# Early Pragmatic Language Development for an Infant Robot

by

Paulina Varchavskaia

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2002



Author .....
Department of Electrical Engineering and Computer Science
February 6, 2002

Certified by .....
Rodney Brooks
Fujitsu Professor of Computer Science
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

# Early Pragmatic Language Development for an Infant Robot

by

Paulina Varchavskaia

Submitted to the Department of Electrical Engineering and Computer Science
on February 6, 2002, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science

## Abstract

Recent years have seen the growth of the idea in AI that machines can be designed as sociable creatures: that interacting with a robot could and should be as natural as talking to a fellow human and that machine learning could be achieved in a benevolent environment with a human teacher in ways similar to children's learning (Roy 1999, Cassell et al. 2000, Breazeal 2000, Breazeal & Scassellati 2000). The two components of the future natural teaching of robots are a maximally natural interface and fast learning of categories and tasks from few examples. A natural interface lets humans intuitively use all the discourse cues and displays normally present in conversation. Sociable machines are embodied creatures which encourage natural interaction from humans and pick up on those cues and displays for the development of their own abilities, such as learning to perform new tasks, or learning to express themselves.

This thesis describes a system that engages in such interactions and grounds its vocalizations in a manner inspired by the way human infants establish a communicative basis with their teachers and learn the meaning of words by engaging in functionally, pragmatically meaningful communication. It is a module of proto-linguistic behaviors designed and implemented on a robotic head that was already developed in the Artificial Intelligence Laboratory. The module includes a network of expressive proto-verbal behaviors which enables the robot to establish a common basis for vocal communication with the human teacher and express its state to the teacher. It also proposes a mechanism for the acquisition of new concepts, which is based on a representation of linguistic knowledge as processes in a concurrent architecture. The current thesis presents the idea, the relevant linguistic and robotic background, and the framework, design and implementation of the new system.

Thesis Supervisor: Rodney Brooks
Title: Fujitsu Professor of Computer Science

# Acknowledgments

I would like to acknowledge the invaluable contributions of members of the Humanoid Robotics group at the Artificial Intelligence Lab: Professor Rodney Brooks, Dr. Cynthia Breazeal (presently at the Media Lab), Paul Fitzpatrick, Dr. Brian Scassellati (presently at Yale University), and Lijin Aryananda. All the work reported here is built for the hardware platform, using the software tools, and extending the systems that originally came into existence through their efforts. Without them, all of this would have been nothing more than a crazy idea inside my head. Maybe the idea would never have happened either without the intellectual environment fostered and nourished in the lab by Professor Brooks and without his role as my advisor in the last two and a half years, and especially in the last six months.

Naturally, I also wish to thank all the other members of the group who provided the atmosphere and the intellectual stimulation, and from whom I learned so much. Special thanks go to my office-mates Jessica Banks and Eduardo Torres-Jara, for putting up with me, watering my plant when I forgot to, feeding me cookies, and in general being so nice to me. More thanks to Dr. Una-May O'Reilley who kept a constant inquisitive interest in this project and sent important references.

I have benefitted immensely from the acquaintance of two people who have shared their first-hand knowledge and experience of language acquisition in embodied agents: Professor Justine Cassell and Professor Deb Roy of the Media Laboratory at MIT, and from attending their courses.

Finally, I would never even be here without the continuous love and support of my wonderful parents Tatiana Varshavskaya and Vladimir Varshavski, and my sister Olga Varshavskaya, who have all my love.

Funds for this project were provided by DARPA as part of the "Natural Tasking of Robots Based on Human Interaction Cues" project under contract number DABT 63-00-C-10102.

# Contents


**1 Introduction** **15**

1.1 Motivation for Natural Language . . . . . 15

1.2 Pragmatic Basis of Language Acquisition by Sociable Machines . . . 16

1.3 Overview of the Project . . . . . 18

**2 Background** **21**

2.1 Human-Robot Interaction . . . . . 21

2.1.1 Sociable machines . . . . . 21

2.1.2 Robots and speech interfaces . . . . . 22

2.1.3 Robotic language acquisition . . . . . 23

2.2 Development of Meaning and Language . . . . . 26

2.2.1 Pragmatic protolanguage . . . . . 26

2.2.2 Infants' perceptual abilities . . . . . 28

2.2.3 Nature of concept and word acquisition . . . . . 30

**3 Kismet: the Robotic Platform** **33**

3.1 The Physical Robot . . . . . 33

3.2 Perception . . . . . 34

3.3 Motivation and Behaviors . . . . . 35

3.4 Motion . . . . . 37

3.5 Social Play and Learning . . . . . 37

3.6 Special Skills . . . . . 38

**4 Robotic Behaviors in C++** 41

4.1 Behavior-Based Programming . . . . . 41

4.2 Lateral and Zac . . . . . 42

4.2.1 The Lateral priority scheme . . . . . 43

4.2.2 Zac: An implementation in C++ . . . . . 45

4.3 Shortcomings of Zac for Adaptive Behaviors . . . . . 49

4.4 BALZac: Better Adaptation to Learning . . . . . 54

4.4.1 Architectural requirements for adaptive behaviors . . . . . 55

4.4.2 The BALZac extensions: implementation . . . . . 58

4.4.3 The BALZac syntax . . . . . 60

**5 Proto-Linguistic Vocal Behaviors** 67

5.1 Protoverbal Behaviors . . . . . 67

5.1.1 Why speak of behaviors? . . . . . 68

5.1.2 New speech-related drives . . . . . 70

5.1.3 Generic behavior design . . . . . 70

5.2 Kinds of Vocalizations . . . . . 72

5.2.1 Emotive grunts . . . . . 72

5.2.2 Canonical babble . . . . . 76

5.2.3 Formulaic protolanguage . . . . . 81

5.3 Overall Architecture and Communications . . . . . 83

**6 Mechanisms of Early Concept and Label Acquisition** 87

6.1 Requirements for Acquisition Methods . . . . . 87

6.2 Concept Methods . . . . . 89

6.2.1 Attribution of label . . . . . 90

6.2.2 Label update and confidence . . . . . 92

6.2.3 Parameter update . . . . . 94

6.3 Preliminary Results from "Fixed" Concepts . . . . . 96

6.4 ConceptMap Methods . . . . . 99

6.4.1 Architecture and operation of the map . . . . . 99

6.4.2 Addition of new concepts . . . . . 101
6.5 Overview . . . . . 102

**7 Conclusions, Limitations, and Future Work** **103**
7.1 Discussion of Results . . . . . 103
7.2 Limitations . . . . . 104
7.3 Future Work . . . . . 104

**A The Phonemes of American English** **107**

**B Syntax of BALZac Script** **109**

# List of Figures


3-1 Hardware configuration of Kismet . . . . . 34

3-2 Kismet: the robotic head . . . . . 34

3-3 The behavioral framework of Kismet . . . . . 36

4-1 A connection object . . . . . 44

4-2 Priority arbitration in a behavior object . . . . . 44

4-3 MeshLink: implementation of a connection object . . . . . 47

4-4 BALZac in relation to Zac and other robot architectures . . . . . 55

4-5 A template sorting behavior . . . . . 57

5-1 The simple state machine of a drive. . . . . 70

5-2 A single protoverbal behavior . . . . . 72

5-3 A grunting behavior in BALZac . . . . . 74

5-4 Activation of grunting behaviors over time. . . . . 75

5-5 State machine for Canonical babble. . . . . 76

5-6 Canonical babble generation on Kismet . . . . . 77

5-7 Activation of Canonical babbling behavior over time. . . . . 80

5-8 Overall architecture of Kismet's protoverbal behaviors. . . . . 84

5-9 Communications in the heterogeneous network. . . . . 85

5-10 Control and data flow between verbal and other behaviors. . . . . 86

6-1 State machine of a `Concept` behavior. . . . . 89

6-2 Result of a call to *BestMatch*(). . . . . 91

6-3 Concepts signalling the results of phonemic matching, and the confidence measure. . . . . 93
6-4 The `HeardThis` behavior and its state machine. . . . . 94
6-5 Timings in state transitions of `HeardThis` and a `Concept`. . . . . 95
6-6 Some data from a label teaching scenario. . . . . 98
6-7 The `ConceptMapManager` behavior and its state machine. . . . . 100

# List of Tables

2.1 Development of human infants' perceptual, motor, vocal, and linguistic abilities during the first year of life. . . . . 31

4.1 Functionality of the BALZac extensions . . . . . 59

5.1 Correspondence between nonverbal behaviors and proto-linguistic functions . . . . . 81

6.1 Results of a label teaching scenario. . . . . 97

A.1 The Phonemes of American English: Vowels . . . . . 107

A.2 The Phonemes of American English: Consonants . . . . . 107

B.1 BALZac Script syntax in EBNF notation. . . . . 110

# Chapter 1

# Introduction

Recent years have seen the growth of the idea in AI that machines can be designed as sociable creatures: that interacting with a robot could and should be as natural as talking to a fellow human and that machine learning could be achieved in a benevolent environment with a human teacher in ways similar to children's learning (Roy 1999, Cassell et al. 2000, Breazeal 2000, Breazeal & Scassellati 2000). The two components of the future natural teaching of robots are a maximally natural interface and fast learning of categories and tasks from few examples. A natural interface lets humans intuitively use all the discourse cues and displays normally present in conversation. Sociable machines are embodied creatures which encourage natural interaction from humans and pick up on those cues and displays for the development of their own abilities, such as learning to perform new tasks, or learning to express themselves.

This thesis describes a system that engages in such interactions and grounds its vocalizations in a manner inspired by the way human infants establish a communicative basis with their teachers and learn the meaning of words by engaging in functionally, pragmatically meaningful communication.

## 1.1 Motivation for Natural Language

There are many reasons to endow a robotic system with a speech comprehension and production interface. It is easy for human users to communicate through the

medium of spoken language (we do not examine the cases in which it is easier to communicate using written, sign, or other linguistic forms). And even though a picture is worth a thousand words, in the cases where words must be used, it is more natural for humans to string them together in meaningful chunks of natural speech, than to learn a new "language" of commands. These reasons pertain to human-robot interaction scenarios, in which the robot plays the role of sophisticated tool that a human may use in order to achieve a goal or perform a task.

In the emerging field of sociable robotics, however, the role of the robot is no longer that of a passive device, but rather that of a participant in a cooperative task with humans. This shift of focus influences the choice of interface in important ways. It becomes essential to design for a smooth and natural cooperative interaction, in which there is an established system of cues, behavioral displays and underlying meanings, which both the human and the robot may use to their advantage. Natural spoken language fulfills this requirement as it is composed of the spoken utterance together with accompanying quality of voice, intonation, posture, gesture, and other natural displays, all of which contribute to the information content that is being communicated. If a robot can pick up on these cues, it will receive a richer message. If it can display such cues appropriately, it will be more successful in communicating a message to the human. Ultimately, the most important part of the message is its meaning that should be recoverable by both sides. If our task is to make the interaction flow as natural as possible for the human, making the robot rather than the human learn the shared system for communication makes even the more sense.

## 1.2 Pragmatic Basis of Language Acquisition by Sociable Machines

The position taken in this thesis is that the shared semantics between the robot and the teacher are procedural or functional, and not declarative or combinatorial. The traditional view of natural language is that of a system of symbols and syntactic com-

bination rules, which encodes a message in a form that can be transmitted through speech or written text. The problem of acquiring such a system is that of acquiring a vocabulary and the applicable combination rules - a notoriously hard problem for machine learning, even in the cases where algorithms are provided with negative evidence. An even harder problem is to link the vocabulary to its referents in the real world - this is known as the Grounding Problem (Harnad 1990) and has been addressed with embodied systems that have perceptual abilities. However, the meaning of natural language symbols does not only come from perceptual grounding.

The pragmatic approach to language acquisition is to consider first of all the intentions of a speech act. Language is not viewed as a denotational symbolic system for reference to objects and relationships between them, as much as a tool for communicating intentions. The grounding of symbols at the pragmatic level is exemplified by the most common everyday conversations. "It's getting chilly in the office" - such a remark is not intended to transmit information from my officemate to me, so I may know that it is indeed getting colder. It may be intended to make me offer to turn up the heating, or may be a way of starting a conversation. The utterance is a way to manipulate the environment through the beliefs and actions of others.

For a robotic sociable creature, speech and language may serve the same purpose. As the robot engages in a cooperative task with a human, utterances should become its tool for influencing the human's behavior, the level of interaction, or the focus of cooperation. The robot's language system must then not only be able to refer to objects or relationships, but also to express intentions in a way naturally interpretable to humans. The acquisition of such a system must begin with goal-directed, motivated prelinguistic vocalizations. We believe that languge acquisition must start with the learning of a protolanguage: a finite set of formulaic utterances, which are not subject to combinational rules but which express a finite set of intentional, pragmatic meanings. Some of these may refer to vectors of perceptual features in the world. Their primary application, however, is not referential; they are intended to achieve a change of state of the world through actions of others.

## 1.3 Overview of the Project

The project reported in this thesis embodies an attempt to create a natural speech interface with a humanoid robot which incorporates and builds on the ideas described above. We develop a system of vocal behaviors for the robot Kismet (Breazeal 2000) which exemplifies the approach we believe should be taken to natural language acquisition by machines. The robot's youthful appearance dictates the sort of interaction that humans will have with it: scenarios of social scaffolding similar to the kinds of interactions that teachers have with human infants. They create a benevolent environment and modify their behavior so as to make the task of learning easier for the infant, so as to taylor the world to her abilities. These actions come naturally to adult humans and the tendency is reinforced when the robot responds appropriately, i.e., as a young infant would.

One step forward is to enable the robot to use the scaffolded environment to its advantage in order to learn to perform tasks and behave appropriately. Learning to communicate with the teachers using a shared semantic basis is one aspect of learning to behave in the world and manipulate it. We augment the existant motivational and behavioral systems of the robot with a set of vocal behaviors, regulatory drives, and learning algorithms, which together constitute Kismet's Protolanguage Module. The purpose of the module is to develop two-way communication with the teacher. This is achieved in two ways. On one hand, we design a set of more convincingly expressive vocalizations, such as grunts and gasps, which are tied into the robot's emotional system. On the other hand, we develop mechanisms by which Kismet may extract meanings from its internal state, its own actions, and the changing environment, and attach vocal labels to them. Meanings, or "concepts", are individual components of the robot's behavior system, which run together in parallel and compete to establish the content and shape of Kismet's utterances.

It is important that the robot's utterances have an intentional basis. They are triggered by the robot's own motivational system, and they serve the purpose of expressing the state of the robot, of regulating the environment through the actions of

the teacher, and of achieving a goal of the robot's behavior system. It follows, and is also important, that the robot's speech acts (or pre-speech vocalizations) are not purely reactive, but actively used by the robot as a tool in world exploration and manipulation, similar to the use of its active vision system. In this Kismet's protolanguage module has a different focus from previous attempts at robotic language acquisition. It also inevitably achieves more modest results. However, we have tried to establish a precedent and a framework for early language development "for the right reasons", and we believe that future work will prove the approach more fruitful.

In this thesis, we first take a brief survey of previous work in robotic language acquisition and present a summary of those ideas, studies, and experiments in the large field of developmental linguistics that most inspired the design of Kismet's protolanguage module (Chapter 2). A description of the robotic platform follows in Chapter 3, as the module must fit into those systems currently operating on Kismet. Chapter 4 outlines an architecture and scripting language for implementing behaviors in C++, which was used and augmented for this project. We then turn to Kismet's system of protoverbal behaviors, its architecture and implementation in Chapter 5. The algorithms for concept and vocal label acquisition, are detailed in Chapter 6. Finally, we present our conclusions and possible directions of future research in Chapter 7.

# Chapter 2

# Background

The problem of early language acquisition has been addressed by many researchers in the fields of developmental psychology, phonetics and linguistics, and more recently, behavior-based robotics and ethology. The present work draws inspiration from many of those studies. It also utilizes many of the results and methods developed in previous research.

## 2.1 Human-Robot Interaction

The idea that a machine should use natural language communication with the user as the ultimate interface is by no means novel. It has been the panacea of the field of Human-Computer Interaction (HCI) (see Cole, Mariani, Uszkoreit, Zaenen & Zue (1995) for one overview). It has seen many research and commercial implementations from dictation systems such as IBM's ViaVoice to telephone enquiry systems such as Jupiter (Zue, Glass, Plifroni, Pao & Hazen 2000). In this section we examine some previous work in speech interfaces on robots and virtual embodied agents.

### 2.1.1 Sociable machines

The project described in this thesis fits into the framework that has been devised in the Artificial Intelligence Laboratory at MIT in the past decade, and in which

in particular the work on sociable machines (Breazeal & Scassellati 2000) has been undertaken. A sociable machine such as Kismet (Breazeal 2000) is a creature endowed with perceptual, behavioral and motor skills that enable it to sustain an interaction with humans in a natural way, and a social drive that ensures some likelihood of such interaction. Verbal communication is one aspect of social interaction, and natural speech one medium of sustaining it. Conversation as humans know it - with each party taking turns to speak, is just one protocol for social communication; however, it is the one protocol that humans feel most comfortable with, since we get used to using it from very early on.

The sociable machine Kismet is the platform on which the research described in this thesis was implemented. As such it will be examined more closely in 3.

