses icons-pictograms because each icon is carrying a sense.

For that, we complied with certain rules:

- A color is selected for each type of icons in order to facilitate reading of a sentence:
  - Yellow for the basic commands
  - Green for the events
  - Red for the tests
  - Purple for the complex commands
- A pictogram is selected for each entity language (its reference is stored in the file which lists all the words of the language). This pictogram is attempting to evoke the word with which it is associated.

4.6.2.2 User feedback

To justify the icons input, it needs to be a faster way of input than the text input. The idea is to presents a palette of valid icons, and by clicking on them, the user put a new icon in the icons sequence, at the current place of the cursor. The cursor can move using the arrow keys. We also need a command to delete an icon from the input. Once the icons sequence is finished, it is translated into text, which has to be a valid phrase of the CCL, and then the text is sent to the GOLD Parser to built an StrTree for the Brain.

4.6.2.3 Some models of icons representation for the CCL

It was not obvious to find an icon representation for the CCL. The main point was how to translate the parameters of the commands and the markers for controls (then, else, end ..). For the two points we hesitated between making them icons or part of the icon (command or control) which owns them. Finally, to implement the user graphic interface, it was easier to make them complet icons, but in this way the risk of bad composition of icons is increased, which is not a real problem because the user is alerted by the parser that the input is grammatically not correct.

4.6.3 Speech Input Interface

4.6.3.1 Speech Recognition with the Microsoft Speech API

In order to command our robot by voice, we install the Speech API on our application. The function of the Speech Recognition (SR) engine is to take a digitalized speech signal as input and convert that into recognized words and phrases of the CCL. The incoming speech is compared with a predefined grammar.

The SAPI provides a high level interface between an application and speech engines. The SAPI notifies the application when a speech event occurs. A speech event might be for instance the start of a phrase, the end of a phrase or a speech recognition. Microsoft SAPI 5.1 can handle more than 30 kinds of such speech events. In this system our primary interest is speech recognition events. When an event occurs, the application will be notified and it receives a structure with information about the event. Because our grammar is built up hierarchically, the structure is typically also hierarchically built up.

The application needs a recognizer object to access the SAPI. There are two ways to set up this object:

- Shared resource instance. This set up allows resources such as recognition engines, microphones and output devices to be used by several applications on the same time.

- Non-shared resource instance. This set up allows only one application to control the resources.

The shared resource instance is the preferred option for most desktop applications. In this way more applications can use the microphone. Therefore shared resource instance is used in this system.

Initially the speech recognition uses a default voice profile, which perform fair for every voice. It is possible to configure the speech recognition system to a specific voice. By doing this the performance is supposed to increase for that specific user. But you also need to have a very good accent in the recognized language (English in our project), this point gave us many problems because of our bad English accent.

4.6.3.2 Grammar or not Grammar?

There are two main types of grammar:

- Dictation grammar: In this type of grammar the SR engine will load its own grammar. The grammar will then typically be very comprehensive. Ideally all allowed phrases in a language could then be recognized. This leads to a huge number of possible phrases. The advantage of dictation grammar is that (theoretically) all legal phrases can be recognized without having to specify the whole grammar first. The disadvantage is that chances of misrecognition will be very high, at least with today's technology. The term misrecognition is the case when a phrase is being recognized as a different phrase.

- Command-and-Control grammar: In this type of grammar we specify our own grammar for the application. The grammar can then be drastically limited. Only phrases that make sense in our field of application would be included and the chances of misrecognition would be strongly reduced. The disadvantage is that the grammar has to be manually specified.

MS Speech SDK supports both dictation grammar and command-and-control grammar. Because our field of application is quite limited, command-and-control grammar is chosen.

There are two types of command-and-control grammar:

- Static grammar: In this type of grammar the grammar is completely predefined and loaded at execution of the application. It is not possible to change any of the rules during runtime. However, it is possible to achieve a kind of pseudo dynamic grammar by specifying all necessary rules and only consider speech events by the rules that should be active at the moment. But sometimes this is not enough, for instance when the user is allowed to specify words or names that are not known at development time. It would be possible for the user to change the contents of the grammar file before execution, but this is not an elegant solution. During runtime it may not be possible at all.

- Dynamic grammar: In this type of grammar the contents of the grammar can change during runtime. By allowing this, we solve the problems of static grammar. MS Speech SDK supports dynamic grammar (and of course also static grammar).

For the constant part of the CCL, we know at design time what phrases we want to be recognized, so there is no need for a dynamic grammar. Therefore our system uses static grammar.

The grammar for Microsoft Speech SDK is a context-free grammar written to a file in XML format.

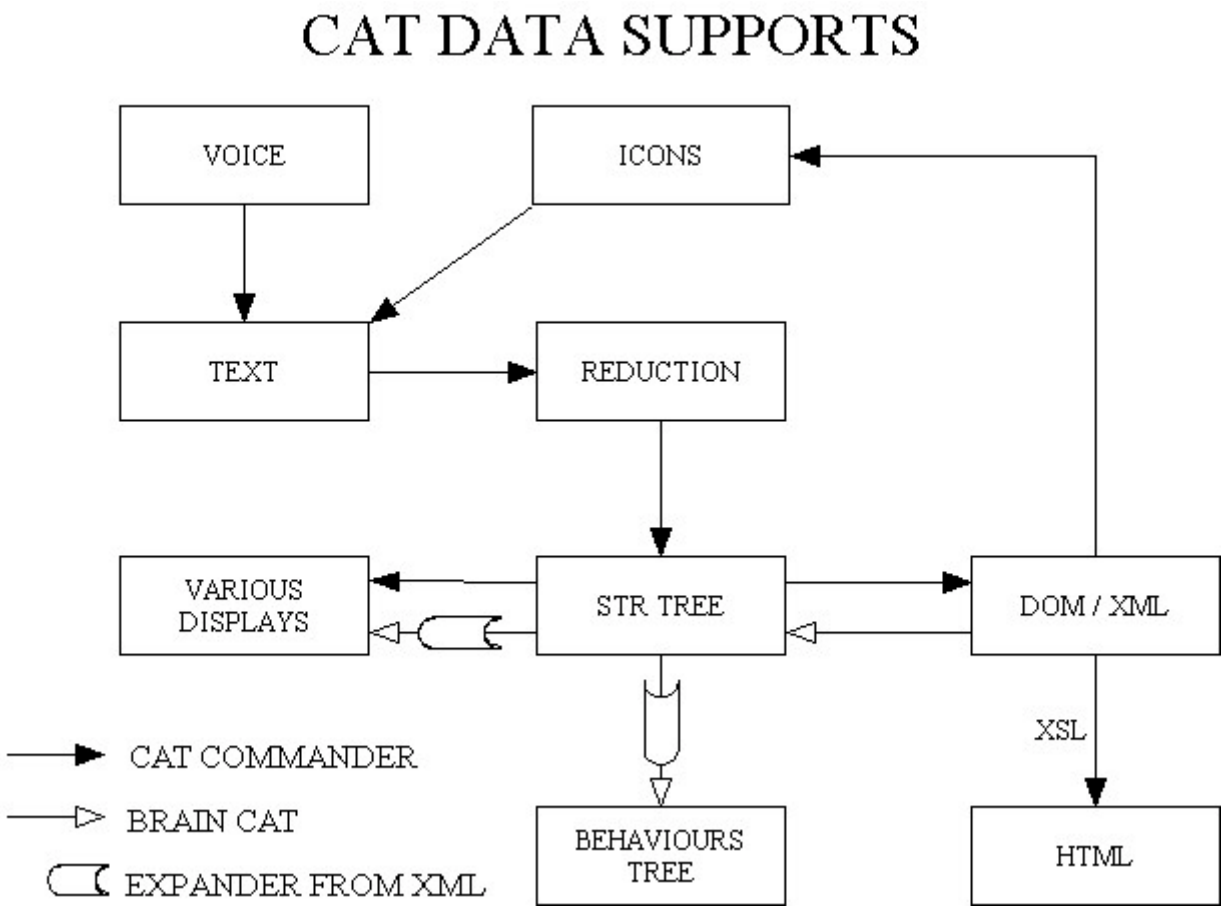
4.6.3.3 The model from SCRAP

The SCRAP application (see section 4.3.1.6) uses a static grammar because all valid phrases and words are known at design time (the SCRAP language is not extensible). It gave us a good support for our grammar, because some complex things were already done, like how to translate numbers written with letters into numbers written with figures. It gave us also a first model to post-process speech recognition.

4.6.3.4 Limitations for the CCL

However, the Speech Recognition does not give good results in our case. Actually the language cat (CCL) is strongly extensible, which obliges to rewrite grammar each time we add words, or to choose dictation grammar, but in this case, the probability of misrecognition is very high. However, combined with the voice synthesis, that opens to us the doors of the vocal dialogue with the robot.

4.7 Supports for the CCL



4.7.1 Reading: Reduction

The GOLD Parser uses a class called 'Reduction' to store the grammatical tree of the text input. Each contains the current rule and the text just parsed. When errors occurred, the reduction is still available, and error informations are stored in the corresponding nodes.

4.7.2 Exchanging: StrTree

The Reduction is not very easy to manipulate, it is why we chose to implement a simple tree structure called StrTree like tree of strings. Each node contains only one string and a collection of children. This structure is used to transform Reduction into ICatBeh, and Reduction into DOM, it is also used for various displays as text phrase, text tree, graphic tree display, etc.

4.7.3 Executing: ICatBeh

The ICatBeh tree is the executable support for the basic commands of the CCL; there is no ICatBeh support for the complex commands (user-defined commands) because the complex commands are expanded into StrTree of basic commands before ICatBeh building. This expansion is done from the XML files, which store the user-defined commands.

4.7.4 Saving: XML & DOM

In order to store some complex commands (sequence of commands), we used the XML language to describe, identify, preserve and restore more complex behaviors bringing into simple orders (or even of other complexes).