### 2.1.2 Robots and speech interfaces

The idea of sociable machines was originally conceived (Breazeal 2000) as an alternative to the traditional perception of robots as servants, where the human-robot interaction is reduced to a command-and-control interface. Traditionally, whenever the interface used the medium of natural speech, it was to enable the human user to naturally communicate a command or request to the robot. For example, the humanoid robot from Waseda University (Matsusaka & Kobayashi 1999) is able to process some simple speech commands and queries regarding its design and purpose, and has a vocabulary of around 1,000 words. Some of the most sophisticated speech interfaces currently developed are embodied in simulated humanoid agents whose purpose is still to assist the human in a specific task. For example MACK (Gesture & Narrative Language Group at the Media Lab 2001) helps users find their way around the Media Laboratory at MIT. Such a speech processing system will understand only a specifically restricted type of human verbal input and will require cross-channel assistance, such as pointing at a hypermap to delimit the location.

To be sure, there has been research into speech interfaces that make small talk and are able to process a whole range of out-of-domain vocabulary that they may encounter as input, resulting in sophisticated agents such as REA (Cassell, Bickmore,

Billingshurst, Campbell, Chang, Villjálmsson & Yan 1999), a virtual real estate agent also developed at the Media Laboratory at MIT. Perceived speech is encoded in a formulaic way at the level of discourse structure, and each separate utterance or words within it may have meaning for the agent which embodies the speech perception only insofar as they enable information retrieval from a database of real estate. We can see that the ultimate purpose in the development of such embodied conversational agents is not the acquisition of language but some kind of functionality that assumes sophisticated language use, as well as the creation of an impression of natural interaction in the human user.

The contribution of the current project is to focus on building a shared basis for communication between the learning robotic agent and the human teacher. The representation for an utterance of natural language is naturally different for the robot than it is for humans; however the focus is on the functional range of the acquired concept and the use that the robot makes of it. Speech is a tool which the robot uses to manipulate the world to fit its "goals", and thus the pragmatic function of the medium is shared with humans.

### 2.1.3 Robotic language acquisition

Some researchers, e.g., Roy (1999), Oates, Eyeler-Walker & Cohen (2000*a*) and Oates, Eyler-Walker & Cohen (2000*b*), approached the problem of acquisition of natural categories and labels by robots from the point of view of perceptual grounding. The robot analyzes the visual scene, and the speech stream into segments, the best correlation between which will form a perceptual concept-label pair which is acquired by the robot.

This approach was taken in particular in the development of CELL (Roy 1999), Cross-channel Early Language Learning, where a robotic platform is developed and a model of early human language acquisition is implemented on it. CELL is embodied in an active vision camera placed on a four degree of freedom motorized arm and augmented with expressive features (eyeballs, eyelids, feathers and a beak). The system acquires lexical units from the following scenario: a human teacher places

an object in front of the robot and describes it. The visual system extracts color and shape properties of the object, and CELL learns on-line a lexicon of color and shape terms grounded in the representations of objects. The terms learned need not be pertaining to color or shape exclusively - CELL has the potential to learn any words, the problem being that of deciding which lexical items to associate with which semantic categories. In CELL, associations between linguistic and contextual channels are chosen on the basis of maximum mutual information.

Specifically, CELL learns lexical units which consist of a linguistic unit, i.e. a word, and a corresponding semantic category. Categories in CELL are "family resemblance categories" - a prototype together with a radius of acceptable deviation. All semantic input is derived from raw sensory signals through contextual channels, and in that sense, CELL's language is grounded in perception. The acquired lexicon can then be used for describing objects, including novel objects, and semantic parsing of speech. It is claimed that the robot's facial expressions and direction of gaze serve to establish a social interaction between itself and the teacher. However, the interaction consists mostly of the human teacher following the direction of gaze of the robot to make sure that she is describing the object that the robot is actually looking at. Although the robotic CELL is an embodied system, it is only situated in a limited and circumscribed portion of the real world. More importantly, however, the acquired lexicon can only be used for denotational purposes such as visual scene description.

Other researchers have applied the principle of grounding word semantics in perceptual inputs of robots to acquisition and evolution of artificial languages. In Steels (1999) and Steels & Kaplan (2001), Luc Steels and his Talking Heads research group develop an environment in which agents learn to describe simple visual scenes using a novel artificial language that they collectively construct as they go along, by engaging in communicative games. The typical Talking Heads experimental setup involves two robotic active vision heads, possessing one camera each, and focusing from a slightly different angle on a two-dimensional visual scene. The scene may consist of various geometric shapes in flat bright colors pasted on a whiteboard. The language learners are software agents which are able to teleport themselves to one of the active cameras

in order to receive the visual inputs of the scene.

Initially, all agents in the population have an empty lexicon. When two agents focus on some portion of the visual scene, one of them starts a communicative game by choosing an object to talk about and describing it to the other participating agent. If the hearer can find the speaker's vocalisations in its lexical repertoire, the object is identified and the game has succeeded. Otherwise, the speaker has to identify the object of the game for the hearer's benefit, both lexicons and conceptual sets are updated, but the game fails. Eventually a population of such software agents will converge on some shared "semantic" representation of the agents' world, as well as on a shared lexicon.

The purpose of the Talking Heads experiments is to provide a starting point for computational theories of language evolution. The languages that the populations of agents evolve can only be used for communication between the agents, and not between a human user and an agent, unless the human chooses to participate in the game and learn the agents' language. From the point of view of robotic language acquisition, one of the limitations of the project is that it relies on the agent's ability to construct a complete hierarchical description of the scene, whose nodes are the attributes which will become the agent's "concepts". This is only possible when the scene consists of relatively few easily segmented objects in straightforward spacial relationships to each other. Describing an object amounts to vocalizing a path in the scene hierarchy according to some parsing strategy. Another limitation comes through when the game fails. The agents do not use the world to identify the object of the game when words fail them; instead, they send an object ID# through a wire. When all objects in the scene are given a priori ID numbers, the language used by the agents becomes grounded not so much in sensory perception as in another symbolic representation of the world.

In order to achieve communication between humans and a sociable robotic creature, words must be a tool used by the robot to manipulate its physical and social world and they must be interpreted by humans as having such a pragmatic functional meaning. In Kismet's case, it will start with proto-language and proto-verbal

behaviors. We draw our inspiration mainly from studies of the linguistic and social development of human infants.

## 2.2 Development of Meaning and Language

Human infants are surprisingly adept at learning about the structure of the environment, how to behave in it, and how to express themselves and understand others all at once, in the space of a few years. We take the approach of Bloom (2000) who has shown that children are good at innately facilitated learning of socially transmitted information. This means that one of the most important attributes of the spoken language in the infant's environment is that it fulfills a social function, to which the infant is sensitive and which enables her to acquire a new word or concept after a very limited number of examples, perhaps even one (see also Pinker (1999) for overview of experiments suggesting extremely fast word learning by young children and infants).

We turn to an account of the social function of the developing language, which was the most inspirational in the design of the language acquisition module on Kismet. In section 2.2.2 we also look at the perceptual and proto-linguistic abilities of very young infants, so we may place Kismet in an appropriate age group.

### 2.2.1 Pragmatic protolanguage

In a seminal work Halliday (1975) presents a longitudinal study of a developing child, Nigel, from 4 months to 2 years of age. He takes the approach on asking why the child develops a language at the early stages, and for what he uses his nascent linguistic ability. Halliday makes the very important distinction between what he calls the *mathetic* and the *pragmatic* functions of human natural language. The mathetic function is to provide an encoding of information channelled through speech or text. If the speaker wishes to inform the hearer that a certain event occurred (e.g., that it has started to rain outside), their common language gives them a shared representation of that information that can be transmitted through speech. Many traditional approaches to machine language acquisition equate language with its mathetic func-

tion and neglect its pragmatic aspect, which is to provide the speaker with a means of manipulating the behavior of other humans. When one person makes a speech act (e.g., the same statement about rain outside), in addition to communicating a piece of information, that person may have intentions to change the state of the world and in particular the behavior of those who hear the utterance, in a particular way. "It has started to rain outside" often really means something like "Close the window, please". The speaker expects the hearer to react in a certain way as a result of hearing the utterance, which then becomes the speaker's tool for manipulation of others and of her surroundings.

This pragmatic view of the function of language is extremely important in trying to explain, or devise, an ability for early language acquisition, because infants and young children specifically learn to use speech as a tool. Halliday identifies three main stages of linguistic development: (I) the child's initial closed proto-linguistic system, (II) the transitional stage to that of adult language, and (III) the learning of the adult language. In the first stage, the child has a finite number of meanings to convey and to that effect uses self-generated labels that may or may not resemble adult words for similar occasions. Halliday posits six initial functions of a developing proto-linguistic system that may be expressed in the first stage:

1. **Instrumental** - satisfying the child's material needs
2. **Regulatory** - controlling the behavior of others
3. **Interactional** - participating in a social situation
4. **Personal** - asserting own unique self
5. **Heuristic** - exploring the environment
6. **Imaginative** - pretending and playing

These six functions of the child's phase I proto-language seem to develop in that sequence and represent the child's growing cognitive ability and awareness. They also present a great starting point and timeline for an artificial system that would acquire

a natural language in a way similar to human children. The infant's protolanguage during Phase I is finite and formulaic (see also Wray (2000) for a discussion of formulaic systems in the evolution of language), as will be the first part of Kismet's language development module.

### 2.2.2 Infants' perceptual abilities

When designing a system for language acquisition whose purpose is to follow the development of a human infant, we should be informed by the relevant research in the domain of developmental phonetics and linguistics. Results in this field establish the habitual timeline for emergence of those perceptual abilities in the infant, which are helpful in the process of language acquisition.

For example, Jusczyk in Jusczyk, Friederici, Wessells, Svenkerud & Jusczyk (1993), Jusczyk & Aslin (1995), and Jusczyk (1997) drew attention to the developing speech perception and production systems. His findings establish the extent and progression of the infant's perceptual and motor abilities during her first years. These influence in an important way the kinds of linguistic knowledge that the infant can acquire, and so they are factors to be considered and possibly replicated in our artificial system. In particular, Jusczyk established which cues enable infants at different stages of development to segment the stream of speech into units such as utterances and words. He showed that the infants most often use prosodic (intonational) information in picking out the most salient information in an utterance. Other sources of information on word boundaries that the infants use include phonotactics and allophone statistics. Phonotactics encode the likelihood of a particular phonemic string to occur within a word versus across a word boundary. For example, in English, the phonemic pair **/kg/** cannot occur within a word. If such a string is heard, it must come from a word boundary, e.g., as with **talk good**. On the other hand, allophones are variations on the same phoneme, such as an aspirated and unaspirated **[k]**, which are used depending on the position of the phoneme within a word or at a word boundary. However, Jusczyk et al have shown (Jusczyk et al. 1993) that infants before the age of 9 months are unable to use phonotactic information, for example to distinguish sounds of their

native language from foreign speech, and must rely exclusively on prosodic contours instead.

Table 2.1 shows the progressive development of some of the human infants' abilities which are important to the process of language development, during the first year of the infant's life. We can see that by the time the infant starts expressing the finite meaning potentials as they are defined in Halliday (1975), she has mastered her vocal tract almost fully and is able to produce canonical babblings. The canonical babble stage, which lasts usually from 6 to 7 months (Boysson-Bardies 1999), is characterized by repeated vocalization of short syllabic sequences. By that time, the infant's vocal chords have more or less assumed the adult's posture and proportions: the vocal tract has acquired its distinctive $\Gamma$-shape, the palate has lowered and moved forwared, the pharynx has opened to allow controlled air flow. At the onset of babbling, the infant has already connnected sound production with the movements of the mouth, and routinely engages in vocal exchanges with adults, reproducing intonation contours of the adult speech. The sequences are at first just one syllable long, progressing with time to at most two or three syllables. They consist of consonant-vowel (CV) phonemic pairs. The phonemes produced show a non-unifrom distribution by both place and manner of articulation, with predominantly bilabial or velar occlusives (stops) and nasal consonants and low-central and central vowels, forming syllables such as /ba/, /pa/, /gu/. Less frequent but also present are dental occlusives. Although it is not known exactly why infants babble the way they do, a simple mechanical theory can explain a lot of the distribution of sounds that we observe in canonical babbling. Infants explore the powers and limitations of tehir vocal tract as tehy would with any other part of their bodies. At first, sounds which stem from the most constricted vocal tract are easier to produce. Simple jaw movements, when fine motor control of the tongue is not yet available, produce sound sequences that go from a simple constriction to an open mouth.

During that stage, the infant must be still mainly relying on prosodic information for the tasks of word segmentation as the ability to use phonotactics and allophone statistics has not yet developed.

### 2.2.3 Nature of concept and word acquisition

Although this research is not directly related to language acquisition, the ethological studies of the grey African parrot reported in Pepperberg (1990), Pepperberg (1997), and Pepperberg (n.d.), who was learning to label novel concepts is clearly relevant to this project. Pepperberg uses novel methods of teaching in her studies, namely the model/rival technique, and social modeling. It is the social modeling theory and the referential mapping technique that present the most interest to this project, since Kismet is not quite developed enough to be able to benefit from the model/rival teaching method. The theory advises the teacher to treat the student's spontaneous utterances as meaningful, and act upon them. This, it is shown, will encourage the student to associate the utterance with the meaning that the teacher originally gave it, so the student will use the same vocalization again in the future to make a similar request or statement. The technique was successfully used on the grey African parrot Alex to give meaning to his spontaneous labels. It has also been applied in teaching language to children with developmental disorders.

This framework of first and foremost establishing a common communicative base between the teacher and the learner, or the caretaker and the infant (or the robot), is also our framework in the development of early language abilities for the robotic platform Kismet (Breazeal 2000). In the next chapter, Kismet is described in more detail.

| Age | 1-2 months | 3-4 months | 5-7 months | 8-10 months | 11-12 months |
|---|---|---|---|---|---|
| **Perception** | Discriminates between native and foreign speech based on prosodic patterns | Reacts to smiles and sad faces | Responds to emotion expressed in voice alone | Sensitive to segmental and phonotactic information in speech | Lost ability to distinguish phonemes foreign to native language |
| **Motor skills** | No control over breathing | Control of respiration | Controlled jaw then tongue movements Follows gaze direction | Gestures expressing emotions | Patterns in variation, manner of articulation, and prosodic contours |
| **Vocal skills** | | Phonation | | Intonation, vocal quality, and rhythm of native language | |
| **Imitation skills** | Imitates mouth gestures | Reproduces intonational contours | Mother-elicited imitation | | Precise gaze following |
| **Word production** | Cries | Eye-contact regulated vocal exchanges | Canonical babble Finite meaning potentials vocalised | Longer (VCV) syllable patterns | Polysyllabic utterances with varied phonemes First words |
| **Concept formation** | | | | Begins to recognize, then understand words | Novel words highlight object category Uses kind differences to infer identity of objects over time |

Table 2.1: Development of some of human infants' most relevant perceptual, motor, vocal, and linguistic abilities during the first year of life (see text for detailed explanation). Compiled from Boysson-Bardies (1999), Bloom (2000), Xu & Carey (1995), Waxman (1995), Shafer & Shucard (1995), Jusczyk (1997), and Pinker (1999).

# Chapter 3

# Kismet: the Robotic Platform

This chapter briefly describes the robotic platform for which the language development system was designed, and on which it was implemented. It is the robotic head Kismet (Breazeal 2000) – a sociable machine developed by Breazeal in the MIT AI Laboratory.

## 3.1 The Physical Robot

Kismet is an expressive robotic head, designed to have a youthful appearance (it is often referred to as an infant robot) and perceptual and motor capabilities tuned to human communication channels. The robot receives visual input from four color CCD cameras, auditory input from a microphone, and proprioceptive input from an inertial sensor. It performs motor acts such as vocalizations, facial expressions, posture changes, as well as gaze direction and head orientation. The motor systems serve the dual purpose of effectuating envelope displays on the one hand, and of steering the robot towards sensory stimuli on the other.

Kismet's control architectures run on a complex network of processors in real time (approaching 30 Hz for visual signals, and 8 kHz sample rate with frame windows of 10 ms for auditory signals), with minimal latencies (less than 500 ms). Low-level visual processing and eye/neck motor control is performed by 15 networked 400 MHz PCs running QNX. The high-level perceptual system, the motivation and behavior

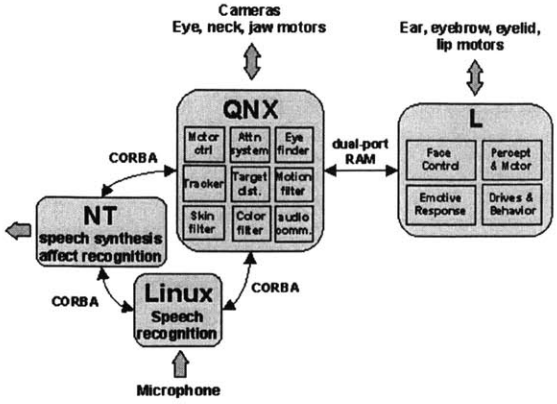



Figure 3-1: The hardware configuration of Kismet. From Breazeal (2000), with permission.

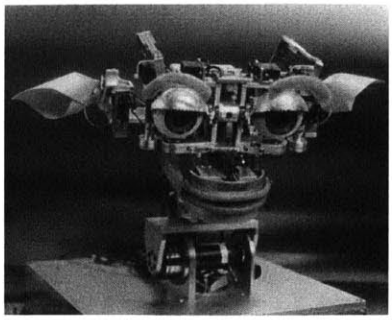


Figure 3-2: Kismet: the robotic head. From Breazeal (2000), with permission.

systems, the motor skill system and the face motor control run on four Motorola 68332 microprocessors running L, a multi-threaded Lisp developed by Brooks in our lab. Expressive speech synthesis and vocal affect recognition execute on a dual 450 MHz PC running NT, and the speech recognition system runs on two 500 MHz PCs running Linux.

## 3.2 Perception

Kismet's most important sensory information comes from visual and auditory channels. The visual hardware consists of four color CCD cameras mounted on a stereo active vision head. Two of these are wide field of view 0.25 inch CCD lipstick cameras with 2.2 mm lenses, manufactured by Elmo Corporation. They are fixed in a central position on the head of the robot. Images from these cameras are used as input into Kismet's attention system, as well as to compute distances. There are also 0.5 inch CCD foveal cameras with 8 mm focal length lenses, mounted inside each eye. They provide inputs for higher-resolution post-attentional processing.

Low-level perceptual processes extract salient features from the visual environment of the robot, such as color and motion. The attention system determines the target

of the robot's focus as a function of these salient features as well as the robot's own agenda – e.g, if the robot is currently in "need" of seeing faces, face-like stimuli will be given attentional preference over brightly colored toys. The target's features are fed into a system for higher-level perceptual processing where they are combined to form behaviorally relevant percepts. The perceptual, attentional, and behavioral systems communicate by a releasing mechanism, conceptualized after Tinbergen and Lorenz (cited in Klopfer & Hailman (1965) and in Breazeal (2000)). A releaser for a salient object or event activates when all the necessary conditions in the environment are satisfied for the response to become active.

The auditory input comes from a wireless microphone that the human wears. The processing is done by two speech recognizers. Speech recognition software developed at MIT by the Spoken Language Systems Group (SLS group) outputs low-level features such as the fundamental frequency or pitch of the signal, its energy and periodicity. Commercially available ViaVoice recognition software outputs an estimate of the phonetic sequence. The latter is extremely unreliable due to the fact that no vocabulary is available to the speech recognizer, since the robot's competencies are supposed to be those of a human infant. These output features are used by the vocal affect recognizer developed in Breazeal & Aryananda (2000). They are also the inputs into the system described in this thesis.

## 3.3 Motivation and Behaviors

Although not a mobile robot and with all computation off-board, Kismet is an autonomous agent in that it pursues its own agenda by engaging in specific behaviors which are tuned to the satisfaction of its own "goals" and "desires". The motivation is provided by a set of internal homeostatic variables called "drives", such as the level of engagement with the environment or the intensity of social play, which must be maintained within certain normal bounds in order for Kismet's system to be at equilibrium. Thus, if one of the variables shows specific understimulation or overwhelming, this triggers a behavior tuned to rectifying such symptoms and bringing

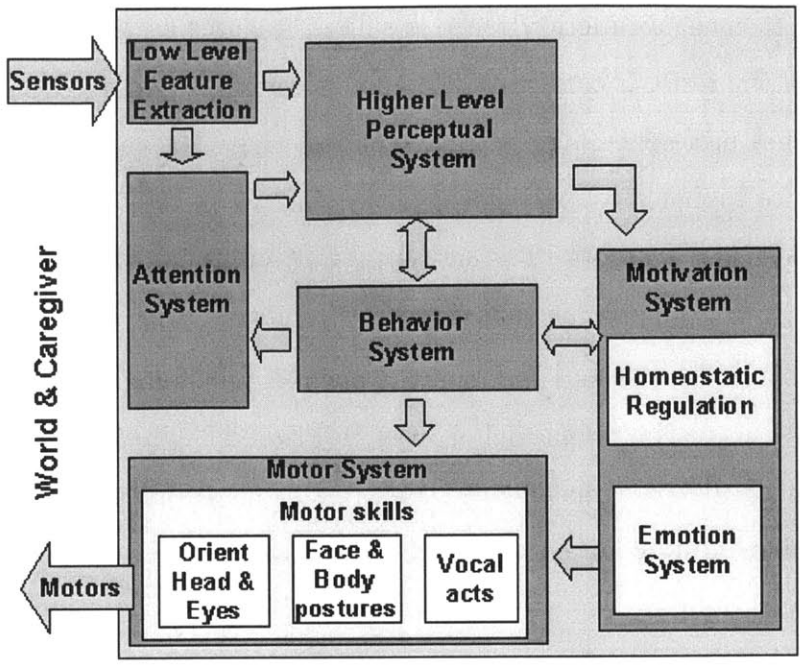



Figure 3-3: The behavioral framework of Kismet. From Breazeal (2000) with permission.

the robot's internal state back to within the norm.

"Emotions" constitute another facet of Kismet's motivational system. The robot's emotional state is modeled, after Ekman, cited in Breazeal (2000), as a point in three-dimensional space, where the axes represent affect, valence, and stance. Kismet can express, using its facial features, analogs of happiness, interest, fear, sadness, disgust, anger, surprise, and a neutral face. The choice of emotion depends on simple appraisals of the perceptual stimuli, and the emotive response may involve a withdrawal or an approach accordingly.

The behavior system provides structure and an arbiter for the robot's multiple behaviors. The latter are all self-interested mechanisms which compete with each other to be active at any one time. The activation of a particular behavior will depend on the current state of the robot's motivational system, as well as considerations of coherency, persistence and opportunism. The robot has to respond adequately to the environmental stimuli. The arbiter also makes sure that, once a behavior has been activated, it remains so for some time, preventing the robot from switching too soon between behaviors without achieving any one behavior's goals. Finally, to

prevent Kismet from being stuck in a rut, an element of randomness is introduced into the decision process. Thus the behavior system ensures that the robot may engage sensibly its complex and changing environment.

## 3.4 Motion

Kismet's motor skills include facial displays, gaze and head orientation, smooth pursuit, and vocalizations. The robot has a 15 DoF face that mirrors its internal "emotional" state expressively. Kismet can perk its ears when interested, or fold them back in anger. Each eyebrow can lower or be raised independently, allowing for expressions of surprise or disgust. Eyelids can be opened or closed, enabling the robot to blink or wink. There are four lip actuators, and a single DoF jaw that together work to lip-synch to the speech produced by a synthesizer. The synthesizer software is DECTalk v4.5, based on the Klatt synthesizer, cited in Breazeal (2000), which models the physiological characteristics of the human articulatory tract. This enables the robot to speak in a youthful tone of voice, and to vary the parameters of the synthesizer to account for variation in its emotional state.

## 3.5 Social Play and Learning

Kismet is a robot for social interaction. Its behaviors regulate its internal "drives" but are also geared towards initiating and maintaining social exchanges with humans. The expressive features on the face convey to the human teachers the "emotional" state of the robot. Kismet engages humans in social play by following the human social protocol in its envelope displays. For instance, Kismet reproduces human turn-taking displays such as looking away when starting to speak, or leaning forward to give up the floor.

Turn-taking behaviors are necessary to establish a flow of exchange between the robot and the teacher. That exchange is part of the benevolent social environment which is established to scaffold the robot's learning. The purpose of designing "for

the human in the loop" when developing Kismet was to create an environment in which learning would become easier. The usual machine learning paradigm assumes an impartial teacher who presents many examples of the problem to the learner until the learner discovers a pattern in them or becomes able to classify them correctly. Breazeal expected, however, that a benevolent teacher and a scaffolding environment would enable a machine to learn human concepts and behaviors from drastically fewer examples, if that machine is made to pick up on the cues that human infants use in their development. This idea informed the whole process of developing Kismet which was to be a robot that learns from social interactions with humans.

So far, the only explicit learning routines implemented on Kismet involved specific problems in speech recognition (see section 3.6) and individual face recognition (Aryananda 2001) from streaming video. These routines did not use the social aspect of Kismet's environment. The current project enables Kismet to actually leverage off its social capabilities to learn the basics of a human natural language from its human teachers.

## 3.6 Special Skills

Since the publication of Breazeal (2000), Kismet has been endowed with other skills, some of which have not yet been described elsewhere. Some of these have been instrumental in the development of the language acquisition system reported here. In particular, Paul Fitzpatrick developed a mechanism for extending a very small initial vocabulary with a clustering algorithm operating on a phone-based "OOV" (out-of-vocabulary) model in a speech recognizer (Varchavskaia, Fitzpatrick & Breazeal 2001). The clustering engine locates the more commonly recovered phone sequences and adds them into the vocabulary as new words. For initial phone recognition, the speech recognizer developed by the SLS group at MIT (Glass, Chang & McCandless 1996) is used, augmented with the OOV model developed by Bazzi and Glass in (Bazzi & Glass 2000). The vocabulary referred to here is that of the speech recognizer; there is no semantic content attached to the sequences of phones that constitute its entries.

The top clusters located by this procedure should be prime candidates for entering Kismet's semantic network as the language acquisition system develops. However, the system presented in this thesis does not yet integrate the above components. Rather, we use commercially available ViaVoice software for Linux to extract the phonemic sequences from the speech signal. In the future (see Chapter 7 for directions of future work) we would like to integrate the OOV model with the Protolanguage Module (Chapters 5 and 6) which could provide the initial small set of recognized words as the initial vocabulary of the OOV system.

# Chapter 4

# Robotic Behaviors in C++

This chapter explains the need for special libraries for behavior-based implementation of robotic software architectures, the Zac library and scripting language (Fitzpatrick 1997) and its extension developed specifically for the purpose of this work by me called BALZac[1], for Better Adapted to Learning Zac.

## 4.1 Behavior-Based Programming

The task of endowing a robot with a behavioral system lends itself naturally to a kind of hierarchical decomposition not necessarily adequately supported by the standard architectures of mid-level programming languages. The constraints of a control system are elements of the design, rather than the implementation stage. However, in the cases where constraints may be embodied in well-defined structural elements, such as augmented finite state machines (AFSMs) and wires in Subsumption, it can be useful for clarity and ease of programming to have those elements as building blocks of an implementation. The object-oriented approach of C++ is a good starting framework for understanding, designing, and implementing a behavioral system. Many helpful

[1]The name predates the system, naturally. Upon reading in Fitzpatrick (1997) that the Zac script was named in tribute to Isaac Asimov - the brilliant widely-read and much-loved author of science fiction classics - I immediately thought that anything I might ever write as an extension of the Zac libraries would be appropriately dubbed BALZac after Honoré de Balzac - the much duller seldom-opened and rarely-appreciated author of the classic "Comédie Humaine". The previous long-winded sentence is also in his honor (sad pun not intended).

ideas and constructs that are not directly present in the programming language itself can be incorporated into libraries of code that would be useful to behavior-based programmers of robotic agents. The Zac library and scripting language is one example of a programming language extension that makes behavior-based code more intuitive and readable, as the resulting constructs correspond to the conceptual decomposition of a behavioral system.

Earlier language support for behavior-based programming included systems such as the Behavior Language (Brooks 1990) for the Subsumption architecture, and ALFA (A Language For Action, cited in Gat (1992)) for the ATLANTIS control architecture. The Behavior Language supports real-time rules written in a subset of Lisp, which are then compiled into AFSMs. ALFA was developed specifically to address the support needed for ATLANTIS, and consisted of computational *modules* communicating through *channels*. One of the crucial components of this kind of support is to include automatic management of channels whenever modules are inserted or deleted, so that the user of the language need not restructure the control network.

## 4.2 Lateral and Zac

This section will briefly outline the Lateral architecture, and its implementation in C++ called Zac, as it was developed and presented by Paul Fitzpatrick in Fitzpatrick (1997). Lateral is a superset of the Subsumption architecture (Brooks 1986) which implements a priority scheme among the robot's behaviors. It allows for a more flexible control than Subsumption, but its original implementation (Zac) still suffers from a lack of adaptability which makes it unusable for applications where the whole repertoire of the robot's behaviors may not be known in advance. An extension to the Zac libraries called BALZac which provides for more adaptability was developed specifically for this project, and is described in section 4.4. In order to understand the additions of BALZac, we first take a look at the system it extends.

### 4.2.1 The Lateral priority scheme

Lateral is an architecture based on two types of objects: behaviors and connections. A behavior is very loosely defined as any process that runs on a robot and implements an aspect of the robot's competence. Just like in the Subsumption architecture, any behavior object may have direct connections to the robot's sensors and actuator controls. All behaviors interact with each other and the world through input and output ports accessed by connections. A behavior object also has a priority which puts it on a *priority plane*, similar to a *level* in Subsumption. Priority propagates throughout the behavioral system via the connections - it is optionally sent as extra information with any message that is transmitted between behaviors. Just like in Subsumption, behaviors are run concurrently rather than sequentially. Computing priority at every pass through the system determines which behaviors have control over (e.g., subsume) which connections.

Connections are objects which transport messages from source to target, allowing for priority arbitration. A connection has a primary source and a primary target, which can be an output and input port, respectively, of a behavior, or another connection (see figure 4-1). In addition, a connection may have any number of secondary sources and targets, which come from other connections tapping into the one in question. This kind of tapping access implements the flexible control of connections in Lateral. When the tapping source's propagated priority is greater than that of the tapped default, such a scenario is equivalent to one behavior subsuming another at the output. However, additional flexibility is seen in the case where there are several secondary sources. They may have the same priority or not, and their priorities may change at runtime, in which case the target will receive inputs from different sources over time. The principled way in which Lateral determines ownership and control of connections is established as follows.

A behavior's inputs and outputs are also objects of the connection type. They are data members of the behavior object. An output does not attach its source to anything, since it transmits messages directly from the behavior that owns it.

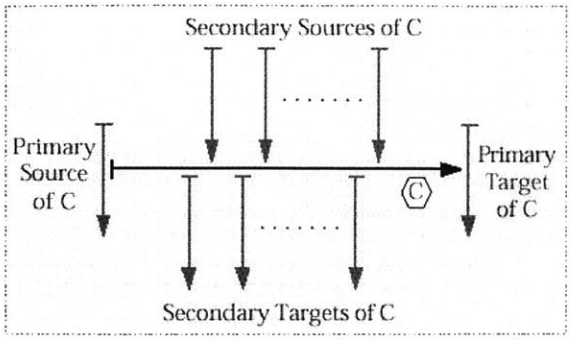



Figure 4-1: A connection object, with primary source and target, as well as a number of secondary sources and targets. C is shorthand for "Connection" and represents a global, or "free-standing" connection object, not owned by any behaviors. From Fitzpatrick (1997) with permission.

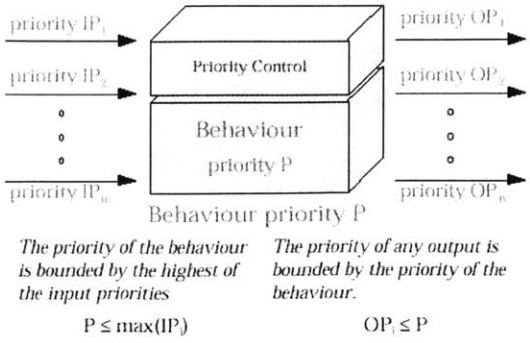



Figure 4-2: Priority arbitration in a behavior object. From Fitzpatrick (1997) with permission.

Similarly, an input does not attach its target to anything, since it is the ultimate target of any connections that terminate at it. In addition, an input does not attach its source to any connection either, leaving the initialization task to those connections which will have that input as their target. This arrangement is consistent with the view of inputs as handles in controlling the behavior which owns them. Several connections may be competing for such control, and the winner, according to Lateral's principled mechanism for conflict resolution, will attach its target to the requested behavior's input. As priorities are recomputed, the controlling connection may change dynamically but the input is always connected to something and does not need to be reconfigured.

Conflict resolution is easy when all sources have different priority. In this simple case, the source with the highest priority maintains control of the connection. When sources have the same priority, Lateral implements the ideas of Subsumption in creative ways. Connections may be type-tagged as "Default", "Shared", "Preferred", and "Replace" - corresponding respectively to defaulting, normal transfer, suppression, and inhibition in the Subsumption wires.

The figure 4-2 shows the mechanism by which a behavior's priority is decided. Priority is represented by a number attached to every behavior, and this information can be carried on top of any message transported by connections originating at that behavior. Thus in a local context a behavior can transmit priority to either "bring up" the recipient to its own level or inhibit it and any of its outgoing connections. The priority of any behavior can only be as high as the maximum input priority. However, it can be made lower than the allowed upper bound by a decision within the behavior itself. Good extensive illustrations of this principle at work are given in Chapter 3 of Fitzpatrick (1997).

Here we will briefly examine the way in which the Lateral architecture and its constructs were implemented in a library of C++ code and a script language called Zac Script.

### 4.2.2 Zac: An implementation in C++

Some of the considerations that led to the particular implementation of Lateral as Zac and Zac Script, are of little issue in the project of pragmatic protolanguage acquisition. However, at the time of development in Fitzpatrick (1997) they were of prime concern. The most important constraint in the implementation of Zac was level of operating system support available. The issue arose as the Zac libraries and tools were developed for programming an autonomous robot whose on-board computational power was minimal. This consideration led to the requirement that Zac run its own concurrency without relying on any multi-tasking in the operating system. Design decisions were also made at that point to require behaviors to be written as augmented state machines, just as in Subsumption, and to limit context switching between behaviors at state transitions only.