We chose format XML because it brings many tools to us for checking, structuring and preserving data in files. The advantages compared to a simple text file are, for example:

- Structure of the language with XML tags.
- Possibility to check if the document is well formed.
- Possibility to check, with the XML parser, the agreement with the DTD, which is the structural model of any XML file.
- Direct (human) reading in the file in an easy way.
- Multiple possibilities of displays with the XSL style sheets.

Indeed, we needed also a table of the symbols (tables of the tokens) for our compiler, in order to preserve the words of the source language (and if possible apart from the source code). Thus the cmdTokens.xml file stores all the information relating to the words of the language.

In addition, in order to satisfy the possibility of preserving certain complex behaviors, we wrote a DTD (Document Type Definition) for XML files describing the language of the complex orders. This DTD is the grammar for these XML files that will have to respect it. Each sequence of orders is saved in an XML file, which bears its name (one gives a name to this sequence of order when it is saved). Afterwards the saved complex command can be reused constantly.

4.7.5 Documenting: XSL & HTML

And for fun, and also to have a good presentation for the help, and also to have a unique XML source file, the same source for all components (Help and Application), we use XSL style sheet language to display XML content in HTML format, on a browser like Internet Explorer (Netscape, Mozilla et Opera does not support XSL).

5. Implementation

The sections in this chapter are described in the chronological order of implementation.

5.1 Programming in Visual Basic...

5.1.1 Basics

Microsoft Visual Basic is a programming language for Windows operating systems. It provided a powerful development studio, and compiles your source code with the Visual Basic framework. This framework gives to Visual Basic a event-based programming style. Moreover the development studio makes you create user graphic interfaces very quickly, and the reuse of ActiveX components is highly facilitates.

5.1.2 Object-oriented features

Visual Basic is not an object-oriented language. It has the concept of classic module, class module and forms (which is a class module with a graphic window). One strange feature is the possibility to execute code while getting or setting a public property of an object. There is no possibility of inheritance, but the concept of interface exists. A class module can implement one or several interfaces, and polymorphism is provided for function parameters but not for resolution of methods name and properties.

5.1.3 Visual Basic not so good

However Visual Basic give us many problems.

Indeed, several aspects of the programming in Visual Basic do not respect the concepts of oriented object programming. Visual Basic is very few adapted for the object design, even if it can do it in theory and in practice, that is not really useful because few object-design-relative tools are accessible in Visual Basic.

Only a concept of interface is available, but polymorphism is also not him not completely supports because typing in Visual Basic is static and not dynamic as in Java or SmallTalk for example. Concretely that adds much lines of code and many objects.

So VB is a pretty toy.

5.2 RobotCat

5.2.1 Spirit features

The ActiveX component Spirit.ocx allows our program to communicate with the RCX PBrick, via the COM port and the infrared transmission. Its interface has a lot of methods, which can be direct or downloadable commands, sometimes both. There methods for outputs, sounds, memory manipulation, etc. When using downloadable commands, we use the Spirit interface like a programming language with control structures; for example, consider this part of code:

```
PB.BeginOfTask tStroll
    PB.On "02"
    PB.Loop srcConst, 0

       PB.If srcSensorValue, iContactLeft, compEqu, srcConst, 1
            PB.SetVar va7, srcConst, iContactLeft
            PB.GoSub sBackTurnDrive
        PB.EndIf

    PB.EndLoop
PB.EndOfTask
```

Here you can see some method call BeginOfTask, If, Loop, which are controls structure for this downloadable program. When this code is executed, nothing is done on the robot. Actually, when the BeginOfTask is called, all the following instructions are considered to be downloaded (some of them could be direct commands otherwise, like On), so they are buffered until EndOfTask is called, and at that time all the buffered program is downloaded on the PBrick, but not executed. To execute it, we have to call ' PB.StartTask tStroll '. This is the way to execute code on the RCX. Our system is using two main tasks:

- tComplexMove: this task is used to make a finite movement like driveBackward 20 cm or rotateLeft 34 deg. The parameters are the directions of each motors (it is why it is the same task for driving and rotating) and the number of rotation steps to complete before stopping (it needs the rotation sensor).

- tEventManaging: this task is called the system task because it is started just after download, and it is responsible to send information about the PBrick to the PC, like the changes of state for contact sensors, and also, which is very important to synchronize actions on the PC, it tells the PC when the tComplexMove task has just finished (the is the source of the end of task event for the robot moves).

We also use some subroutines (it like tasks but subroutine are called synchronously and tasks asynchronously) but the RCX has no call stack so a subroutine cannot call another subroutine.

5.2.2 Interface

We decided to put all features of the RCX robot together, behind a unique interface to have a unique communication object.

This interface contains all features of the RCX like movements, music, light… but it can also launch events from the RCX like contactLeftPushed or endOfTask (let us note that the first event is accessible by the user and the second, not).

This interface is called IRobotCat.

It allows us to have a CRCXRobotCat class, which represents the real robot and a second class CFacticeRobotCat which allow us to test our application at home without the physical robot. Both classes implement the IRobotCat interface.

5.3 BrainCat

5.3.1 Classes

- 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 is the current behavior tree, once the execution started, it has to organize its own internal execution of its nodes. The implementation of the 'Action()' method from should calls specific actions on Body components.

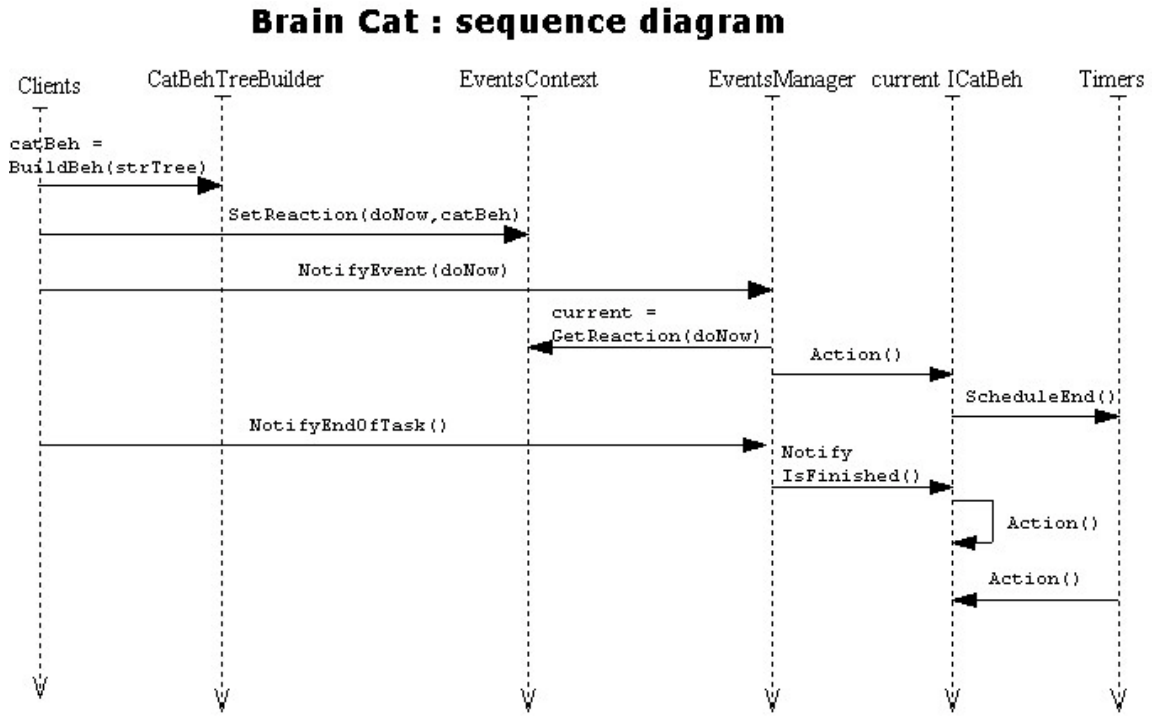
- The Events Context stores all the event-reaction couples. While executing, the user can change these couples using special control command.

- The Knowledge object is the knowledge base of the system, inside are stored variables of different types which reflect the spirit state of the cat (combination of booleans), and also some integer and string values used for events and actions.

- The CatBehTreeBuilder manages the input command 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, which is discussed in section 4.7.4.

- 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 is not receive (which can happens when the robot goes too far and loses infrared contact.

5.3.2 Sequence Diagram



5.4 CCL Grammar & GOLD Parser

5.4.1 CCL Grammar in BNF representation

Here we present the BNF representation of the grammar of the Cat Commands Language, as we wrote it for the GOLD Parser Builder, the grammar checker and compiler for the GOLD Parser engine.

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!! Grammar Properties !!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

"Name"     = Cat Command Grammar
"Author"   = Priam PIERRET & Guillaume BARRAUD
"Version"  = 1.0.0
"About"    = Basic Grammar for Cat Command Language

"Case Sensitive" = True
"Start Symbol"   = <sequenceCmd>


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!! Characters Sets Definition !!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

{String Ch} = {Printable} - ["]


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!! Terminals Sets Definition !!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

ID            = {Letter}{AlphaNumeric}*
IntegerNumber = {Digit}+
FloatNumber   = {Digit}+'.'{Digit}*
StringLiteral = '"'{String Ch}*'"'
Unit =
meter|centimeter|degree|second|minute|hour|meters|centimeters|degree
s|seconds|minutes|hours|m|cm|deg|sec|min|h


!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!! Grammar Rules !!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!