#### Original implementation of Lateral constructs

The original implementation of Lateral, called Zac, became the basis of all extensions developed for this project, and described in section 4.4. Zac operates through *scan*

*cycles.* Each of the active behaviors is executed for one state in turn, which constitutes one scan cycle. Active behaviors are defined as those with non-zero priority. During each cycle, before execution, all inputs of a behavior are reevaluated according to the Lateral arbitration scheme, and its priority recomputed.

Connections in Zac serve not only as information-transporting wires between behaviors, but also as a means for the system to determine the optimal order of execution for behaviors in any given scan cycle. The best order of execution for a tree-like dependency between behaviors is one where a single cycle is required to propagate a change through the height of the tree. If a cycle is present in the structure, then behaviors are ordered optimally for execution if a change in behavior X propagates through the cycle and back to X during a single scan cycle. This is achieved by following two rules, which constitute Zac's "Pull system" :

1. To update a connection: first, update all its sources which have not yet been updated during this scan cycle. Then, if the connection is an output of some behavior, update that behavior using rule 2, unless the behavior has already been updated during this scan cycle or is already being updated.

2. Before executing a behavior, update all its connections according to rule 1; then execute it.

Since the rules call each other recursively, the conditions in rule 1 are present to ensure that infinite loops do not occur.

The mesh data structure is the basis of Zac, and comprises in an extremely flexible way all the information about the connection objects that is needed to determine their relationships within a behavioral system. The mesh enables access to the following properties of a connection in a quick and efficient manner, without search:

- its primary source

- the list of all its secondary sources (i.e., all connections that have the current one as their primary targets)

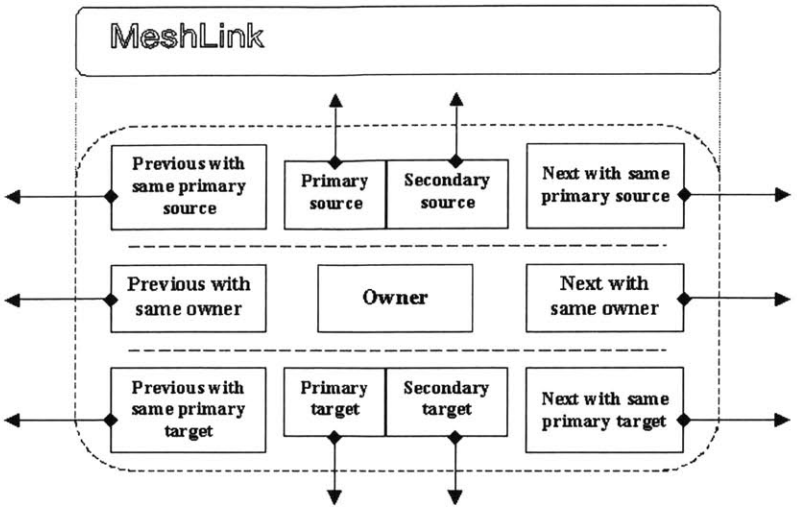



Figure 4-3: MeshLink: implementation of a connection object. Modified from Fitzpatrick (1997) with permission. Each box with an arrow is a pointer data member of the object.

- its primary target
- the list of all its secondary targets (i.e., all connections that have the current one as their primary source)
- the behavior that owns it ("free-standing" connections, which are not owned by a behavior, are allowed)
- all connections with the same owner as the current one

A mesh consists of MeshLinks (the actual library class modeling relationships between connections has the name `MeshLinkBase`), which are essentially the connection objects - see figure 4-3.

A MeshLink is an element of three separate lists: the list of connections with the same owner, the list of connections with the same primary target, and the list of connections with the same primary source. Members of a MeshLink include a single pointer to a secondary source and a single pointer to a secondary target. This is enough to give access to the list of all secondary sources or targets for a behavior because the three lists above are maintained. Naturally, a search is required to find a

particular secondary source within the list, but the list itself may be accessed without search.

Zac adds an implementation of Lateral's priority system and pull with the class `ZACMeshLinkBase` which is derived from `MeshLinkBase`. A template class `ZACMeshLink<Type>` adds the ability to carry a datum of a particular type. Connections are objects of this class template. A Mesh is a collection of MeshLinks, and a `ZACMesh` a container for `ZACMeshLinks`, from which the class `ZACProcess` is derived, adding the ability to execute an augmented state machine. Behaviors are objects of classes derived from `ZACProcess`, each of which implements a particular state machine, by taking advantage of C++ polymorphism.

### Zac Script syntax and translation

It is possible, using the libraries outlined above, to write behavior code directly in C++. The addition of a Zac Translator tool automated much of the cumbersome code generation, leaving the programmer with the task of specifying the state machines, their inputs and outputs, in a direct, easily readable, and efficient manner. Thus Lateral systems can be programmed using Zac Script, which is essentially a superset of C++. This section outlines the syntax used in Zac Script - for a formal description of the syntax, and details on the workings of the translator, please consult the original work in Fitzpatrick (1997).

The programmer using Zac Script specifies a behavior as follows:

```
BEHAVIOR Name {
    <Input Connections>
    <Output Connections>
    <Local variables>
    <Local functions>
    <State Machine>
};
```

The translator creates a new class `ZACProcess_Name` derived from `ZACProcess`, which overloads its functionality with the specified state machine. Input and output connections are declared as members of the new class. A single instance of the new

behavior is created: `ZACProcess_Name *Name;`. The translator takes care of declaring and defining all members of the new behavior in order. It also performs all the initializations necessary for the Zac engine to operate. The programmer should only take care to specify inputs strictly before outputs inside a behavior. The reason for that has to do with the hidden initialization phase, in which a valid Mesh has to be created, and connections have to be properly linked together.

Connections may also be created as global variables outside any behavior.

Zac Script syntax is shown formally and fully in Table ?? along with the BALZac Script extensions. Examples of usage are provided in the sections that follow.

## 4.3 Shortcomings of Zac for Adaptive Behaviors

The use of Zac libraries and especially of Zac Script severely limits the behavior-based programmer in her ability to create adaptive behavioral systems. The designer of Zac assumed that all behaviors are known in advance, as well as the connections between them. Although control of connections may be transferred from one behavior to another at runtime, depending on the changing priorities, these connections must be defined and initialized before execution of the system starts.

Suppose that we want a robot to be able to respond to an unpredictable change in its environment by modifying the set of its competencies. Coming up with a simple example of such a case is a non-trivial problem, since simple examples as a general rule can be hard-coded and have little use for adaptability of the kind discussed here. Nonetheless, suppose for the sake of argument, that a robot is able to perform a task that can be characterized by a set of principles and a set of parameters. For example, it can engage is a simple sorting task. The principles require it to inspect an object, perform a test with respect to some aspect of that object, and put that object in one of two piles, depending on the outcome of the test. The parameters of this task are the decision rule and the aspect of the object to which it applies. Suppose further that in the beginning, the robot "knows" how to sort objects by color into the red and green piles. That means that the robot has a decision rule for telling red from

green objects, and a state machine specifying how to go about the sorting task. This scenario can be readily implemented in Zac Script as something like the following:

```
BEHAVIOR sortByColor {
    //...connections...
    INPUT (ObjectType *, iObject, NULL);
    OUTPUT (PileType *, oPile, cPile);

    //...local variables...
    ColorType objectColor;

    //...state machine...
    @CONTROL
        CONTINUE;

    @examineObject {
        if ( iObject.Delta() ) {
            objectColor = iObject.Value()->color;
            NEXT applyRule;
        }
        else
            NEXT examineObject;
    }

    @applyRule {
        if (objectColor == RED)
            oPile.Set(redPile);
        else
            oPile.Set(greenPile);
        NEXT;
    }
};
```

The above behavior examines each new object presented to it through the input connection, applies the decision rule which happens to be trivial in this case, and sends out a pointer to the correct pile. We need a sender of objects, and a receiver of piles, which can also be trivial for the purposes of illustration. Note that `cPile` is an "outside" connection object, not owned by a behavior.

```
BEHAVIOR sendObject
{
    OUTPUT (ObjectType *, oObject, sortByColor.iObject);

    @send {
```

```
        //...somehow newObj is assigned to something somewhere...
        oObject.Set ( newObj );
        NEXT;
    }
};

BEHAVIOR getPile {
    INPUT (PileType *, iPile, NULL);

    PileType myPile;

    @get {
        myPile = * ( iPile.Value() );
        //...do something useful with it...
        NEXT;
    }
};
```

Finally, we must connect `cPile` to the right input.

```
CONNECT (PileType *,
    cPile
    POINT(sortByColor, oPile),
    POINT(getPile, iPile),
    1.0);
```

Now suppose that the robot learns of a new aspect to objects, their shape, which can be either round or square. It learns how to discriminate between the two, and we would like it to apply this knowledge to the sorting task, so objects may be sorted by shape as well as color in the future. There is no obvious way of doing that in Zac Script. Once the script is translated, `sortByColor` is declared as follows:

```
class ZACProcess_sortByColor : public ZACProcess {
 public:
    virtual int ZAC_Run ( int ZAC_state );
    virtual const char * ZAC_GetName()
        { return "sortByColor"; }

 private:
    //...local variables...
    ColorType objectColor;

 public:
```

```
    ZACINPUT_DECLARE (sortByColor, ObjectType *, iObject, NULL);
    ZACOUTPUT_DECLARE (sortByColor, PileType *, oPile, cPile);
};

extern ZACProcess_sortByColor *sortByColor;

enum {
    // Control state present ZAC_LINE_CONTROL
    ZAC_LINE(sortByColor, examineObject),
    ZAC_LINE(sortByColor, applyRule),
    ZAC_LINE(sortByColor,ZAC_STATE_COUNT)
};
```

Then the virtual function ZAC_Run is defined to specify the functionality of the behavior, including the decision rule of the sorting task:

```
int ZACProcess_sortByColor::ZAC_Run ( int ZAC_state ) {
    int ZAC_next = ZAC_LINE_DEFAULT;

    switch ( ZAC_state )
    {
        case ZAC_LINE_CONTROL:
            CONTINUE;
        break;

        // First normal state of the state machine
        case ZAC_LINE(sortByColor, examineObject): {
            if ( iObject.Delta () ) {
                objectColor = iObject.Value()->color;
                ZAC_next = ZAC_LINE(sortByColor, applyRule);
            }
            else
                ZAC_next = ZAC_LINE(sortByColor, examineObject);
        }
        break;

        case ZAC_LINE(sortByColor, applyRule): {
            if (objectColor == RED)
                oPile.Set(redPile);
            else
                oPile.Set(greenPile);
            ZAC_next = ZAC_next + 1;    //Continue to next state
        }
        break;

        default:
```

```
            ZAC_next = ZAC_LINE_NOTSET;
        break;
    }  //  End of switch on ZAC_state
    return ZAC_next;
} // End of ZACProcess_sortByColor::ZACRun
```

The change of parameters from color to shape with a different decision rule are hard to accommodate with this kind of code generation. Here, only a single instance of a behavior is defined; and the state machine is hard-coded to use only one kind of decision rule on only one type of object (`ColorType`). For a sorting behavior based on shape and a new decision rule, the program has to be stopped and rewritten, adding a new automatically generated and instantiated class `ZACProcess_sortByShape`, the definition of which will re-implement with minimal changes all of the principles of `ZACProcess_sortByColor`'s state machine. But we may not want to rewrite the behavioral system when a new competency is required; and especially when that competency is so much like an existing behavior of the robot. We would like to be able to write instead type templates for behaviors, then use C++ polymorphism as before to overload decision rules depending on the type of the behavior object. We would also like to be able to create multiple instances of the same behavior *class*, possibly with different types. Then we could define a single template `sort` behavior, and instantiate `sort<ColorType>` and `sort<ShapeType>` from it. However, one cannot write according to C++ syntax directly in Zac Script:

```
template <Type> BEHAVIOR sort {
    <Input connections>
    <Output connections>
    //... etc ...
};
```

The Zac Translator will render this as a declaration of a template class

```
template <Type> class ZACProcess_sort:  public ZACProcess;
```

but it will also attempt to create a single instance of this class as follows:

```
extern ZACProcess_sort *sort;
```

However, this will not compile in C++ as an object of class `ZACProcess_sort` must

have a Type associated with it. We must create a principled way of writing template behaviors.

The above example illustrates the limitations of assuming single instances of each behavior. Another aspect is that of requiring that connections between behaviors be known in advance. It is still possible to specify all connections in the sorting example above, so long as the typed behaviors are all known and declared in advance.

Although the examples in this section may seem contrived and confusing, these problems arise rapidly as soon as we contemplate creating behavioral systems which can learn to modify themselves in non-trivial ways. In Chapter 6 of this thesis, a behavior-based conceptual network is developed, in which both its elements and the connections between them fluctuate at runtime. This is made possible by an extension to the Zac libraries, tailored to the current project and described in the next section.

## 4.4 BALZac: Better Adaptation to Learning

This section details the extensions to Zac that make the Lateral architecture more suited to adaptive and learning behavioral systems. We call the extended Zac BALZac (Better Adaptation to Learning for Zac). BALZac was developed specifically for the project of creating a conceptual network for the robot Kismet, which is described in Chapter 6. Here, we present in turn the architectural requirements of adaptive behavioral systems, which BALZac meets, and the actual extensions forming the BALZac library implemented in C++.

Figure 4-4 shows where BALZac stands in relation to its parent Zac, and the conceptual architecture Lateral. This kind of figure is present in many reports of behavior-based architectures ever since Brooks (1986) so here we follow the trend. There is very little fundamental innovation over the original Lateral architecture. Essentially, all the functionality implemented in BALZac is present conceptually in Lateral, but not in the design of Zac. Thus, BALZac extends the programmer's opportunity to create adaptive system in the Lateral framework. As such, it nudges the paradigm slightly towards the ideal architecture in the direction of cognitive complex-

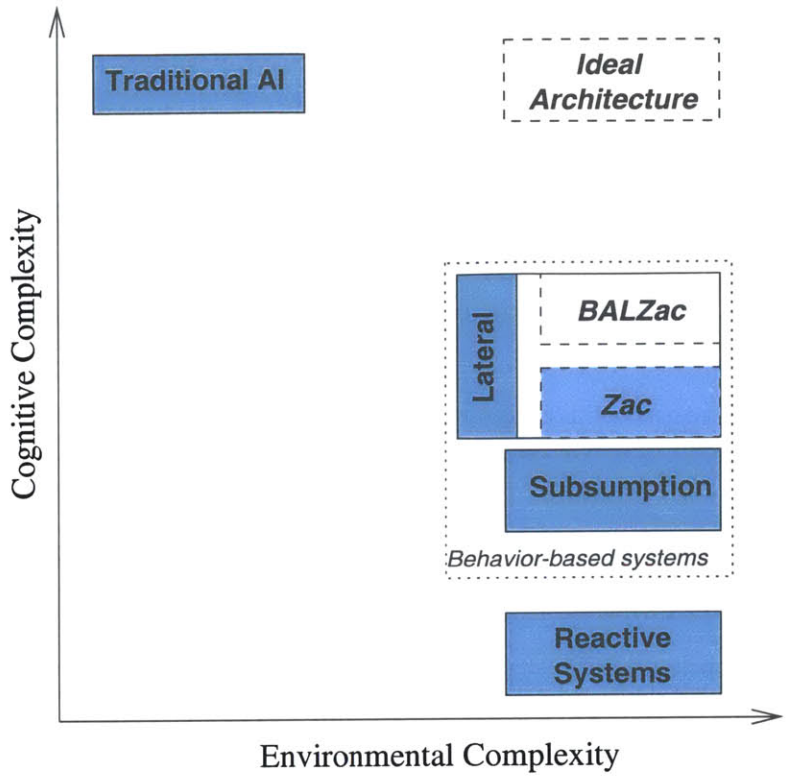



Figure 4-4: BALZac in relation to its parent Zac and other robot architectures. The relative positions of architectures are not drawn to scale; this figure should not be interpreted to mean that BALZac is close to the ideal architecture

ity. The improvement is in terms of ease of programming learning systems, which allows for greater cognitive complexity of the robots endowed with such systems. However, the architecture itself is essentially that of Lateral.

### 4.4.1 Architectural requirements for adaptive behaviors

We here formally present the requirements for programming adaptive behavioral systems using Lateral constructs. We must be able to easily and efficiently perform the following tasks at runtime (i.e., after the initial setup and initialization of the system, and after the hard-coded behaviors have started executing):

- Dynamic management (creation and removal) of input and output connections on existing behaviors

- Dynamic creation of "free-standing" connections between behaviors (objects depicted in figure 4-1)

- Dynamic management of new behavior objects from existing classes and typed templates

The last item requires certain prerequisites, e.g., the possibility of specifying typed behavior templates. We examine these requirements in more detail below.

### Dynamic management of inputs and outputs

It is possible in C++ to dynamically allocate memory to host new objects and pointers to objects that were not specified at compile time. We would like to preserve that ability in programming behaviors. Therefore, we should be able to dynamically manage new MeshLink objects owned by behaviors. In particular, we should be able to create new input and output ports on the fly, connect them to other behaviors or to "free-standing" connection objects, and delete them when they are no longer needed. It could be argued that making a connection inactive by setting its priority to zero would do the same job in terms of removing that object from the control structure of the system. However, deletion is still needed for management of dynamically allocated memory in C++, so there is an independent reason for implementing an effective way of removing unneeded inputs and outputs. The same reasoning is naturally true of other connection and behavior objects that may be allocated during runtime.

The main problem of dynamic input and output ports is that of mesh initialization. In Zac, all inputs are ordered strictly before outputs inside a behavior.

### Dynamic management of connections

In Zac, independent connection objects are declared as global variables with owner member set to `NULL`. They attach their primary source and target during the initialization phase, and their secondary sources and targets are attached during the initialization of those distinct connection objects. In BALZac, we want to be able to

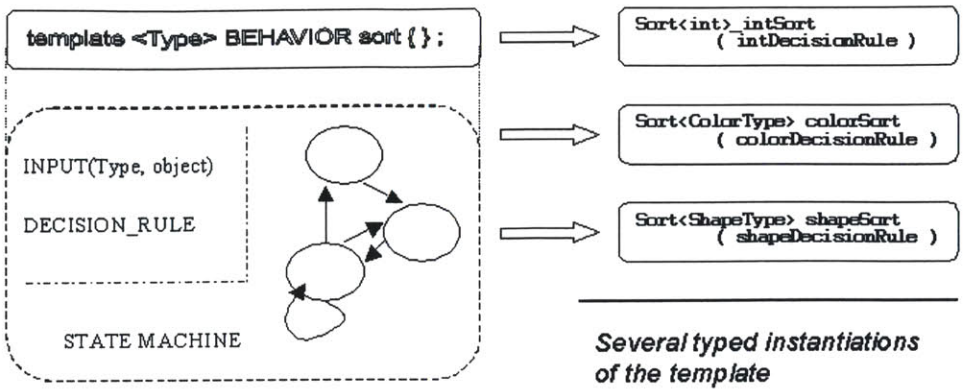



Figure 4-5: A template sorting behavior is defined. Instances of it will be typed and provided with their own decision rule. They will all share the same state machine functionality, save for those parts which depend on the decision rule.

add, manage, and remove such global connections dynamically. This ability would be useful, for example, in the case where a new output is created in behavior A, and a new input in behavior B, which have to be connected together, and we suspect that other behaviors may benefit from tapping that connection in the future. A new "free-standing" connection objects with its source at A.<output> and its target at B.<input> is then the best choice.

### Dynamic management of behaviors

In addition to specifying a behavior class and its state machine in advance and creating a single object of that class, we want to be able to create behavioral typed templates for the scenarios such as the sorting task described above (see figure 4-5 for an illustration of how such typing is used). Furthermore, we would benefit from being able to dynamically allocate and manage multiple instances of a behavior class. This requirement arose in the design of the conceptual mesh for Kismet's pragmatic language acquisition. In this project, a concept is regarded procedurally as a kind of behavior. It may be activated through input connections receiving information from the "sensors" of the perceptual and motivational systems. It may propagate its activation through output connections to other elements of the mesh. It may also output

a certain signal - its label - to the "actuator", in this case the synthesizer which generates Kismet's voice and the lip-syncing software. The reasons for thinking of words and concepts in terms of processes or behaviors, and the advantages of doing so, are examined in detail in Chapter 5. The conceptual mesh structure and algorithms are also detailed there and in Chapter 6. Here, this kind of setup is mentioned to illustrate the need for creating instances of behaviors dynamically. It is clear that as the robots learns new words or concepts, new processes must be created and readied for execution inside an existing mesh that is already running. These processes may be only slightly different from each other in terms of the state machines implementing their functionality, differing for example only by the type of their input connections. In this case they are most conveniently implemented as typed instances of a template.

Dynamic management of behaviors only suffers from initialization problems insofar as the new behavior object's connections need to be initialized. Depending on the scope in which new instances of the behavior are created, it should be possible to automate this process.

### 4.4.2 The BALZac extensions: implementation

In this section we describe the implementation of the extensions to Zac that satisfy the above requirements.

#### Overall architecture

The overall mesh architecture is inherited directly from the Zac implementation of Lateral without any modifications. All of the functionality of the requirements above is already available in C++ and only limited by the Zac Script syntax and the issues of initialization and order of inputs and outputs in a behavior. Therefore, the `BALZACProcess` class derives from the existing `ZACProcess` class and adds the ability to automatically manage the re-initializations that are required whenever input and output connections are allocated dynamically at runtime. For convenience only, we also define a derived class `BALZACVector <Type>` from the C++ Standard Template

| Class | Methods | Functionality |
|---|---|---|
| BALZACProcess | BALZAC_AddInput() | Creates a new input connection and attaches to an existing BALZACVector of inputs that is a member of the process. |
| | BALZAC_AddOutput() | Creates a new output connection and attaches to an existing BALZACVector of outputs that is a member of the process. |
| | BALZAC_RemoveInput() | Removes an input connection. |
| | BALZAC_RemoveOutput() | Removes an output connection. |
| | BALZAC_Connect() | Links an output to an input or a global connection. This is possible in Zac only as a global macro. Here, the function may be called from inside a behavior's state machine, as needed. |
| | BALZAC_Reinitialise() | Reinstates any outputs of the behavior after a new input has been added. This is essential due to the nature of the Zac Mesh construct. |
| BALZACVector | <auxiliary methods> | This class is created for convenience only. Its functionality is mostly present in the vector container of STL. |

Table 4.1: Functionality of the BALZac extensions.

Library container class vector <ZACMeshLink <Type> *> which facilitates management of both global and input and output connections. Table 4.1 details the functionality of new additions to the architecture.

### Global and behavior-owned connections

Using the vector class of the Standard Template Library in C++ takes care of the dynamic management of connections. For convenience, a new class BALZACVector<Type> derived from vector <ZACMeshLink<Type> *>, which defines auxiliary methods for insertion, deletion, and management of elements, and makes the code more readable. "Free-standing" connections may still be initialized in the global scope as before. They may also be accessed from the behaviors which own their sources or targets as the need for dynamic management arises.

### Template behaviors

Any class deriving from `BALZACProcess` can be declared as a template class in the usual manner:

```
template<class Type> BALZACProcess_extension<Type>:  public BALZACProcess
```

The methods for adding inputs and outputs to a behavior, connecting and re-initializing them are template functions in `BALZACProcess`. The extended version of the translator tool discriminates between normal and template behaviors, and is described in section 4.4.3. The only incompatibility with the original Zac syntax is that no instances of behavior are created automatically, and so the programmer must create those.

## 4.4.3 The BALZac syntax

The most important addition to the system is to enable the behavior programmer to continue writing simple and readable behavior-based code with the Zac and BALZac constructs and automate the generation of new code, while at the same time give flexibility to the programmer. After all, the idea of dynamic allocation is there for the sake of flexibility. This concern led to an extension of the Zac syntax to allow template behaviors and multiple instances of behaviors as well as all the manipulations of links within a behavior's state machine. The new BALZac Script syntax is described formally in Extended Backus-Naur Form (EBNF) notation in Table B.1 in Appendix B.

Practically, the sorting example that we used earlier may now be coded in the new version of the scripting language as follows:

```
TEMPLATE <class Type> BEHAVIOR sort {
    //...connections...
    INPUT (ObjectType *, iObject, NULL);
    OUTPUT (PileType *, oPile, cPile);

    //...local variables...
    Type objectProperty;

    //...state machine...
```

```
    @CONTROL
        CONTINUE;

    @examineObject {
        if ( iObject.Delta() ) {
            objectProperty = findProperty(iObject.Value());
            NEXT applyRule;
        }
        else
            NEXT examineObject;
    }

    @applyRule {
        if (objectProperty == TRUE)
            oPile.Set(Pile1);
        else
            oPile.Set(Pile2);
        NEXT;
    }
};
```

The translator tool interprets the first keyword `TEMPLATE` as an indication that the following behavior declaration should be made into a template class and outputs:

```
template <class Type> BALZACProcess_sort<Type> :  public BALZACProcess ;
```

The important difference from the original translation of Zac Script is that objects of the new class are not declared automatically. It therefore falls to the programmer to write:

```
INSTANCE <ColorType> (sort, sortByColor);
INSTANCE <ShapeType> (sort, sortByShape);
```

These two instance declarations translate respectively to:

```
BALZACProcess_sort<ColorType> *sortByColor = NULL;
BALZACProcess_sort<ShapeType> *sortByShape = NULL;
```

The programmer then needs to define the overloads to those functions depending on `Type`. Overloading `findProperty()` can be done directly in C++, taking care to declare these functions beforehand:

```
ColorType findProperty (ObjectType object)
{ return object->color; }

ShapeType findProperty (ObjectType object)
{ return object->shape; }
```

In the very simple world where properties are boolean, this is enough for the template to work. If the decision rule is more complicated than that, it may need to be overloaded in a similar way. For example, in the behavior declaration above, the state `applyRule` will now look like this:

```
    @applyRule {
        oPile.Set ( decisionRule (objectProperty) );
        NEXT;
    }
```

Later on, the function `decisionRule` is overloaded:

```
PileType decisionRule (ColorType objectProperty) {
    if (objectProperty == RED)
        return redPile;
    else if (objectProperty == GREEN)
        return greenPile;
    // ...
    else
        return defaultPile;
}

PileType decisionRule (ShapeType objectProperty) {
    if (objectProperty == ROUND)
        return roundPile;
    else if (objectProperty == SQUARE)
        return squarePile;
    //...
    else
        return defaultPile;
}
```

In order to enable the translator tool to read the function and global variable declarations that should come at the start of a `.zac` file, a keyword `DECLARE` is provided. The inside lines of a `DECLARE` statement are copied without modification at the start of the generated `.cpp` file.

Vectors of connections are supported by the keywords INPUTS, OUTPUTS, and CONNECTIONS. A vector of input connections may be declared inside the first section of a behavior, before the STARTUP constructor or the state machine. Although it is no longer essential to declare inputs before outputs, since the latter are re-initialized automatically upon addition of new connections, this is only true for vectors of connections. If the usual Zac declarations of INPUT or OUTPUT are used, the former must be ordered strictly before the latter for the mesh to initialize properly.

```
BEHAVIOR example {
    //input vector
    INPUTS (int, ins);
    //output vector
    OUTPUTS (double, outs);

    //Local variables, startup, state machine
    @addNewInput {
        ADD_INPUT (ins, 1);
        DIRECT_CONNECTION (int, example,
                           POINT(example, ins.back()),
                           cNew, 1.0, 0);
    NEXT;
    }
    //... more state machine...
};
INSTANCE(example);
```

To take the example of inputs for the sake of illustration, this declaration is translated as

```
BALZACVector <int> ins;
```

Originally the input vector is empty. When later a new input connection needs to be created, it is done as above in the addNewInput state of the state machine. The macros translate to calls to the method example->BALZAC_AddInput() , which performs the insertion of a new connection into the vector ins, and the method example->BALZAC_dirrect_connection() , which attaches a global connection cNew to the last element of ins (makes the last element of ins cNew's target. We assume in this example that the relevant behaviors, output and global connections are already defined and exist in the mesh.

It is possible to create new instances of behaviors at runtime. Due to the nature of the Zac Mesh, it must be done from within the state machine of another behavior, for example as follows:

```
TEMPLATE <class Type> BEHAVIOR sort {
       //...define sort behavior as before...
};
//... Note that no instances are created by default...

BEHAVIOR sortManager {
       //...input connections...
       INPUT(LINK(int), iType, cType);

       //...output connections...
       //...local variables...

       //...state machine...
       @CONTROL {
          if (iType.Value() == SORT_BY_COLOR)
               NEXT createSortByColor;
          else if (iType.Value() == SORT_BY_SHAPE)
               NEXT createSortByShape;
          else
               CONTINUE;
       }

       @createSortByColor {
          INSTANCE <ColorType> (sort, sortByColor);
          INSTANCE_INIT (ObjectType *, sortByColor, iObject, 0);
          INSTANCE_INIT (PileType *, sortByColor, oPile, 1);

          NEXT;
       }
       @createSortByShape {
          INSTANCE <ShapeType> (sort, sortByShape);
          INSTANCE_INIT (ObjectType *, sortByShape, iObject, 0);
          INSTANCE_INIT (PileType *, sortByShape, oPile, 1);

          NEXT;
       }
};
INSTANCE (sortManager);
```

Once created and initialized from `sortManager`, the two instances of `sort`, `sortByColor` and `sortByShape` will be executed from the next scan cycle. When adding behavior

instances from within the state machine of another behavior like above, it falls on the programmer to re-initialize the inputs and outputs of the new behavior objects using the `INSTANCE_INIT` construct. It is also possible to create re-initialize freestanding connections from within a behavior state machine by calling:

```
DIRECT_CONNECTION (PileType *, sortByColor,
                   POINT(sortByColor, oPile),
                   cPile, 1.0, 1);
```

This statement translates to:

```
BALZAC_direct_connection <PileType *> (sortByColor, oPile, cPile, 1.0, 1);
```

a function that sets the correct sources and destinations of all objects. The floating point parameter is the default priority of the connection, and the integer parameter is equal to one when the point to be connected is an output port, and 0 if it is an input port.

BALZac is a tool the purpose of which is to assist in the development of new behaviors for the robot Kismet. In this thesis, we focus on proto-linguistic behaviors, ranging from simple expressive grunts to the first steps towards the acquisition of concepts and words of spoken English.

The following chapters report on the design and implementation of the proto-linguistic behavioral system.

# Chapter 5

# Proto-Linguistic Vocal Behaviors

In this chapter we describe the hardwired protoverbal behaviors of the robot Kismet. Their functionality was inspired and influenced by the ideas of developmental linguists such as Halliday (1975) and McCune, Vihman, Roug-Hellichius, Delery & Gogate (1996) and researchers in the evolution of language such as Wray (2000). The set of these processes, operating together in parallel, constitutes the behavioral foundation on which more complex learning mechanisms can be built. These processes themselves are, however, human-designed and built to mirror and express vocally the existing hardwired behavioral and emotional states of the robot.

## 5.1 Protoverbal Behaviors

The vocal behavior system of Kismet is called protoverbal for two reasons. On the one hand, the behavior exhibited by the system, if observed in a human infant, would be called a precursor to language development. The goal of the system is to produce the kind of vocal output that a prelinguistic infant may produce in the age range of 10-12 months, as reported in the developmental linguistics literature (see chapter 2 for a brief overview). On the other hand, we believe that this foundation of vocal behaviors can serve as the pragmatic basis for more sophisticated natural language acquisition by the robot. In this and the following chapters the notions of "proto-linguistic behaviors", "protoverbal behaviors" and "protolanguage module"

are used interchangeably to refer to a system which produces articulate phonemic vocalizations, which may be spontaneous or voluntary, and to which meaning may be attributed by teachers, but which do not necessarily resemble words of English. Another distinguishing feature of these vocalizations is that they lack any grammatical structure, i.e., they compose a formulaic protolanguage in the sense of Wray (2000).

### 5.1.1 Why speak of behaviors?

In Breazeal (2000) a behavior of the robot Kismet was defined as a "self-interested, goal-directed entity, competing to establish the current task of the robot". The design of protoverbal behaviors presented here fits this description. Each of the independent grunts, formulaic verbal responses, or later on (see chapter 6) concepts is a self-contained, self-interested and goal directed entity. All of them compete to establish which mode of verbal expression the robot is to engage in, and what phonetic string it is to produce.

- **Self-interest**. Each verbal behavior computes its activation locally at all points in time and attempts to overwrite its connections to other behaviors and to the `SayThis` buffer containing the string that the robot will say next. A single verbal behavior does not depend on the operation of others, although it may receive other behaviors' level of activation as one of its inputs. This is the only means of communication between behaviors.

- **Goal-directedness**. The goal of each verbal behavior is to assuage the robot's communication drive by making it say something. Different kinds of vocalizations may satisfy different drives, e.g., the vocal exploration drive is satisfied by canonical babbling.

- **Competition**. The competition between verbal behaviors is regulated by the Priority Scheme in the Lateral architecture (see chapter 4 and Fitzpatrick (1997)). Each behavior assigns its own priority locally to its computed activation level. The activation level is computed according to the algorithms used

in the original implementation of the behavioral system on Kismet (Breazeal 2000)[1]:

If a behavior is active at time $t$, then the activation at time $t+1$ is given by:

$$A_{t+1} = max(\sum_{n}(R_n * G_{R_n}), A_{update,t}) \tag{5.1}$$

where $R_n$ is the value of the n'th receptor on the behavior (receptors activate when certain releasers are present),

$G_{R_n}$ is the corresponding gain, and

$A_{update}$ is calculated according to the following formula:

$$\begin{aligned} A_{update,t} &= \sum_{n}(R_n * G_{R_n}) + \sum_{m}(M_m * G_{M_m}) + \\ & success(\sum_{k} Rgoal_k) * (LoI_t - frustration_t) + bias \end{aligned} \tag{5.2}$$

where $M_m$ is the value of the m'th motivational receptor,

$G_{M_m}$ is the corresponding gain,

$Rgoal_k$ is the k'th receptor of goal releasers,

$LoI$ is the level of interest, initially the default persistance of the behavior, then calculated as

$LoI_{t+1} = LoI_t - decay(LoI_t, gain_{LoI_t})$,

$frustration$ increases linearly with time, unless the goal is achieved,

$bias$ is a constant, and

$decay(x, g)$ $= x - x/g$ if $g > 0, x > 0$, and 0 otherwise

If the behavior is inactive at time $t$, then its activation is computed as follows:

$$A_{t+1} = max(\sum_{n}(R_n * G_{R_n}), decay(A_t, G_B)) \tag{5.3}$$

where $G_B$ is the gain associated with behavior decay

[1]All references to releasers, receptors, and gains are explained more thoroughly in the next sections of this chapter.