<sequenceCmd> ::= <simpleCmd> ',' <sequenceCmd>
                | <simpleCmd>

<simpleCmd>   ::= <controlCmd>
                | <whenCmd>
                | <definedCmd>
                | <defParamCmd>

<whenCmd>     ::=  when <eventToken> <thenToken> <sequenceCmd> end
```

```
<controlCmd>  ::= <ifCmd>
                | <repeatCmd>
                | <whileCmd>
                | <doWhileCmd>
                | <doBothCmd>

<ifCmd>       ::= if <booleanTest> <thenToken> <sequenceCmd> end
                | if <booleanTest> <thenToken> <sequenceCmd> else
<sequenceCmd> end

<repeatCmd>   ::= repeat <numTimes> <sequenceCmd> end

<whileCmd>    ::= while <booleanTest> <sequenceCmd> end

<doWhileCmd>  ::= do <sequenceCmd> while <booleanTest>

<doBothCmd>   ::= doBoth <sequenceCmd> and <sequenceCmd> end

<definedCmd>  ::= ID
                | ID <cmdParam>
                | ID params <seqParam> end

<defParamCmd> ::= defParam ID

<eventToken>  ::= ID

<booleanTest> ::= ID

<thenToken>   ::= then
                |

<numTimes>    ::= once
                | twice
                | IntegerNumber times
                | IntegerNumber

<cmdParam>    ::= <number>
                | <number> Unit
                | StringLiteral
                | ID

<seqParam>    ::= <cmdParam> , <seqParam>
                | <cmdParam>

<number>      ::= IntegerNumber
                | FloatNumber
```

5.4.2 GOLD Parser Builder & Compiled Grammar

After editing the grammar in clear text, the GOLD Parser Builder compute the DFA (Deterministic Finite Automate), which accept the CCL language, and also the LALR table (Look Ahead Left Recursive), which is used to choose the current accepting, rule. The compiled grammar is actually a file containing the DFA and the LALR Table.

5.4.3 GOLD Parser Engine & Reductions

Once you have the compiled grammar, you can load it in the GOLD Parser engine (which is an ActiveX Dynamic Linking Library) at runtime, but is not possible to change the grammar at run-time because it needs to be recompiled, and there is no component to do it your own program.

The parsing can be done step by step (it means you can interpret the results step by step) or you can do it all and then browse the Reduction structure that represents the grammatical tree of the input sentence. We preferred the second solution because when an error occurs from a badly formed input sentence, we don't want to interpret the results, it is why we do the parsing until an error appears, and if not we build an StrTree from the Reduction.

5.5 XML & DOM

We easily choose Microsoft XML API because of its simplicity of integration in Visual Basic like an ActiveX component, and also because it allows to check that the XML input file complies with a DTD or Schema.

5.5.1 The Tokens Handler

To take care of all tokens of the CCL, we have the cmdTokens.xml file which lists all tokens of the language. We built a class called TokensHandler, which take care of this file. This class uses also both ImportXml and ExportXml classes to write or read some complex command in a xml file.

So the user can add some new complex command as new word of the language (the new complex command is transformed in an xml file and its name is added to the cmdTokens.xml file).

5.5.1.1 Saving into XML

Saving complex commands in XML is made the ExportXml class, which is able to read a StrTree and to transform it into an XML file according to the DTD.

5.5.1.2 Reading from XML

Reading XML to extract some complex commands as StrTree is made by the ImportXml class. We need two kinds of XML reading:

- To load the tokens of the CCL language, to be used in the Icons Interface and to check the tokens after parsing CCL text. The cmdTokens.xml file contains all this information.
- To expand a complex command (user-defined command) into a tree of basic commands. Here there is an XML file for each complex command.

5.5.2 The DTD for the CCL

In order to describe the XML files which contains complex commands, we wrote a DTD (Document Type Definition) quite similar to the CCL.

Here is the DTD for our xml files :