The active update rule essentially chooses the maximum between the activation computed from pure releasers and the activation influenced also by motivation and success. The inactive update rule chooses the maximum between pure releaser activation and the behavior's own current decaying activation level.

### 5.1.2 New speech-related drives

In order to maintain automatic control of the new vocal behaviors, we have augmented Kismet's motivational system with two new drives: the Speech and Exploration drives. Both are implemented as very simle BALZac behaviors, whose state machine can be seen on figure 5-1. A drive grows until it is satisfied, i.e., reset to its minimum value.

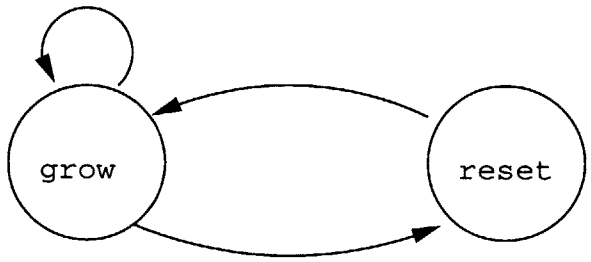



Figure 5-1: The simple state machine of a drive.

A drive may be satisfied by a number of consummatory behaviors. The Speech drive is satisfied by any behavior that produces winning vocal output. The Exploration drive is satisfied specifically by the Canonical babble behavior, which is explained in detail in section 5.2.2. Since the transition from `grow` state to `reset` state is conditional on receiving a signal from an input connection, in principle any behavior may overwrite the connection to establish the satisfaction of a drive.

### 5.1.3 Generic behavior design

Protoverbal behaviors are all based on a similar generic design, which is summarized in figure 5-2. It is a BALZac process running a state machine and augmented with internal variables as detailed below.

- **Receptors.** Receptors are sensitive to a particular kind of input, where an input may be either a global releaser or an input connection from another behavior. E.g., there may be a receptor for the level of arousal, or the level of perceived color red.

- **Gains.** Gains determine how important the receptors' response should be.

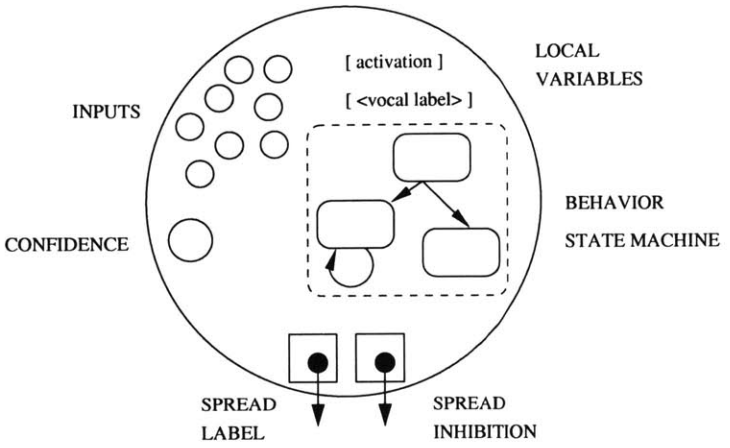
- **Elicitors.** Each behavior has at least one elicitor. An elicitor is like a complex releaser. It is computed using behavior-specific rules from receptor response and associated gains. For example, there may be an elicitor for an emotion, such as anger, or for a complex perceptual feature, such as large attention box with high motion index and high level of the color red.

- **Activation.** Activation is computed by default according to the update rules described in the previous section.

- **Vocal Label.** Associated with each behavior is a vocal label, stored as a string of phonemes and an associated confidence value. Behaviors may be indexed or keyed by their vocal labels.

- **Outputs.** Each behavior spreads out its vocal label whenever its activation exceeds a threshold. The priority of the output connection is proportional to the activation level of the behavior. The outgoing connection targets the vocalizations buffer of the system. Other outputs are possible, e.g., messages to other protoverbal behaviors in the system.

- **State Machine.** By default, each vocal behavior has a simple state machine which monitors receptors, computes elicitors and activation, updates gains and the vocal label, and spreads the label. The update rules are discussed in chapter 6. An outline of the generic state machine is presented on the right of figure 5-2.




```
//State machine
@CONTROL {
   if (update_gains_condition)
      NEXT updateGains;
   else if (update_label_condition)
      NEXT updateLabel;
   else
      NEXT activate;
}

@initial {
   <all initializations>
   NEXT read;
}
@read {
   <activate Receptors>
   <compute updating conditions>
   NEXT;
}
@activate {
   <compute elicitors>
   <compute activation>
   <set priority to activation>
   <write outputs>
   NEXT read;
}
@updateGains {
   <update Receptor gains>
   NEXT activate;
}
@updateLabel {
   <update vocal label>
   NEXT activate;
}
```

Figure 5-2: Representation of a single protoverbal behavior. The structure consists of input ports, two output ports for propagation of the behavior's activation and label, some local variables (state) and a behavior state machine. The algorithmic details of the state machine are shown on the right in BALZac with descriptions between <> substituting for function calls.

## 5.2 Kinds of Vocalizations

Kismet is capable of making three broad kinds of vocalizations, based on their expression and content. Ranging from the strictly "emotional" involuntary grunts, through exploratory canonical babble, and finally to protoverbal expressions of meaning potentials and acquired concepts. In this section, their nature and design are examined in turn.

### 5.2.1 Emotive grunts

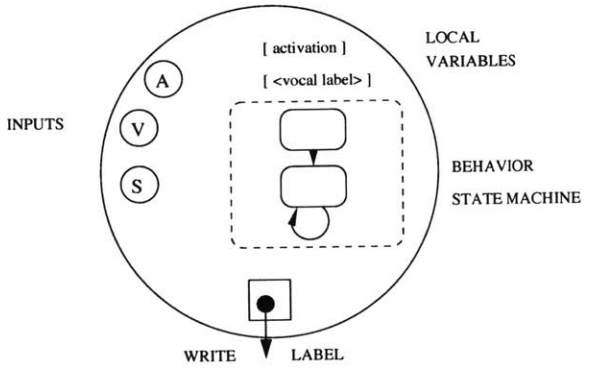
At the lowest level of the architecture, grunts are the protoverbal behaviors which are reactive, involuntary and strictly emotive. In humans, grunts are a byproduct of the endocrine mechanism which maintains the steady state of the organism. Here by a grunt we mean any such sound, for example, audible gasps.

The actual grunting sounds in humans are the result of a minor disruption of respiratory function. When breathing is normal, the vocal cords abduct during inspiration and adduct without sound during expiration. However, if the effort required for breathing is greatly increased, human infants will attempt to compensate by partially closing the vocal cords. An audible grunt is then produced during the phase of expiration. The work of breathing is increased in humans with a general increase in physical effort. Other involuntary sounds which we here lump as grunts, may be audible gasps similarly caused by air expelled against partially closed vocal cords, as a response to an endocrine signal, as for example in an acute emotional state (Boliek, Hixon, Watson & Morgan 1996).

These expressive vocalizations of exertion or emotions are soon used by infants as vocal communicative gestures, as shown in studies such as McCune et al. (1996), Roug-Hellichius (1998) and Tincoff (2001). It is believed that grunts become communicative when the infant begins to form and express conceptual content prior to learning the appropriate adult vocal forms. Thus grunts are one striking manifestation of the affordances of embodiment: the result of an individual body's hormonal signalling process, they are picked up on by other individuals of the species and interpreted as a communicative signal. They may also form the original pragmatic basis of vocal gestures.

The robot Kismet does not have a sophisticated endocrine system. It also lacks respiratory organs whose function would be made more difficult through exertion. However, in certain cases we can model instinctive vocal productions, which would serve as the same pragmatic basis for early language acquisition as grunts seem to serve in human infants. Kismet's emotional system is capable of high degrees of simplified synthetic equivalents of anger, surprise, and disgust. It also is sensitive to frustration. The level of frustration rises as a behavior continuously fails to achieve its goal. This can be used as a measure of effort on the robot's part.

The design of Kismet's new protoverbal behaviors allows for one behavior associated with each of these cases: frustration, anger, surprise and disgust. These are instances of the BALZac behavior `Grunt`, which are designed by hand, as they are to




```
BEHAVIOR Grunt {
  <private data members>
  OUTPUT (LINK(char *), oVocal, cWhatToSay);

  <auxiliary methods>

  //State machine
  @CONTROL
     CONTINUE;

  @init {
     <all initialization calls>
     NEXT;
  }
  @persevere {
     <read Receptors>
     <compute elicitor>
     <compute activation>
     <set priority>
     <write output>
     NEXT @;
  }
};
```

Figure 5-3: Example of a grunt representation as a BALZac behavior. The output is connected to the vocalization buffer. Initialisation functions set variables to default values and open log files. The main loop computes the value of **elicitor** and **activation**, sets the grunt's priority, and outputs the vocal label (**grunt**). Details of private data and methods are omitted for clarity.

model instinctive endocrinal responses. Each grunting behavior is activated through releasers associated with the emotional state of the robot (mediated via the Arousal, Valence, and Stance - or AVS - scheme (Breazeal 2000)), as is illustrated by figure 5-3. Activation is computed according to the update rules in Breazeal (2000), since these behaviors essentially shadow the emotional system of the robot.

Activation is propagated through Lateral's priority scheme, explained in chapter 4, to the behavior's connections. The primary connection of each grunting behavior targets the vocalization buffer. This connection transmits the vocal label of the grunting behavior, which is the sound that the robot will make when that grunt wins the priority competition. One issue in designing the vocal labels for grunts was that the software synthesizer can only make phonemic sounds. Therefore four short phonemic sequences were identified that approximately conveyed the grunting sounds, and they were adopted as the grunting behaviors' vocal labels[2]. A number representing the emotional state is attached to the label and communicated to the synthesizer to modify the settings of certain parameters, such as voice quality and

[2]It should be possible to evaluate the grunting sounds by assessing human response to them. A simple experiment for such an evaluation is underway

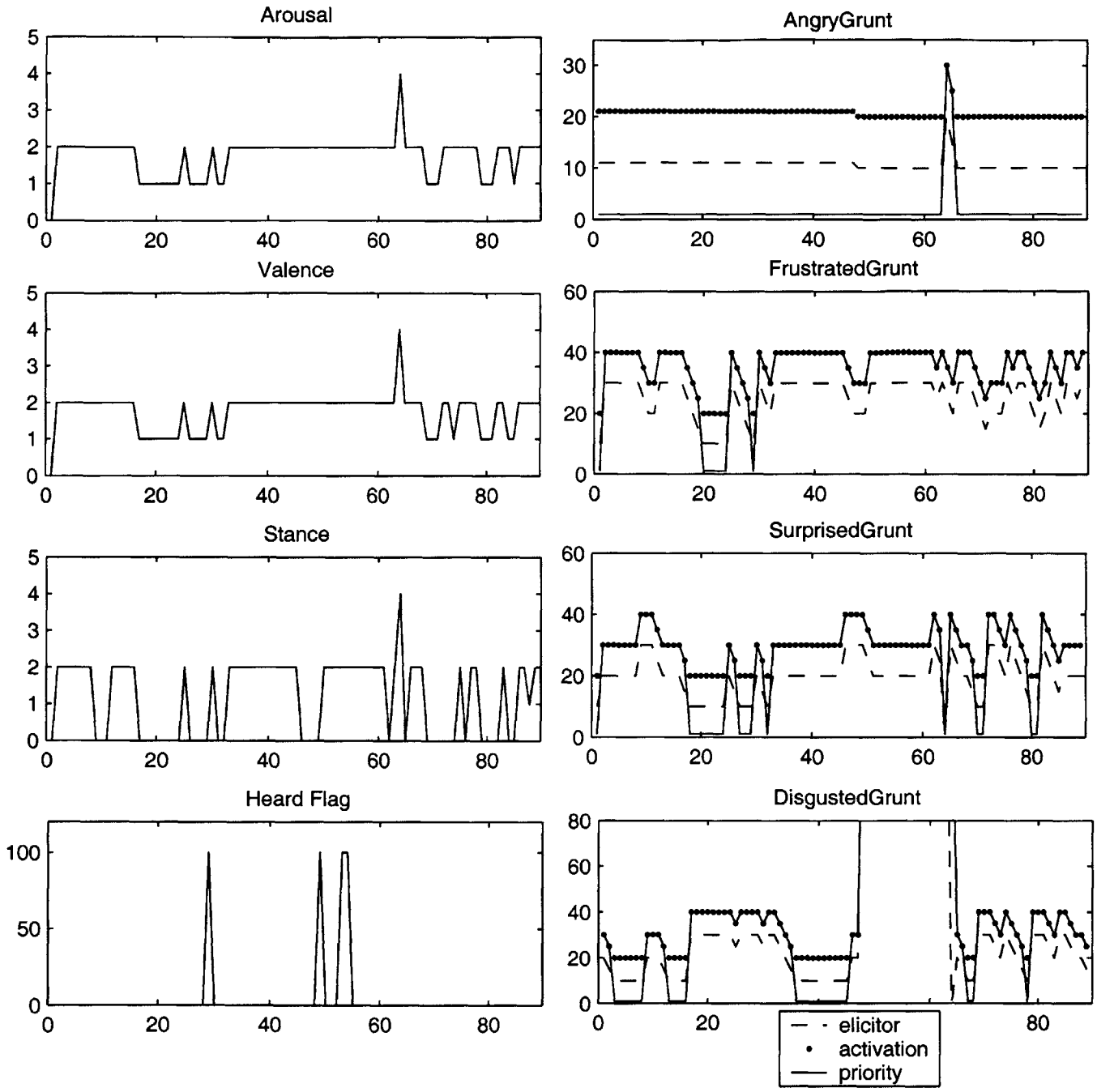



Figure 5-4: Activation of grunting behaviors over time. Data were polled every second from a 1.5 minutes of a teacher interacting with Kismet. Thresholds were set differently for individual grunting behaviors, so that Kismet produced one AngryGrunt, two SurprisedGrunts, and one DisgustedGrunt during the interaction. On the left, the corresponding values for Arousal, Valence, and Stance are given, as well as a flag showing when Kismet heard the teacher speaking.

average length of vowels, so they become more expressive of the current emotional state of the robot. Figure 5-4 shows the activation of individual grunting behaviors over time.

### 5.2.2 Canonical babble

During the sixth and the seventh months of life, the human infant goes through the stage of canonical babble. This is characterized, as explained in chapter 2, by repeated patterns of syllabic vocalizations. The syllables themselves are of the simplest type: a consonant followed by a vowel, or CV, where the consonant is most often occlusive, i.e. the result of the most restriction in the vocal tract; and the vowel is most often central or low-central, i.e. resulting from a simple opening of the mouth without necessitating any sophisticated tongue or lip movements. Developmental psycholinguists, e.g., Boysson-Bardies (1999), believe that this behavior, mostly observed when the infant is not the center of a teacher's attention and not engaged in a turn-taking situation, is a way for the infant to explore her vocal space and exercise her vocal tract. The other benefit of these vocalizations is that parents will often interpret these spontaneous babbles to be the infant's first words, e.g., Halliday (1975) and Boysson-Bardies (1999). The mechanism of learning by referential mapping (see chapter 2) starts with the teacher tagging meaning onto a spontaneous utterance. Babbles which sound like syllables from English words are more likely to be thus interpreted by adult speakers of English, and so producing them is a precursor to this kind of learning.

The robot Kismet does not have a physical vocal tract that it would need to explore in the same way. Indeed the sounds that the robot may produce are limited

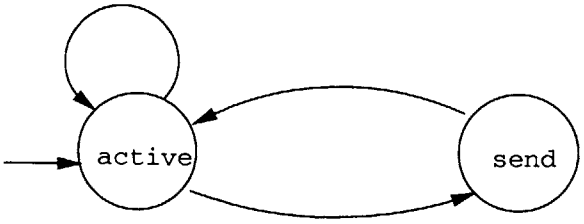



Figure 5-5: State machine for the canonical babble behavior.

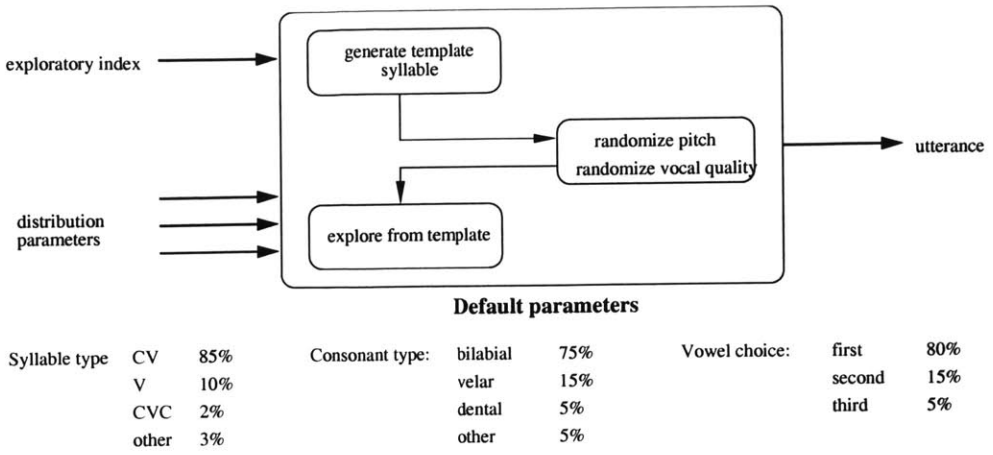



Figure 5-6: Process by which Kismet's canonical babbles are generated. The parameters of the model are the exploration index, which controls the amount of exploration from the set syllable template, and the frequencies of kinds of syllables, consonants, and vowels. Default parameters are adapted from Boysson-Bardies (1999) and Holzman (1997). A table of phonemes of American English can be found in Appendix A. The choice of vowels depends on the preceding consonant: bilabials are usually followed by central or back vowels, velars are most likely observed in conjunction with back vowels, and dentals are most often followed by front vowels.

to English phonemes preprogrammed into the synthesizer, so that little exploration is possible. Nonetheless, the robot may still benefit from exhibiting the same vocalizations, for two reasons. First, it adds to the believability of the robot's youthful character. And second, it will be a likely trigger of the mechanism for establishing a common communicative basis between the robot and the teacher through the process of learning by referential mapping, as described for infants above.

Therefore, a stochastic model of English infant babbling was implemented for Kismet. We know from Boysson-Bardies (1999) and Holzman (1997) that canonical babbling is syllabic, and that the most frequent syllable types are the open syllable (CV), and the vowel or diphthong, whereas closed syllables (CVC) are extremely rare. Certain consonant places of articulation are dominant: bilabial, dental and velar consonants (see Appendix A for a table of the phonemes of American English) are more common than palatal, alveolar, or glottal ones. In terms of manner of articulation, occlusives and nasals make up 80% of the consonants produced during

the stage of canonical babbling.

The complete model of Kismet's canonical babbling is shown in figure 5-6. Its functionality is implemented as added features to the BabylBox software (Breazeal 2000), which are activated whenever the BALZac behavior `CanonicalBabble`, whose state machine is shown in figure 5-5, wins the priority competition. When a request for canonical babble is received, the following algorithm is called to produce the randomized phonemic sequence from a template and according to the model:

```
Choose speaking rate for utterance
Set number of words in utterance to 1
Randomly choose number of syllables
Randomly choose the syllable with tonic accent
if template syllable not set
    Set a random template
For each syllable do
    if accented
        Get accent
    Explore from template given exploration index
Choose DecTalk punctuation at end of word
```

where the random template is generated according to the following algorithm (called above at `Set a random template`):

```
Get syllable type according to distribution
if type = CV
    Choose consonant according to distribution
    Choose vowel depending on consonant choice
    Randomly choose vowel length
else if type = V
    Choose independent vowel
    Randomly choose vowel length
else if type = CVC
    Choose consonant according to distribution
    Choose vowel depending on consonant choice
    Randomly choose vowel length
    Choose independent consonant
```

The template gives a "theme" to the exploration. The utterances will consist of syllables generated as variations on that theme. The exploratory babbling utterance is created from the template as outlined below. The parameter `exploration index` varies between 0 and 10 and specifies the likelihood of variation from the template.

```
For each phoneme in template
    Decide whether to keep it or vary depending on exploration index
    if keep
        Copy to new syllable
        if vowel Randomize vowel length
    else if vary
        Randomly choose variation within phoneme group
        if vowel Randomize vowel length
```

Phoneme groups, as described above and tabulated in Appendix A, are formed by place of articulation for vowels (i.e., back, central, and front vowel groups) and for consonants (i.e., bilabial, dental, and other consonant groups).

The parameters of this model are hand-coded rather than learned, which is justified for this project. Kismet's babbling is not an end in itself - the robot is not trying to either map the phonemic space of English (that is encoded in the speech recognition and synthesis software which it runs), or learn to operate its vocal tract, since it has none. Hence, the only reason for Kismet to vocalise in ways similar to human babies is that of credibility from the perspective of a human teacher, with the idea that it should bootstrap the process of referential mapping.

Figure 5-7 shows the activation of the Canonical behavior during a sample six-minute run of the system, during which a teacher was interacting with Kismet. We can observe the activation of the behavior grow as a function of the Speech and Exploration drives. One interesting anomaly happens around the 250th second on the graphs. Although the robot is hearing speech and responding, so that the Speech drive is reset, the Exploration drive continues to grow as it has been a long time since Kismet last explored its vocal space. Therefore the Canonical behavior wins the competition shortly after that. Note that the lowest graph presents the priority of the behavior, which is only set to a high value when the activation reaches past a threshold.

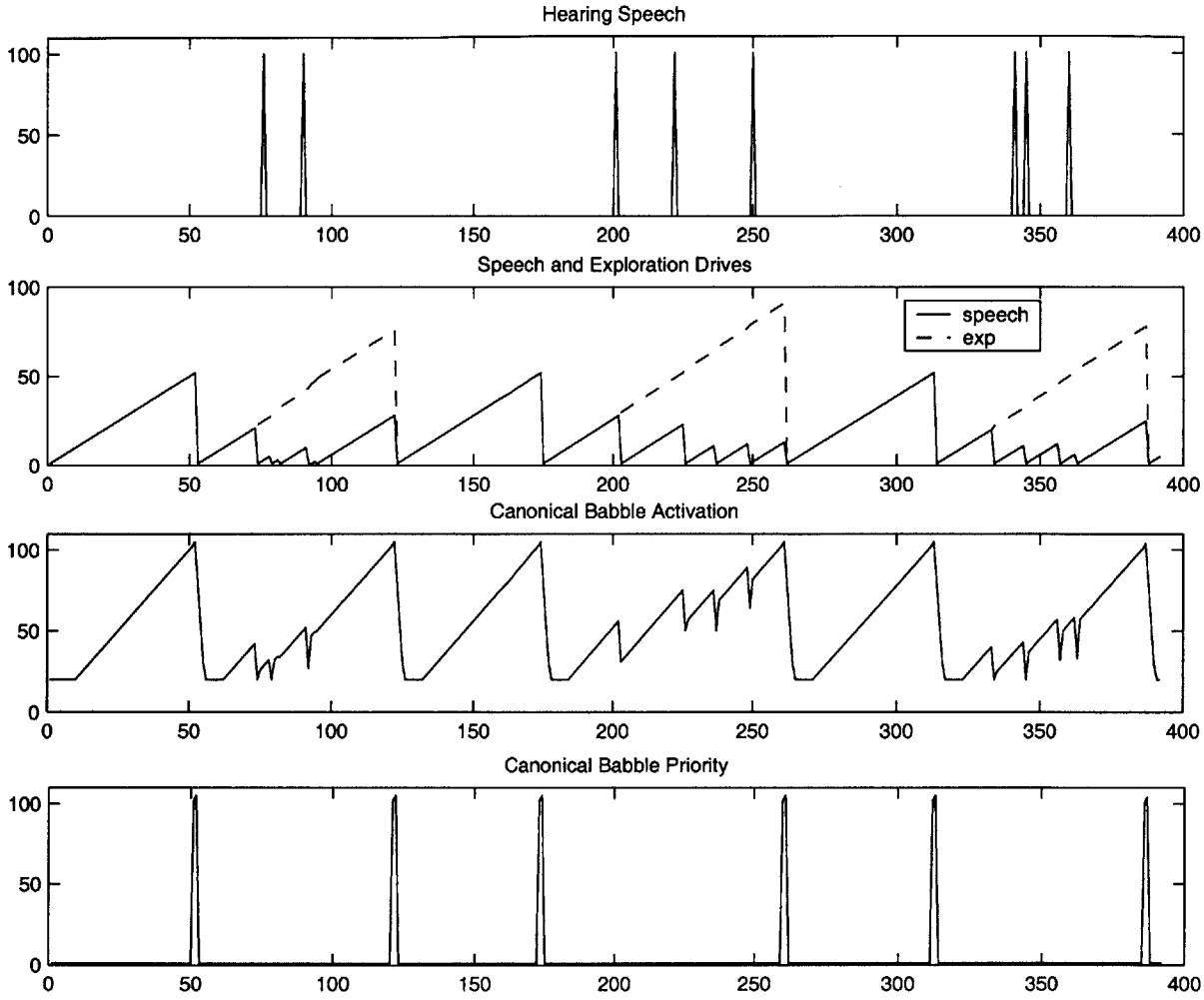



Figure 5-7: Activation of Canonical babbling behavior over time, as a function of the levels of activation of Speech and Exploration drives, shown on the second graph down. The Speech drive is satisfied (reset) whenever Kismet speaks. The Exploration drive is reset whenever the Canonical behavior wins the priority competition. The top graph shows a flag that is set whenever Kismet hears the teacher's speech. Other behaviors are activated then and the Speech drive is reset if they produce a vocal output.

| Emotion | Behavior | Proto-linguistic Function |
|---|---|---|
| anger, frustration | complain | regulatory |
| disgust | withdraw | instrumental or regulatory |
| fear, distress | escape | – |
| calm | engage | interactional |
| joy | display pleasure | personal or interactional |
| sorrow | display sorrow | regulatory or personal |
| surprise | startle response | – |
| boredom | seek | – |

Table 5.1: Correspondence between Kismet's nonverbal behaviors and proto-linguistic functions in Halliday (1975). In some fields, – indicates that there is no clear correspondence. In this case, the grunting behaviors may be active.

### 5.2.3 Formulaic protolanguage

It is when the infant starts to use her vocalizations deliberately that we may speak of communicative grunts or meaning potentials. The step is to begin using more or less articulate syllabic vocalizations as a gesture, to point at something or to request something (Roug-Hellichius 1998). This marks the beginning of the development of a formulaic protolanguage in the infant - what Halliday names Phase I protolanguage. It is characterized by a small finite set of vocalizations used deliberately and consistently to fulfill specific functions or express specific meaning potentials (see chapter 2 for an overview). Formulaic protolanguage is structurally flat and does not obey any grammatical rules; it is not combinatorial the way an adult natural language is (Wray 2000). It is therefore an easier task to learn for the human infant and for a machine running a learning algorithm.

For Kismet, it is natural to look to the existing behavioral hierarchy for some functions that a protolanguage may serve. Table 5.1 shows a correspondence of the robot's existing behavioral displays to some of the early functions of protolanguage, as identified by Halliday. This concerns particularly emotional displays, some of which do not have a corresponding meaning potential identified in Halliday (1975), but do have an associated grunt.

Each of these fixed protoverbal behaviors is instantiated from a generic template

which was described previously in section 5.1.3. Each of them has a functional or pragmatic meaning, as its output competes to be vocalized based on activation, which is established in a principled way depending on the state of response of the behavior's receptors. When Kismet engages in visual search and is unable to achieve the goal of visual search, causing it to vocalize its frustration, that sound has meaning insofar as it is used in those particular circumstances only (and thus has communicative value to the listener) and insofar as the listener chooses to interpret it as meaningful. In Breazeal (2000) experiments show that humans entrain their behavior to the robot. By treating these vocalizations as meaningful, the human teacher provides the robot with more consistent environments in which the sound is applicable.

These units of Kismet's formulaic protolanguage are called concepts. Concept instances are created and maintained from a single object `ConceptMap` which stores pointers to all of the currently available concepts. `ConceptMap` also has methods for indirectly calling updates on the gains or the vocal label of a concept. All initializations for connections between concepts and between a concept and the `SayThis` buffer are also managed from `ConceptMap`, when such connections (and possibly the concept that owns them) are created at runtime. Note that once a new protoverbal behavior has been defined, allocated, and initialized in BALZac, it is automatically added to the Mesh and is executed in turn for one state per cycle, just as any other, pre-specified behavior would. Pointers to concepts are stored in an associative map (hence, `ConceptMap`) with their vocal labels acting as keys.

The labels are relatively short strings of phonemes, possibly with stress markers. They are produced either randomly according to the algorithms of canonical babbling, or by attempting to recover and match chunks from the teachers' speech. We use the commercially available ViaVoice software to recover phonemes from speech[3], with an empty vocabulary. We bias ViaVoice to recognize shorter strings of phonemes, with a maximum of 5 syllables per recognized chunk. The error rate of phoneme recognition is very high since there is no possibility of checking it against an existing vocabulary.

[3]We also use SLS software to recover sub-phonemic features of the speech signal, such as f0, energy, and periodicity of the sound. See Chapter 3 for details.

The recovered chunks do not necessarily correspond to words. They may span across the word boundaries or be a part of a word. While this kind of performance would be unacceptable for dictation systems or other speech recognizers in which accuracy is essential, However, it may be adequate for the purpose of attaching a vocal label to a concept.

In chapter 6 we detail all the rules and algorithms for creating new concepts and managing and updating their parameters and vocal labels. Here, we simply show the kinds of protoverbal behaviors that Kismet engages in and the way they interact.

## 5.3 Overall Architecture and Communications

All of the above behaviors execute and interact in the overall architectural framework shown in figure 5-8. There are two specialized behaviors, `YarpReader` and `PhonemeReader`, which read the YARP (Fitzpatrick & Metta forthcoming) connections and copy results into global `Releaser` and `CurrentHeard` buffers. A single `YarpWriter` behavior is responsible for sending a speech request over to the robot.

All of the protoverbal behaviors (rounded boxes on the figure) have access to the global releasers and the currently heard string of phonemes. Since all the behaviors in the system are executed for one turn at each Zac cycle, there is no concern for access violation, and we can make sure that every behavior's activation level is based on the same inputs at any time. Not shown in the figure are possible connections between protoverbal behaviors, where the activation level of one may become an input message for another. This possibility is relevant for concept formation, which is discussed in chapter 6. In such cases, receptors for those kinds of inputs are created in the target behavior, and their response is accounted for in the calculations of elicitors and level of activation.

The nature of the speech request is determined by competition among individual protoverbal behaviors implemented through the Lateral priority scheme. To that effect, relevant behaviors write their vocal labels to free-standing connection objects, shown as small circles on the figure. The behavior with the highest activation, and

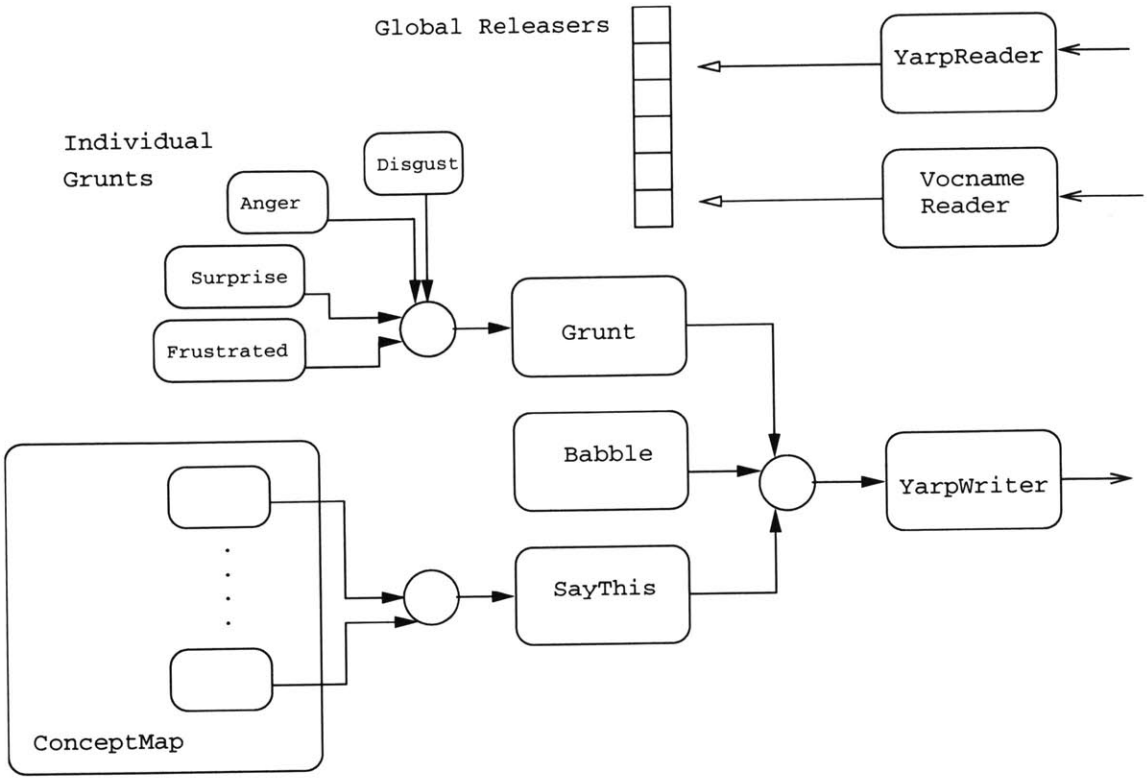



Figure 5-8: Overall architecture of Kismet's protoverbal behaviors. Rounded boxes represent instances of BALZac behaviors. Small circles represent free-standing connections between behaviors. Smaller rounded boxes inside **ConceptMap** are pointers to instances of the **Concept** behavior. **Anger**, **Surprise**, **Frustrated**, and **Disgust** are individual grunting behaviors. **Grunt** is a dummy behavior acting as a buffer between them and **YarpWriter**.

therefore the highest priority, will succeed in overwriting all other request strings with its own, which **YarpWriter** will end up passing on. A speech request may be a request for a grunt, a babble, or a "word" - i.e., a phonemic string that is attached to one of the concept behaviors. Competition happens in two stages. First, the most active grunt writes its request to the **Grunt** buffer and the most active concept writes its label to the **SayThis** buffer. Then, the most active of the three types of request buffers writes its output to **YarpWriter**. Any of these behaviors only produce output when activation is above a threshold (determined empirically), so some of the time, the Protolanguage Module does not produce an output, and the robot remains silent.