```
<!ELEMENT complexCmd (seq|if|while|doWhile|repeat|doAll|when)>
<!ELEMENT seq ((definedCmd|if|while|doWhile|repeat|doAll|when)+)>
<!ELEMENT definedCmd (id,param*)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT param (#PCDATA)>
<!ELEMENT if (test,seq,seq?)>
<!ELEMENT test (id)>
<!ELEMENT while (test,seq)>
<!ELEMENT doWhile (seq,test)>
<!ELEMENT repeat (numTimes,seq)>
<!ELEMENT numTimes (#PCDATA)>
<!ELEMENT doBoth (seq)>
<!ELEMENT when (event,seq)>
<!ELEMENT event (id)>
```

5.6 StrTree

5.6.1 Description

The StrTree class has no methods and only two properties: the string contained in this node and a collection of children. It is the simplest tree structure (containing string on each node) we can implement. So it is easy to build and to browse a StrTree.

5.6.2 Use

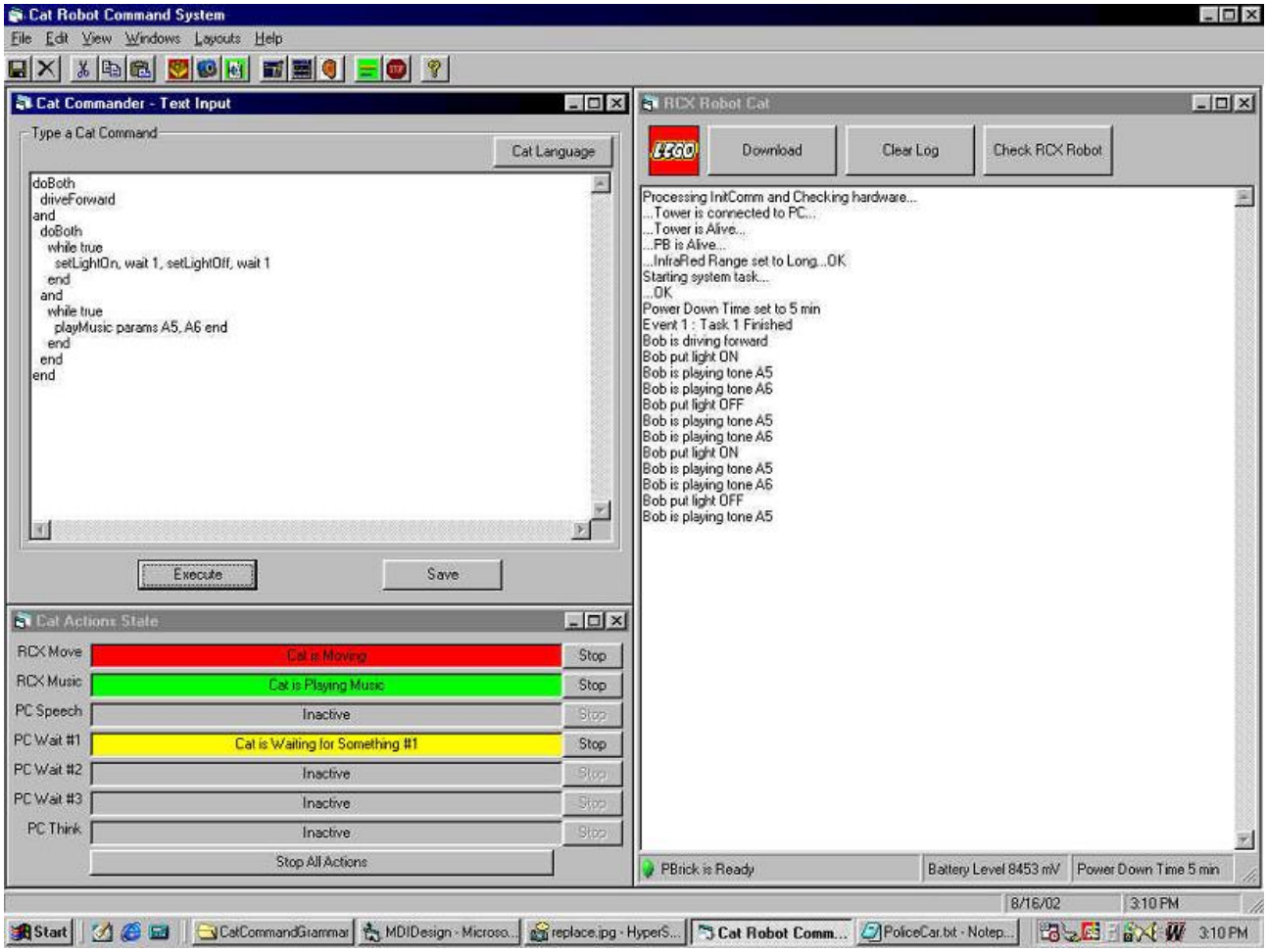
We implemented several algorithms for tree structure conversion relative to StrTree:

- Reduction to StrTree
- DOM to StrTree
- StrTree to ICatBeh
- StrTree to DOM
- StrTree to text tree
- StrTree to CCL sentence

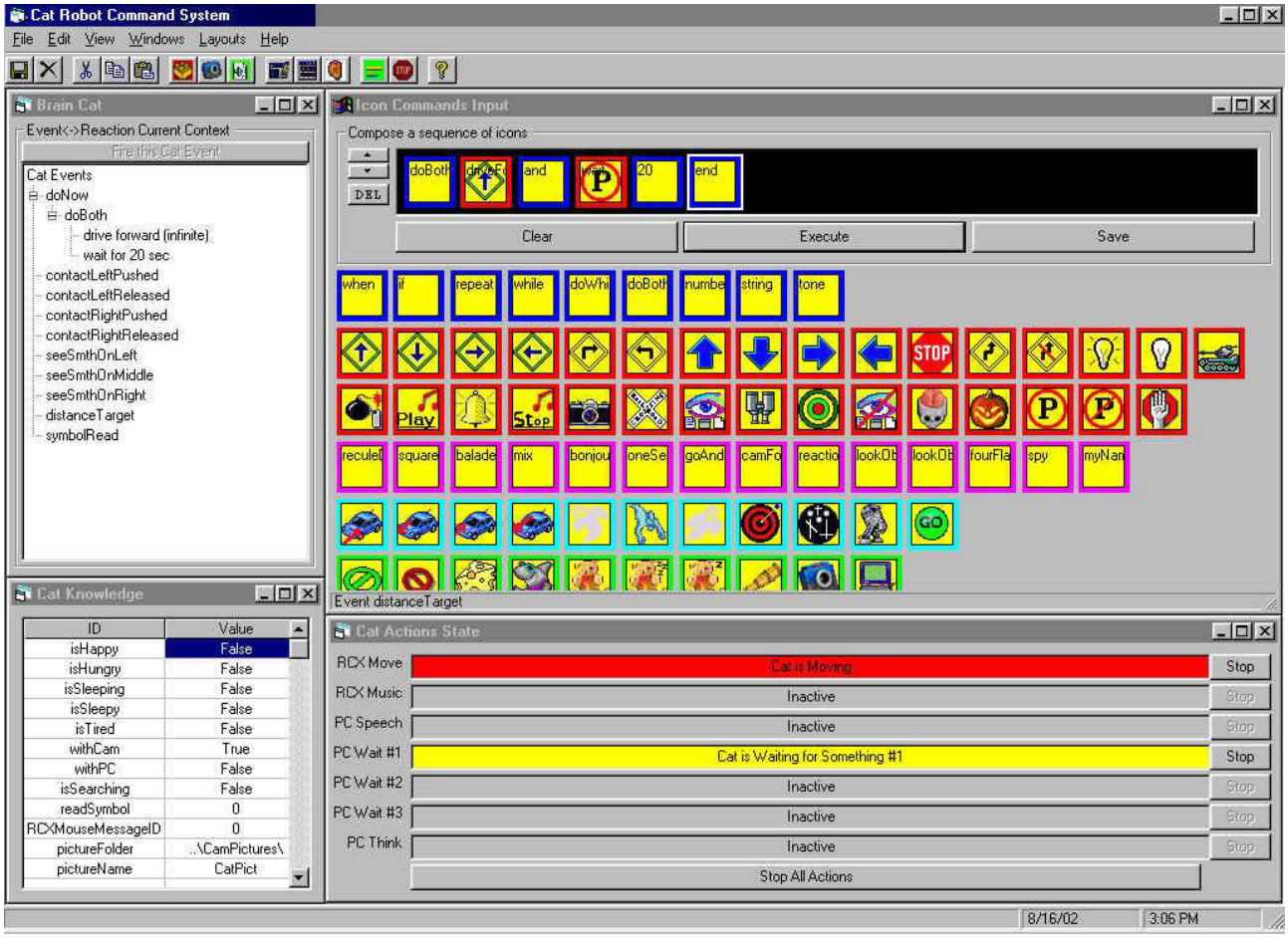
5.7 The User Interface

5.7.1 The MDI Parent Window

We chose to design a user graphic interface using a MDI window (Multiple Document Interface). This kind of interface allows the main window to contain other windows (called child windows). For the most part of the components of MAELIA, a child window is associated. We define several layouts corresponding to the different interests that the user can have.



This layout displays the text input, the RobotCat window and the Timers window.

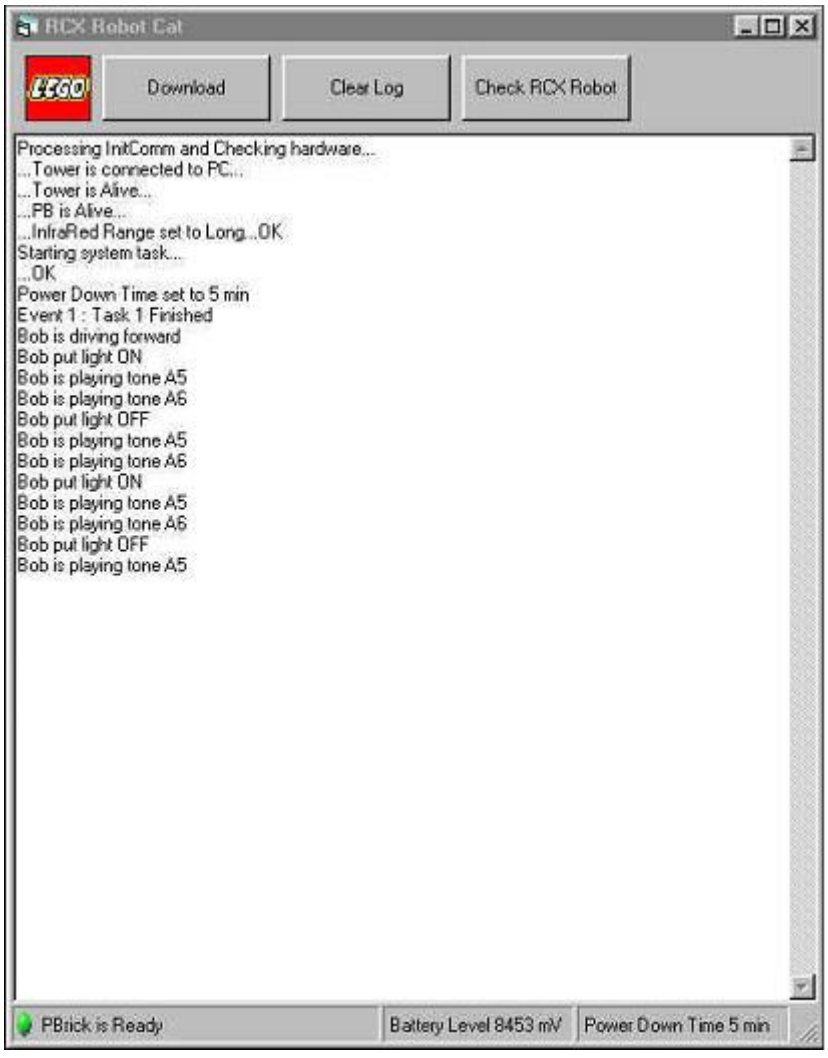


This another layout display the Icon input, the Events Context, the Knowledge and the Timers window.

5.7.2 The MDI Children Forms

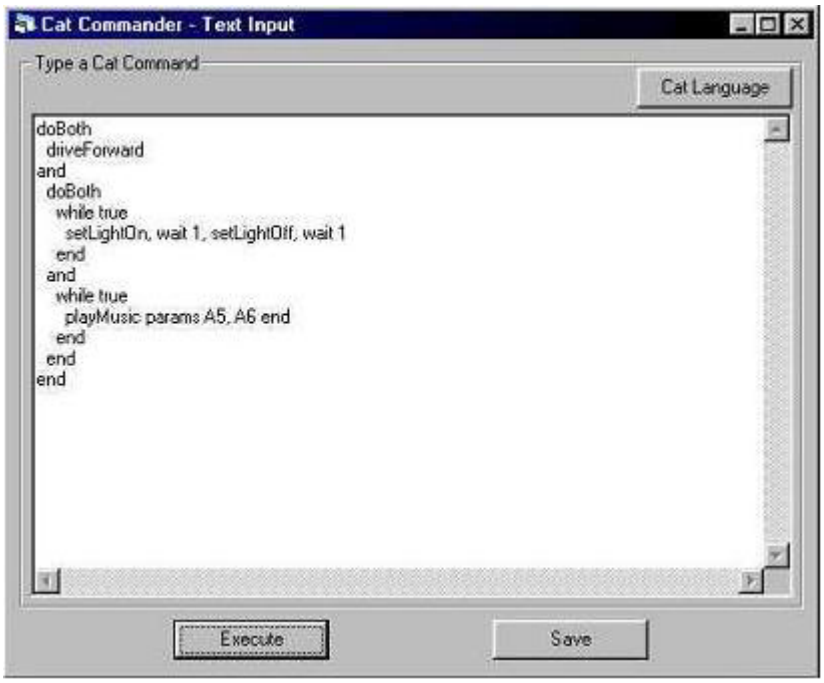
5.7.2.1 FrmRobotCat

This window displays all information about the PBrick: communication status, actions and events on the PBrick, etc. There one command to download our RCX code in the PBrick. There is another one to Clear the log text area, and the last one check if the robot is accessible, and if not, we try to give a diagnostic to know where the problem comes from (COM port, IR transmitter or PBrick).



5.7.2.2 FrmTextInput

This window allows the user to input CCL-compliant text to be executed or saved. There are two command buttons to execute and to save, the last command button displays the CCL reference in the Help Browser window.

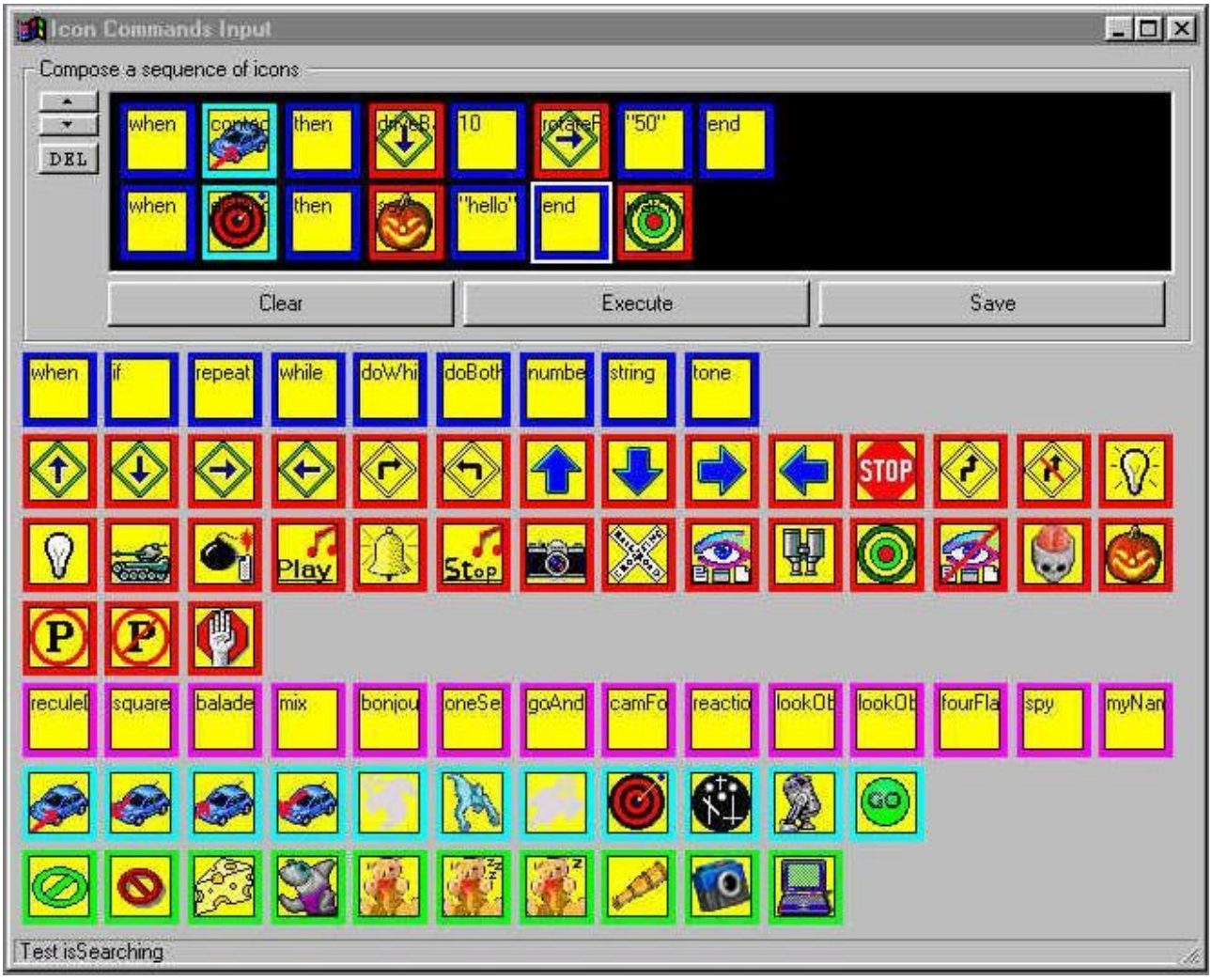


5.7.2.3 FrmTimers

This window displays the current actions running. There are 5 categories of actions : moving, playing music, speaking, waiting (3 available timers) and thinking.

5.7.2.4 FrmIconsInput

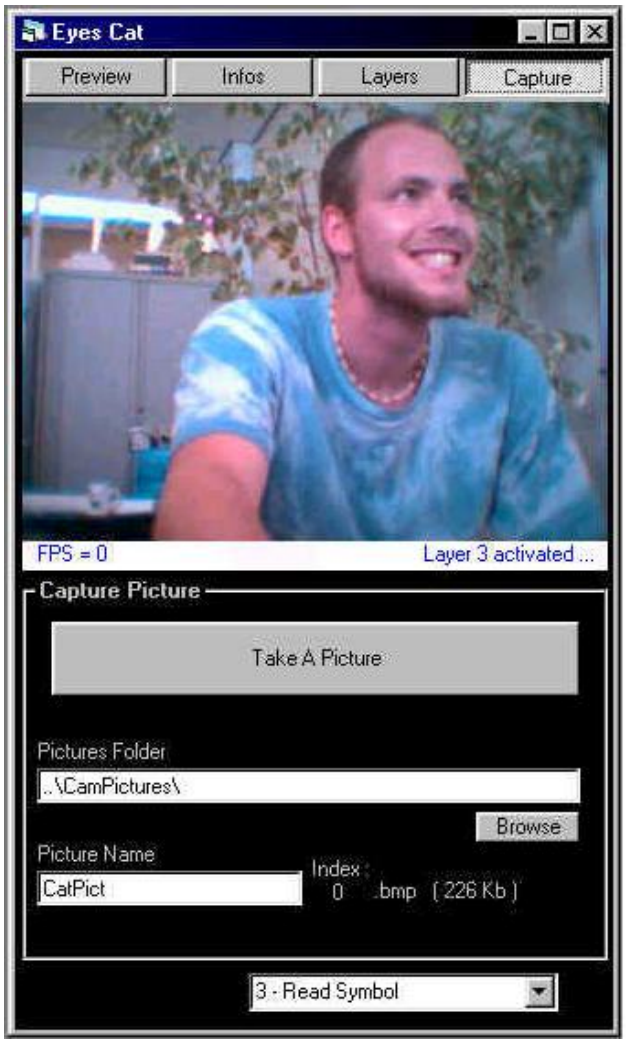
The window allows the user to input an icon sequence by clicking on an icons toolbox. Like the text input window, there are two commands buttons to execute and to save. Here we don't need to display the CCL reference because the icons toolbox already displays all the tokens of the language.



5.7.2.5 FrmEyesCat

This window display the camera preview, and some others features depending on the chosen mode:

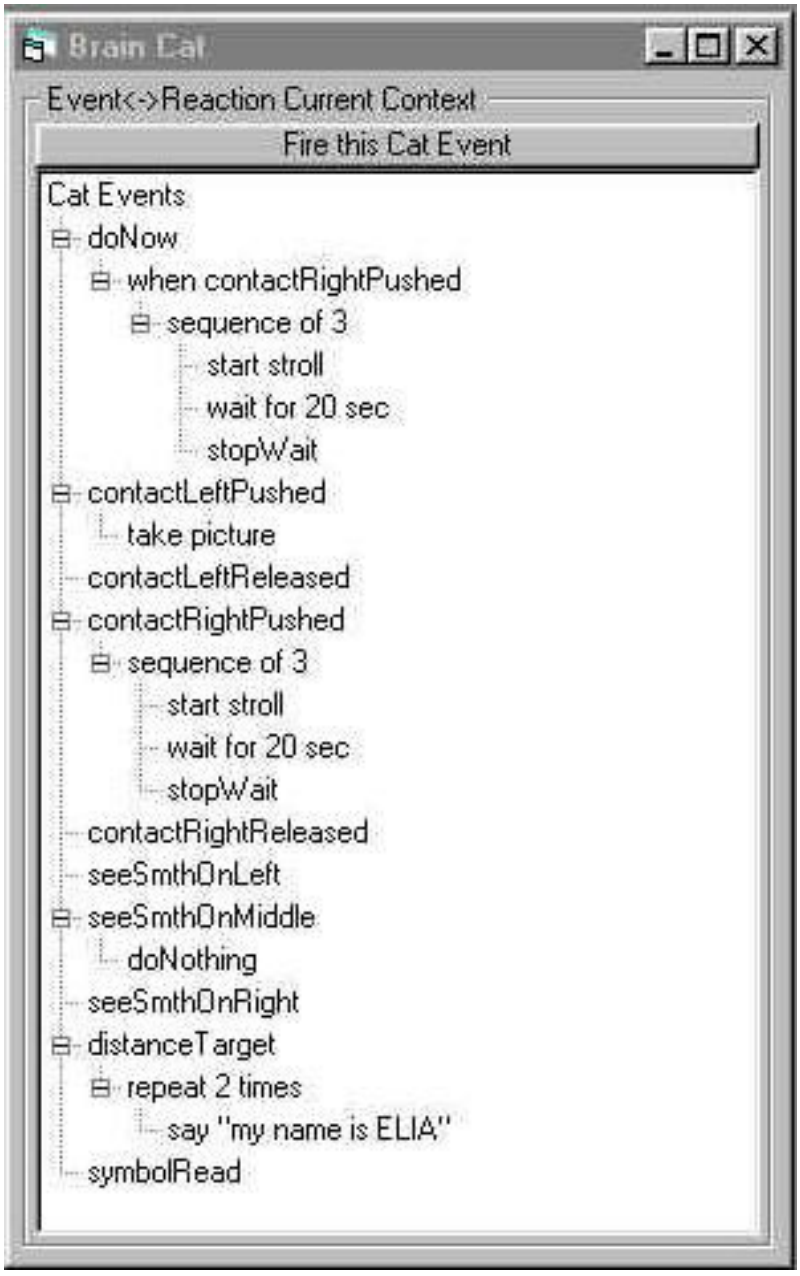
- Preview: displays just the camera screen.
- Infos: displays the screen and a log text area about camera actions and events.
- Layers: display the original screen and another screen with the active layer.
- Capture: displays the screen and some commands to directly take pictures.



Here the capture mode is enabled.

5.7.2.6 FrmBrainCat

This window displays the current Events Context of the Brain. It is possible to artificially fire some events by selecting them and click the command button “Fire this Cat Event”.



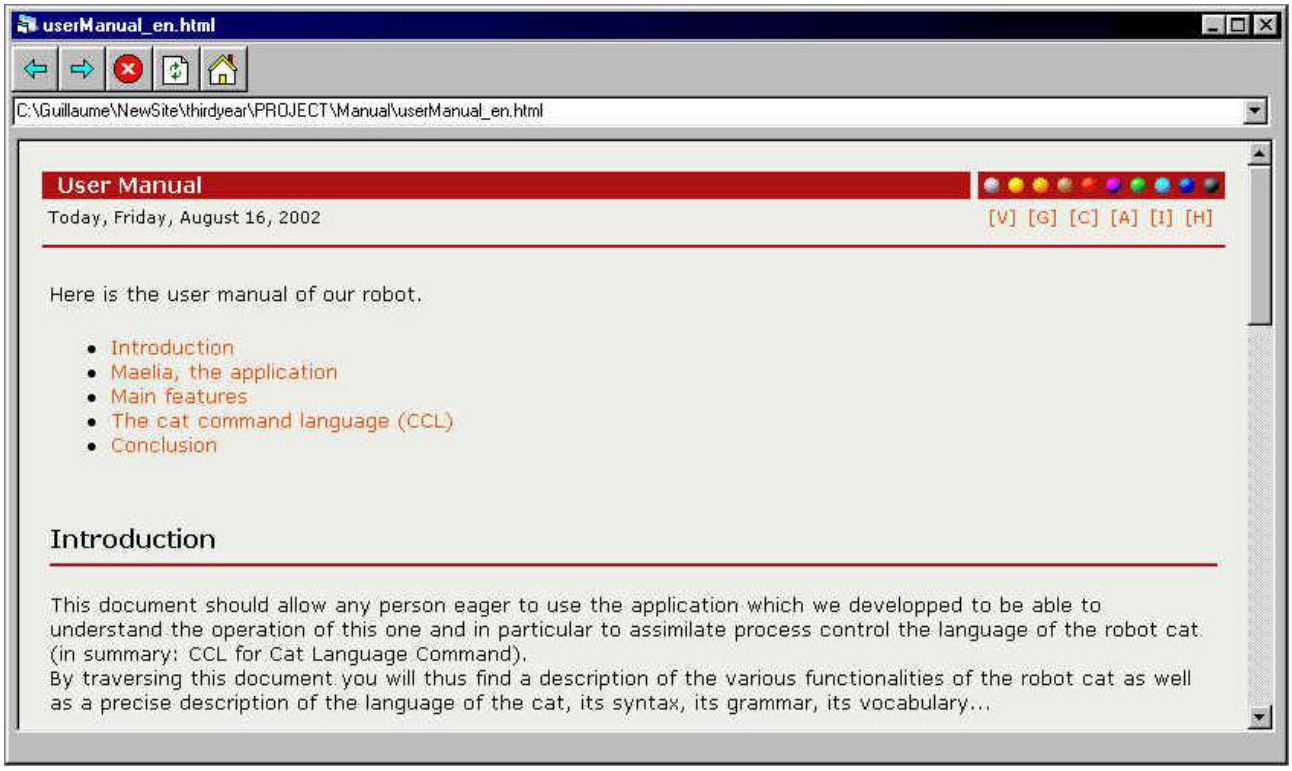
5.7.2.7 FrmVoiceCat

To help user better to understand what the cat is saying, we built a small window displaying the uttered text at the same it's uttered.

User can also use this window to set the volume and the rate of the voice.

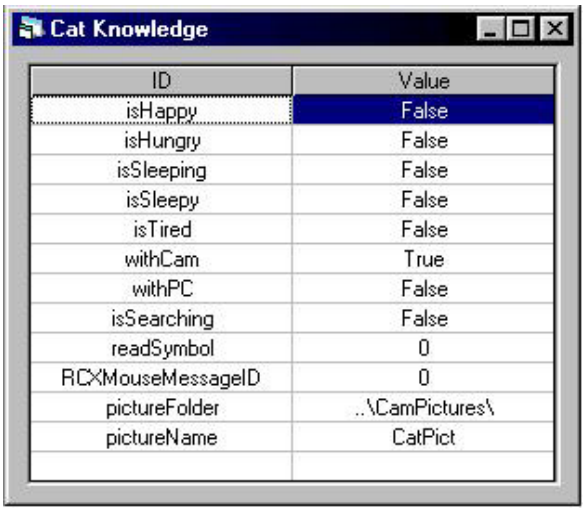
5.7.2.8 FrmHelpBrowser

To provide a constant help for user, we built a browser window able to display all references about the language and also all information about this work through the web site we did. This help window is exactly the web browser of Internet Explorer, but the graphic interface is reduced to the minimum. It's done by using the Web Browser ActiveX control from Microsoft. Because it is actually Internet Explorer, the XSL style sheets works in our application.



5.7.2.9 FrmKnowledge

This window displays the contents of the knowledge base of the system, and it's all. No commands are available; it is just for display.

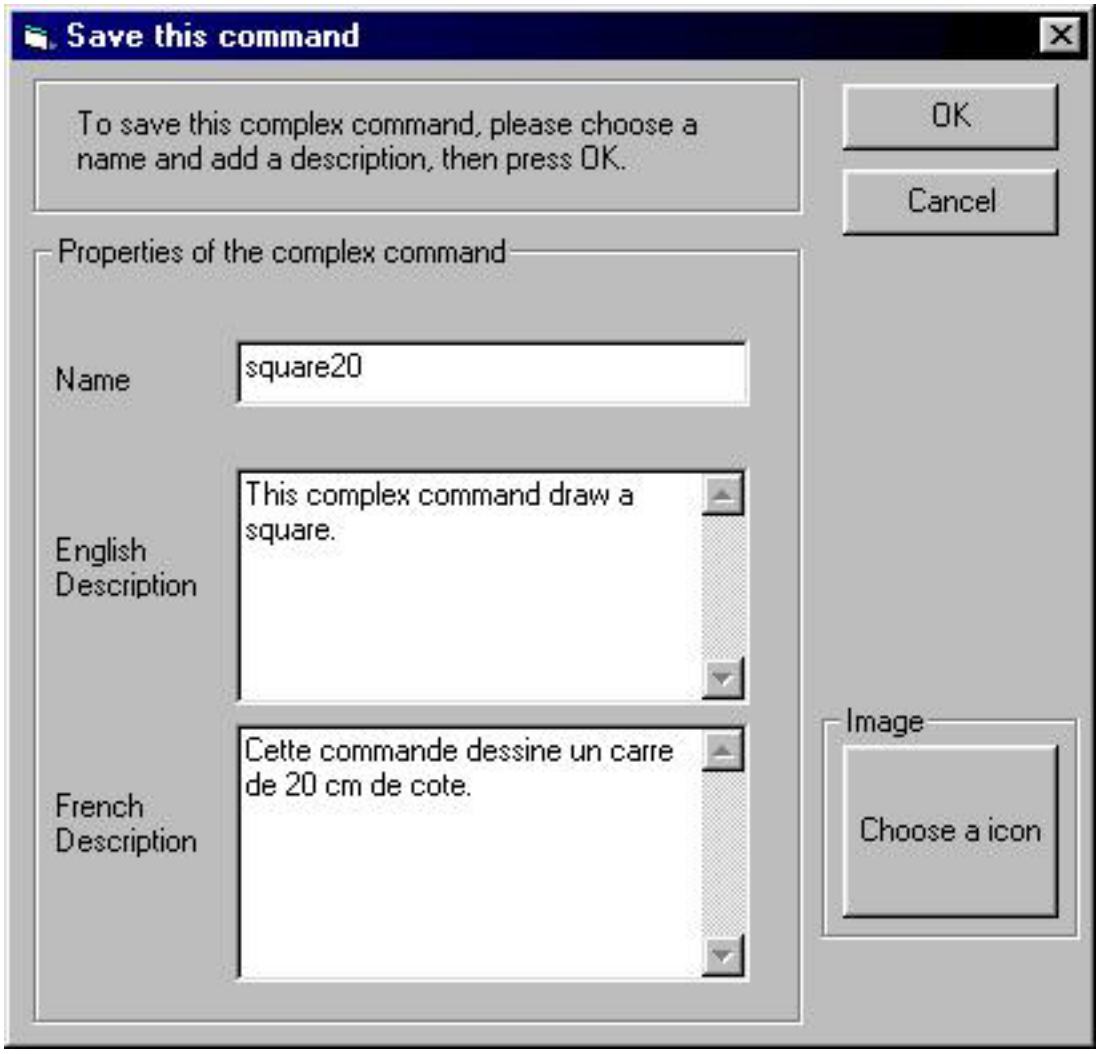


5.7.3 The Dialog Boxes

5.7.3.1 FrmSaveAsXml

This dialog window allows the user to save a composition of commands that the user entered in the text input or the icons input.

The user can use this window when he clicks on one of the buttons "Save" of the FrmTextInput window or of the FrmIconInput window. The user is then invited to input a name for this sequence of orders and a description (optional but useful for help).



5.7.3.2 FrmDelete

This window allows user to delete some complex commands from the cmdTokens.xml file. It uses the TokensHandler object. You just have to choose the name of the complex command you want to delete and press ok.

5.7.3.3 FrmSplash

This window is launched during the load of our application, which can take a lot of time in regard of all components it has to load. It's just a presentation of MAELIA.

5.8 VoiceCat

5.8.1 Using the Text To Speech (TTS) API

We simply used the TTS API to generate voice from text.

There are two ways to generate speech from our program:

- Synchronously: the call to the Speak method is done synchronously, so the following instruction in the code is done after the speech is completely uttered.

- Asynchronously: the call to the Speak method is done asynchronously, so the call returns immediately, and the only way to know the end of the speech generation is to listen to the EndStream event.

We chose the Asynchronous way because it respects the existing model for the commands on the robot cat, which is characterized by two properties: the actions are called asynchronously, and the Events Manager receives an EndOfTask event (which is EndStream here) to synchronize with others actions.

5.8.2 Following the Speech Generation

We used some speech event to synchronize others actions with voice and also to underline the uttered text when it is displayed in the FrmVoiceCat. For the last point we used the Word event, which occurs every time a word is going to be uttered. This event has two parameters that indicate the positions in the text of the uttered word (beginning and end).

5.9 Icons Graphic Interface

5.9.1 Visual Basic User Control

Visual Basic offers the possibility to create your own graphic control, to be reuse as other classic VB controls in your application. So we designed our graphic control which represent a icon of the CCL. It is just a square area with a colored border, containing a picture (ICO file format) and displaying (or not) the name of the command represented. The color of the border represents the type of the icon (control, basic command, complex command, event or test).

5.9.2 Dynamic loading

Because the CCL is extensible for commands, events and tests, at design time we only know what are the control icons. All the others type of icons are loaded dynamically, from the information contained in the file called cmdTokens.xml, which defines the valid elements of the CCL. We use the TokensHandler class to retrieve this information (see section 5.5.4 for details).

5.9.3 Dynamic layout

Because we don't know at design time how many icons we need to display in the icon toolbox, the graphic organization of the icons is done at run-time, and re-done every time the icons input window is resized. It's organized according to the type of the icons, and a new line of icons is done for every types.

5.9.4 Translating into CCL-compliant text

Each icon has a name, which is an element (token) of the CCL. When the user input an icons sequence, it is quite easy to build the corresponding sequence of names, but this sentence is not CCL-compliant, because in CCL the comma is used as the separator between commands. So we need to add comma between icon names, but not everywhere, it was not an easy task to find the correct algorithm, but finally we did and it works pretty well.

5.10 EyesCat

5.10.1 Defining layers for intelligent perception

Now we can define what layers will be useful, these were our ideas :

1. WatchLeftMiddleRight layer: to know if a moving object is in front of the cat (middle) or on the sides (left and right). Three events, one for each position.

2. WatchTarget layer: to know if we are close or not from a colored object in front on the cat. Only one event 'distanceTarget' fired with a value (1-5), which gives the distance (1 close, 5 far).

3. ReadSymbol layer: to read a black symbol on a white panel. We define very simple symbols that we can recognize with color detection if the camera is well positioned. One event 'Symbol Read' fired with the ID of the symbol just read.

4. Inactive layer: when we use the camera to take pictures, we don't want some events to be fired, so we need this inactive layer that makes no detection.

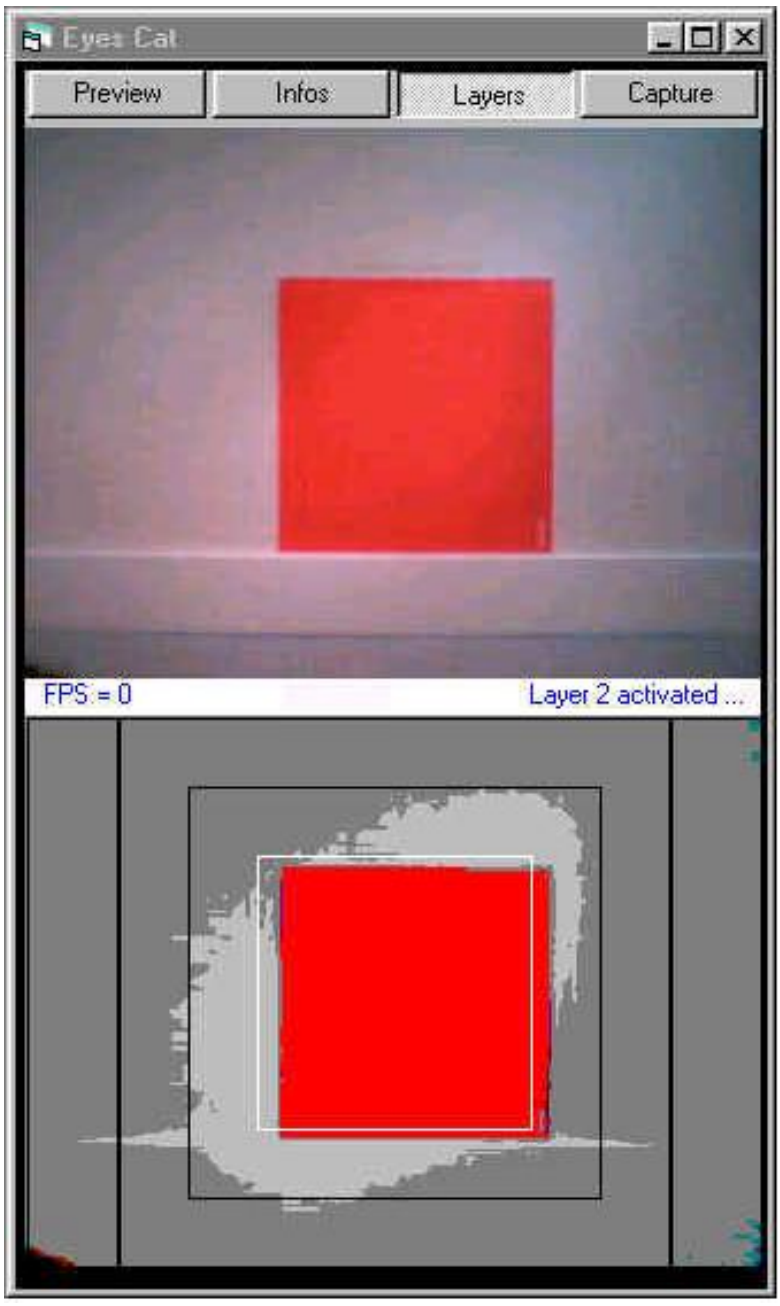
5.10.2 LeftMiddleRight layer

Here is a screenshot of the layer.

The vertical black lines show the different zones in the layer, here there are three zones. An event is fired when 20% of the pixels of one are changed because of movement.

5.10.3 Target layer

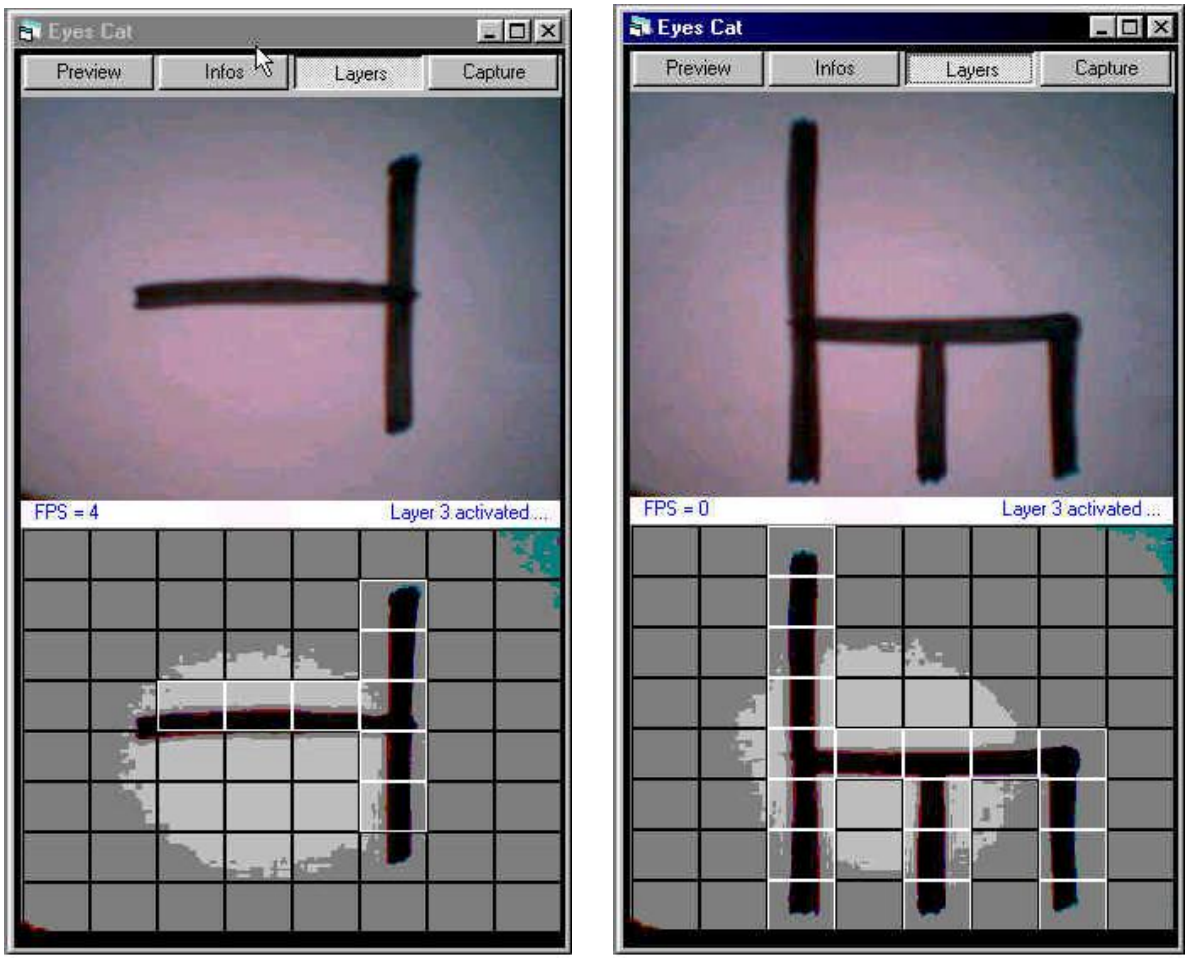
Here is a screenshot of the layer.



We can see three square zones and two strip zones on the sides, the smallest square zone has a white border because the red color detection status is true is this zone, it means that more than 50% of the pixel of this zone are red. When getting closer from this red panel, the status detection in the other square zones will become true, and new distanceTarget events will be fired.

5.10.4 ReadSymbol layer

Here are two screenshots of the layer.



This layer has the maximum number of detections zones (64). It detects black color, and then we process the grid to try to recognize symbols. We can see that the symbols are quite well represented by the white zones on the black zones. When a symbol is recognized, the ReadSymbol is fired with the ID of the symbol read.

5.11 Speech Interface

5.11.1 XML format for Speech Recognition grammar

Underneath follows an explanation of the structure of the XML grammar file used in this system. For a complete documentation on the structure of this XML format, see Microsoft's documentation on Microsoft Speech SDK.

```
<GRAMMAR LANGID="409">                  Start of grammar. 409 is the language ID of English

    <DEFINE>                            Definition list, where all properties are
        <ID NAME="SetLightOn"           associated with a value.
         VALUE="105"/>                  A property can be a rule or a sub phrase. In this case
                                        the rule setLightOn are associated with the
        …more definitions…              ID number 105. The ID numbers are used when the
                                        recognized phrase structure is analyzed by the
    </DEFINE>                           command handler.

    <RULE NAME="setLightOn"             Rule
      TOPLEVEL="ACTIVE"….>
        …rule body…
    </RULE>

    …more rules…

</GRAMMAR>                              End of grammar
```

As an example of a rule follows a *simplified* version of the Move rule, which is used to make the robot go forward or back

```
<RULE NAME="Move"                       Rule header. Attributes: name of the rule (which is
 TOPLEVEL="ACTIVE">                     associated with a value in the definition list), top level rule
                                        means it can be the start of a phrase (not only a sub rule).
    <O>                                 A (sub) phrase inside <O>…</O> tags is an optional part of
                                        the main phrase.
        <L>                             <L>…<L> contains a list of alternative (sub) phrases.
            <P>go</P>                   <P>…<P> contains a (sub) phrase.
            <P>drive</P>
            <P>move</P>
        </L>
    </O>
    <RULEREF="Number">                  A pointer to a sub rule. Number is a rule that represents a
                                        maximum three-digit number.
    <P>centimeters</P>
    <L PROPID="Direction">              A list can have a property ID, that will be associated with a
                                        value. This ID and value is used in the analyzing of the
                                        phrase.
        <P VAL="0">forward</P>          The VAL attribute contains the value the property is
        <P VAL="1">back</P>             associated with.
    </L>
</RULE>
```

Examples of valid phrases:

- "go ten centimeters forward"
- "drive fifty-two centimeters back"
- "two hundred and thirty four centimeters forward"

5.11.2 Some little problems...

However we met a lot of problems with the Speech recognition.

- The first limit was to rewrite the grammar each time we added some new words or new commands because our grammar in this XML file is static. For example each time you add a new complex command, you must in theory add it in this grammar to be recognized by the speech recognition.
- The second was the difficulty to utter good sentences understandable by the computer, because the technology is not still very efficient.
- The last but not the least was to install the Speech API and to manage to use it.

6. Extensibility

MAELIA 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 own their actions and events. The input layer is also extensible while the new input can be translated in CCL.

See the Maintenance Manual in the Annexes (section 11.3) for explanations about how to extend MAELIA.

Some ideas for extensions will be given in this manual.

But a very good extension for our cat could have been the icons recognition by the Lego camera. Indeed we built a icon language and in a second part, we built a module able to recognize symbols by the Lego camera. So we thought to combine these two process to be able to recognize some icons belonging to the CCL. By this way, it could be a fourth way to command the cat.

7. Conclusion

It was a wonderful project for us. An enrichment on all aspects of our work. This training period abroad will remain certainly unforgettable.

Our work in TU Delft was really one of the most interesting we have ever done. We learned a lot about Artificial Intelligence and communication with Intelligent Agent.

We met a lot of people with who we share some knowledge and cultural aspects and not only from Netherlands but also from entire Europe and from all the planet. That is because we also lived in a residence with a lot of Erasmus-Socrates students from all Europe.

We are very glad to have made this project in this university. Thank you Leon.

8. Glossary

AI : Artificial Intelligence.

Artificial intelligence : Simulation by computer of the process of the thought. (a AI problem is a problem requiring the development of an artificial human brain). An example of high technology in AI is Eliza (or My_elisa made by Sisca), and it is about all. It is also the use of principles of heuristics and of data bases containing of the rules, to simulate the working procedure brain of an expert. This definition of the AI gave the "expert system", which can have surprising performances. An artificial intelligence is known as "Strong" when it is able to think for good as a true brain (that still holds of the myth).

Artificial life : Discipline cousin of the artificial intelligence, seeking to create artificial alive beings, in particular organisms able to evolve according to rules which were not written by the initial creator. It is discussed, especially because it works enormously on the viruses, which are the most primitive forms of life as well in nature as in the computers.

ASR : Automatic Speech Recognition.

Case Based Reasoning : Method in which we take old solutions, found to regulate similar problems to that that we study, which we modify to find the solution with the present problem. (CBR).

Chaining : The postpones chaining and the front chaining consist in connecting logical deductions, respectively to go to its effect or cause. In practice, that gives things of the kind: "If... Thus... Then... Because... Thus... If... ". Very much used in artificial intelligence, in the expert system.

Cognitive : Which relates to the thought. Cognitive sciences. The problem of cognitive sciences is that they deal with artificial intelligence: many hopes, few achievements, enormously science fiction.

Detector : There are two types: external sensors and internal sensors. An external detector is a brick LEGO which can be connected on a input port on a RCX in order to supervise any change of environment. The contact and light detector are external examples of detector. The infra-red receptor is an internal detector.

Detector port : On the RCX, one of the three gray plates on which external detectors can be connected (called input port).

EA : Evolutionary Algorithm.

EP : Evolutionary Programming.

Expert system : Application able to carry out in a field of the logical reasoning comparable with those which would do of the human experts of this field. It is based on data bases of facts and knowledge, like on an inference engine, enabling him to carry out logical deductions (chaining before and back). It is before a whole computerized decision-making system.

Fuzzy logic : Logic in which the veracity of a proposal is a real number of the interval [0,1]. The "false" Boolean is at (0), "true" Boolean truth at the other (1). With this system, you can say for example if something is hot or cold. You can also say if this thing is tepid (i.e. hot and cold at the

same time!). This idea was proposed for the first time by Lotfi Zadeh. These intermediate states, between two major statuses, do not exist in traditional logic, so called Boolean logic.

GA : Genetic Algorithm. Type of algorithm, in which vectors of parameters evolve such as the chromosomes to a solution of a very complex problem, by using the principles of the evolution of the natural species.

Heuristic : Technique consisting in learning gradually, by taking account of what we previously did for get closer to the solution of a problem. The heuristics does not guarantee at all that we arrive at an unspecified solution in a finished time. Opposed to algorithmic.
The heuristics is primarily used in the antivirus, to detect viruses by recognizing them according to what they are able to make rather than according to a fixed signature.

Inference : The realization of a logical deduction, used by an expert system to appear intelligent. See also inference engine.

Inference engine : Program carrying out the logical deductions of an expert system starting from a base of knowledge (facts) and of a base of rules.

Infra-red transmitter : External tower connected to the PC which communicates with the RCX.

Intelligent : Said itself of a system, when it is able to make things which we would not have expected. And like we do not expect it, after a first good surprise, the problems start...

LIPS : Logical Inference Per Second. Measuring unit used in artificial intelligence, measuring the number of logical inferences carried out per seconds.

Micro-programs : Here, it is about the process allowing the PC and the RCX to communicate together. It plays the role of operating system for the RCX.

Operating system : Together basic functions (but sometimes being able to be very advanced), allowing the use of a computer, and without which nothing is possible.

Output port : On the RCX, one of the three black plates on which output (such as an engine) can be connected.

RCX (Robotics System Command) : Programmable brick LEGO which makes it possible to store and to execute programs already downloaded from the PC. Each RCX has three output and three detector input.

Remote loading : Process to transmit to the RCX a program in RCX code or a micro-program from the infra-red transmitter.

Shape recognition : Analyze of a whole of data (e.g. an image) in order to find a shape, i.e. predetermined configurations, like phonemes (voice recognition) or characters (character recognition).

Test of Turing : Test conceived by Turing Alan Mathison, in order to determine if a computer thinks: a experimenter-tester on a side, and a machine and a human of the other. If the tester is made have by the machine and cannot make the difference between the man and the machine, then the machine thinks. In an alternative of the test, the machine alone must be made stand for a man.

Voice recognition : Sound analyze to separate the various marked syllables and thus to find the text which is known as.

9. Bibliography

Icons et communication

Designing Icons and Visual Symbols :
http://www.acm.org/sigchi/chi96/proceedings/tutorial/Horton/wh_txt.htm
http://www.acm.org/sigchi/chi95/Electronic/documnts/tutors/wkh_bdy.htm

Sémiotique, phénoménologie et jeux de langage (in french):
http://www.univ-nancy2.fr/ACERHP/perso/bour/sempheno.html
http://cavi.univ-lemans.fr:8900/public/unesco/m3.4.1/M3413.html

Langage et communication (in french):
http://www.linguistes.com/langue/intro.htm

AISolutions :
http://www.generation5.org/solutions.shtml

Pascal Vaillant : Sémiotique des langage d'icônes. Honoré Champion éditeur. 1999

Technical references

Lego Construction :
http://www.ceeo.tufts.edu/curriculum/index.htm
http://prelude.psy.umontreal.ca/~cousined/lego/

Rcx Technical Support :
http://neuron.eng.wayne.edu/LEGO_ROBOTICS/lego_robotics.html

RCX Internals :
http://graphics.stanford.edu/~kekoa/rcx/
http://www.crynwr.com/lego-robotics/

Controlling LEGO Programmable Bricks Technical Reference :
http://www.dcs.ex.ac.uk/academics/reverson/outgoing/pbrick.pdf

RCX Sensor Input Page :
http://www.plazaearth.com/usr/gasperi/lego.htm

Microsoft Speech SDK 5.1 :
http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/msdn-files/027/000/781/msdncompositedoc.xml

Visual Basic :
http://www.laltruiste.com/coursvbscript/constante_msgbox.html
http://www.a1vbcode.com/

The LegoCam camera

Logitech :
http://www.logitech.com/

Programming with the LEGO Cam :
http://www.generation5.org/aisolutions/rob06.shtml
http://www.plazaearth.com/usr/gasperi/viscommand.htm

SDK QuickCam Developer Program :
http://developer.logitech.com/sdk/

VCS ActiceX :
http://www.snoozysplace.be.tf/

Some projects with the camera :
http://www.nullgel.com/projects.html

EdVidCap ActiveX control :
http://www.shrinkwrapvb.com/
http://www.shrinkwrapvb.com/ezvidcap.htm

Some ideas

Design Patterns (Catalogue des modèles de conception réutilisables).
Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides.
Editions Vuibert, Paris, 1999.

Serious Lego :
http://jpbrown.i8.com/

Publications :
http://www.bartneck.de/publications.html

LegOS :
http://www.noga.de/legOS/

RCX Command Center :
http://www.cs.uu.nl/people/markov/lego/rcxcc/

A lot of links :
http://www.cs.unc.edu/~lastra/comp006/lego_mindstorm_links.html

If we have done it with Java

RCX Java API :
http://www.escape.com/~dario/java/rcx/

RCXPort, Java Interface to the Lego Mindstorms RCX :
http://www.slewis.com/rcxport/

LeJOS :
http://lejos.sourceforge.net/

10. Annexes

10.1 User manual

10.2 Maintenance manual

10.3 Statistics

In this annex you will find the listing of all the files of the source code of MAELIA. There are several types of source files, for Visual Basic we have CLS, FRM and BAS as file extensions, respectively for class module, form module and class module. We also counted the XSL files used to generate HTML from our XML files.

For each file we give the number of lines of code, and the total for each subfolder. At the end you can see that the final version of source code is more than 15000 lines of codes.

10.4 Source code

Because the source code of MAELIA is more than 15000 lines of codes, we compute that we need more than 300 pages to put it in the report, thus we don't give the source code here. If you want to have a look on it, you can download an archive ZIP file containing all the sources code of MAELIA, at the following URL:

http://users.lug.com/gbarraud/Enseirb/thirdyear/project/docs_en.html