Verbal behaviors are implemented in BALZac (see chapter 4) on one PC run-

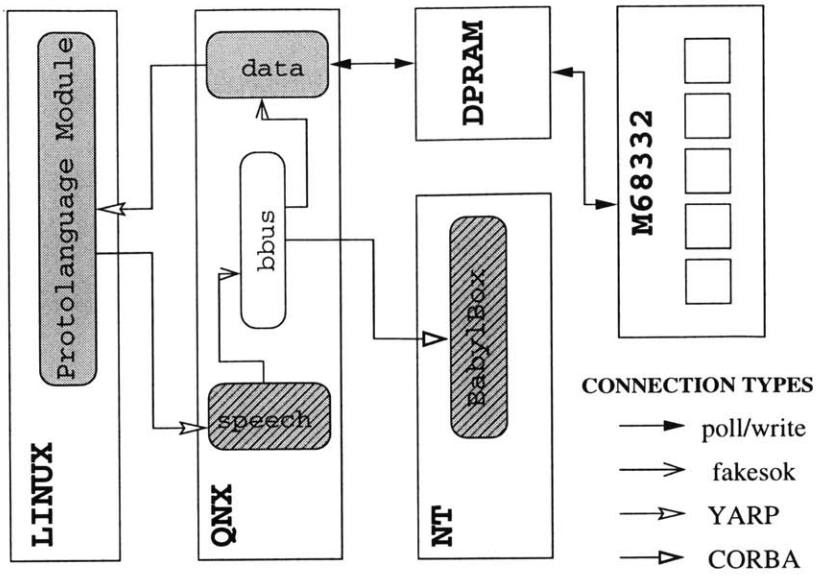



Figure 5-9: Communications in the heterogeneous network. Filled boxes are modules implemented for this project. Hashed boxes are modules that were modified or only partially implemented by me for this project. Types of arrows indicate types of connections used in the implementation of the parts of the verbal behaviors system residing on different parts of the heterogeneous network. Some details have not been included for simplicity of the diagram. In particular, existing perceptual and behavioral systems on the M68332 and QNX sides are not shown. Also, the `bbus` does not write directly to `BabylBox` but through a server not shown in order to avoid clutter.

ning Linux. The Zac libaries implement their own version of concurrency and enable any behaviors to run virtually in parallel on a single machine. They constitute the `Protolanguage Module` which can be seen on Fig. 5-9. The figure shows the types of connections that are used in the system for data and command communication between those parts which are implemented on different physical platforms in Kismet's heterogeneous hardware network. The verbal behaviors are influenced by data on the robot's current perceptual, emotional, and behavioral state. These data are polled from the dual-port RAM (DPRAM) and from existing streams of perceptual data on the QNX side of the system and sent to the verbal processes via YARP connections. Whenever the verbal behaviors determine what action (e.g., babble, grunt, or say something) should be taken next, the resulting `SPEECH_COMMAND`, and possibly a phonemic string are written to the behavior bus (`bbus` on Fig. 5-9) via a receiving YARP connection which pipes its output to another stream. The message broadcast

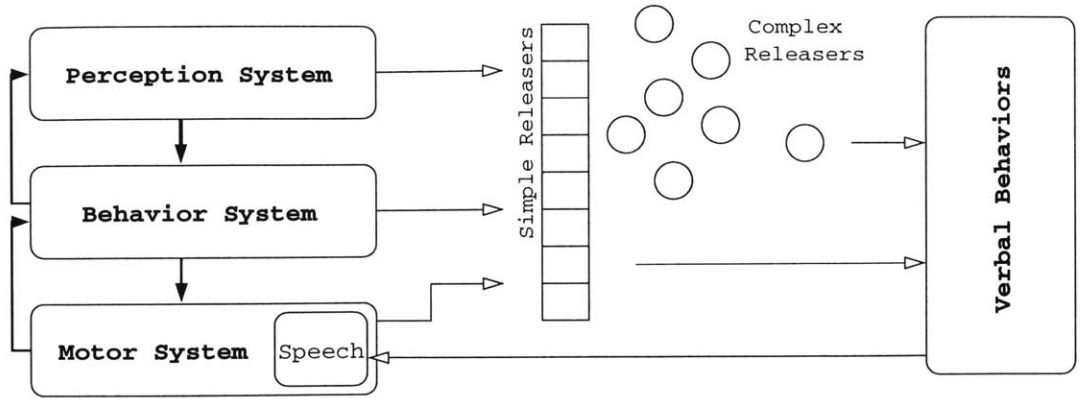



Figure 5-10: Control and data flow between verbal and other behaviors in Kismet's architecture. Dark arrows indicate combined control and data flow between old components of the architecture. Light arrows follow the data flows only between the verbal behaviors and the rest of the system.

from `bbus` is picked up and passed on to the `BabylBox` application running on the NT side of the project. There the sounds finally materialize using the DecTalk4.1 software synthesizer (more details on the hardware supporting Kismet can be found in chapter 3).

Perhaps more importantly, the figure 5-10 represents the way data and control flow between existing software components of Kismet's architecture and the verbal behaviors developed here.

The Perception, Behavior, and Motor Systems communicate the current values of `Simple Releasers`, implemented as variables of global scope, to which any component of `Verbal Behaviors` has access. `Complex Releasers` are computed from these and also become inputs to the new `Verbal Behaviors`. Finally, the outputs of the system are written directly to the speech stream, overwriting any existing value with the one determined by `Verbal Behaviors`.

The entire protoverbal system shown on the right of figure 5-10 includes the implementation of algorithms for concept and vocal label acquisition and updates, to which we turn in the next chapter.

# Chapter 6

# Mechanisms of Early Concept and Label Acquisition

In this chapter we finally present the algorithms by which Kismet acquires new concepts and attributes vocal labels to them, as well as those used for updates and maintenance of the existing concept map. We describe some preliminary results from simple experiments with the robot and examine the performance of the system.

## 6.1 Requirements for Acquisition Methods

The scope of the project was to provide a framework and a pragmatic, behavior-based approach to the problem of early concept and vocal label acquisition. Therefore, emphasis was placed on the architecture of the system rather than on the machine learning algorithms involved. Consequently, the methods described below are hardly state-of-the-art algorithms which may be used in machine learning research. They are often meant as placeholders for more sophisticated models.

Nonetheless, the acquisition algorithms have to abide by certain requirements in order to make possible experimentation in our domain. These broadly fall in 3 categories:

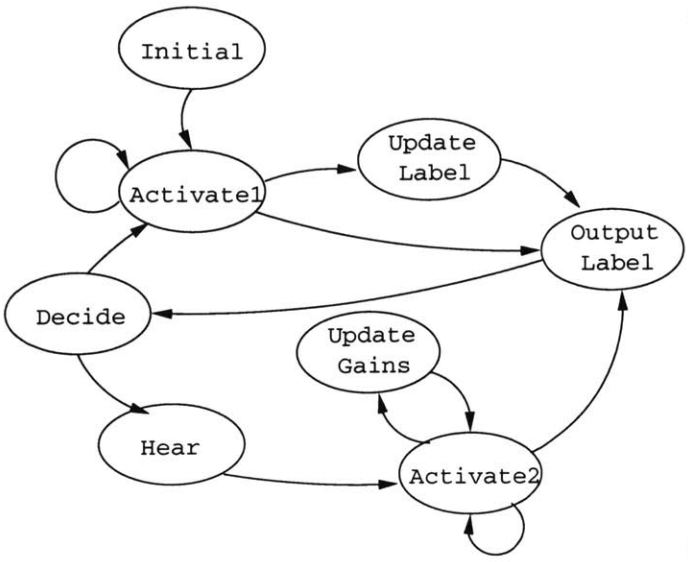
- **Online operation in real time** - We must be able to demonstrate acquisition of a new category and its vocal label, or update of the label for an existing

category within the scope of a short human-robot interaction, i.e., several minutes. We also must demonstrate real time responsiveness of the system. This precludes the use of any batch algorithms, and reduces data collection to a minimum.

- **Local computation** - We postulate that each concept is a self-sufficient goal-directed behavior. Therefore any computation in which it engages must be done locally with available inputs only. It is possible to access a limited number of global variables, which represent the global state of the behavior system, akin maybe to the level of serotonin in a human brain (i.e., not something that is computed locally, but provided by the physicality of the system). However, we must forgo the use of any machine learning techniques which rely on global search.

- **Adaptability and persistence** - Perhaps obviously, these mechanisms must provide for enough adaptability of the system to create new concepts and update existing labels. At the same time there should be enough persistence in the existing behaviors to allow "good" acquired concepts to be relatively stable.

These requirements set the research presented here apart from the current general tendencies in the field of machine speech and language recognition, where extensive data collection and powerful global searches are employed as a general rule, as in Zue et al. (2000). They also mean that we cannot apply the same standards of accuracy to the experiments presented below, also because all learning is done from the starting point of an essentially empty vocabulary. Objective recognition measures will of necessity be very low. Neither do we focus on theoretical investigations, such as the statistical analysis of Siskind (1995) to show robust language acquisition, as those kinds of models always involve compuatation over a global matrix of data. The limitations of the current project, as well as possible research directions to overcome them are discussed more thoroughly in Chapter 7.




```
BEHAVIOR Concept {
  int threshold;
  double activation;
  vocname vocalLabel;
  double vocalConfidence;

  MatchFinder bestMatch;
  vocname result;

  vector <Receptor> myReceptors;
  vector <double> myGains;

  //...public inline functions...

  INPUT (LINK(double), iCheard, cCheard);
  INPUTS (double, varReceptors);

  OUTPUT (LINK(char *), oVocal, NULL);
  OUTPUT (LINK(double*), oMatchResult, NULL);

  //...State machine...
};
```

Figure 6-1: State machine of a `Concept` behavior. See chapter 4 for details of BALZac behavior execution.

## 6.2 Concept Methods

The `Concept` class implements the functionality of a single vocal behavior. Its overall architecture was briefly explained in section 5.1.3, and its state machine is presented in more detail in figure 6-1 here. After initialization, at which point the `Concept` object is established in the Mesh and connections are setup, the behavior can be in one of the seven states. By default the behavior returns to the state specified in the `NEXT` statement. On the figure, two non-default transitions are shown from the `Decide` state: to `Activate1` and `Hear` states. They are determined as follows: after the `OutputLabel` state has executed, the transition goes to `Activate1` unless the behavior receives a signal that there is a new speech input, in which case the transition goes to `Hear`.

`Activate1` computes the behavior activation according to the generic update rules specified in section 5.1.1. If the activation is above a threshold, it transitions to `OutputLabel`, whence the vocal label of the behavior is sent out. Otherwise, the next state remains the same, unless the condition for label update is satisfied. Then the transition function leads to the `UpdateLabel` state, the implementation of which is presented in section 6.2.2.

### 6.2.1 Attribution of label

When new speech input arrives, this activates the `Hear` state of every `Concept`, which attempts to match the heard phonemic string to the behavior's vocal label. The match value is computed based on the vocal label as a template against which to match, the confidence value for the behavior's own vocal label, and an empirically determined global phoneme confusion matrix:

$$BestMatch(input, template) = max(MATCH(input, template)) \tag{6.1}$$

where $MATCH(input, template)$ is a vector whose elements are best matches given a certain window size (window size is measured in phonemes and not in single characters):

$$MATCH(input, template) = \frac{1}{n} \begin{bmatrix} Match_m(input, template) \cdot m \\ Match_{m+1}(input, template) \cdot (m+1) \\ \dots \\ Match_n(input, template) \cdot n \end{bmatrix} \tag{6.2}$$

where $m$ is the minimum allowed window size and

$$n = min(MAX\_WINDOW\_SIZE, length(template))$$

The minimum window is set so that spurious one-phoneme matches are discarded. The matching measures returned by $Match_i()$ are scaled by the window size in order to bias in favor of longer substrings. $Match_i()$ itself uses the standard brute-force string searching algorithm. The implementation of $Match_i()$ returns a result of type `class MatchResult`, which defines the following data members:

```
class MatchResult {
    double distance;    //inverse of matching level or 0 for perfect match
    int inStart;        //match starting point in input
    int temStart;       //match starting point in template
```

```
    int window;          //window used in this match call

    //...methods...
};
```

*BestMatch*() implements a method of the `class MatchFinder` with the following structure:

```
class MatchFinder {
    vocname result;              //winning substring
    MatchResult bestResult;      //winning match properties
    double confidence;           //confidence value on the match

    //...methods...
};
```

We can see the result of a *BestMatch*() call in figure 6-2. Arrows point to the starting and finishing locations of the matching substring. It is a perfect match, so the distance is set to 0.

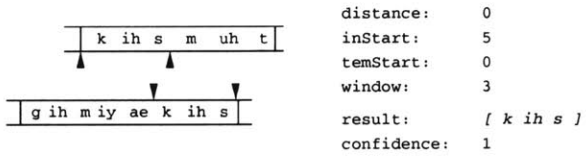



Figure 6-2: Result of a call to *BestMatch*().

`Hear` automatically transitions to `Activate2` which computes the activation level of the behavior based on the match determined earlier and the values of the behavior's receptors. If a discrepancy between the response to receptor values and the response to the speech input is above a threshold, and the confidence in the vocal label is high, `Concept` transitions to the `UpdateGains` state, the functionality of which is explained in section 6.2.3.

If the activation level reaches above a threshold, the behavior transitions to the `OutputLabel` state, which sets the behavior's priority to a value dependent on its activation and the confidence of the match, sets the output priority to the behavior's

priority, and outputs an inhibitory signal at that priority level. The inhibitory outputs of each concept connect back to the `HeardThis` behavior. Such a signal received by `HeardThis` propagates further to `ConceptMapManager` and inhibits the default creation of a new `Concept` when a new string of phonemes is heard. The implementation of `ConceptMap` is examined in detail in section 6.4.

### 6.2.2 Label update and confidence

Whether or not the vocal label of a `Concept` should be updated is determined by a combination of conditions on the `Concept`'s confidence $C$, the confidence $C_{heard}$ in the accuracy of the heard phonemic string, the value of the best match between them and the current activation $A$ of the `Concept`. The update label value $U_L$ is compared against a threshold to decide whether an update should take place (where $k$ is a scaling constant, and $\theta$ the activation threshold, $d$ is the distance measure between the two strings in the result of the $BestMatch()$ call):

$$U_L = \begin{cases} k \frac{C_{heard}}{C} d & : \quad A > \theta \\ 0 & : \quad otherwise \end{cases} \tag{6.3}$$

If the `Concept` is active, then $U_L$ will be proportionate to the confidence attached to the heard string, and inversely proportionate to the goodness of the match and to the level of the `Concept`'s own confidence. If it is not active, there is no reason to update the label. It may be a good reason, however, to update the gains, in the cases where there is a good match but no activation, as is explained in section 6.2.3.

These computations depend on the confidence measure on the input speech string. This is computed in the `ComputeConfidence` state of `HeardThis` in the following way:

$$C_{heard} = default - \frac{1}{n} \sum_{n} \frac{1}{d_n + 1} C_n \tag{6.4}$$

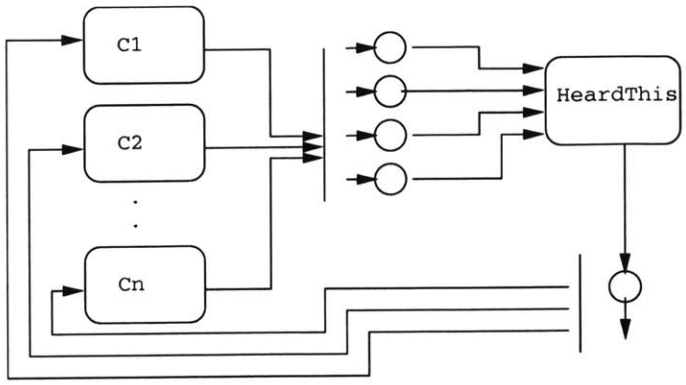
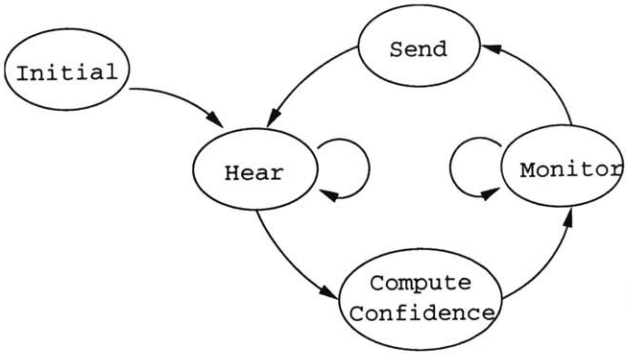



Figure 6-3: Concepts signalling the results of phonemic matching, and the associated confidence measure. Boxes are behaviors and small circles are free-standing connections. The arrows showing the transmission of signals from concepts (`C1` through `Cn`) do not point at any connection in particular, since their target will depend on the activation level of the behavior. `HeardThis` outputs its confidence measure to a single connection, which is tapped by all concepts.

| | | |
|---|---|---|
| where | $default$ | is a constant |
| | $d_n$ | is the relative distance measure of the match between *heard* and the $n$'th template |
| | $C_n$ | is the confidence measure for the $n$'th template |

and the sum is taken over all templates that had at least a minimal match to the heard string. The individual $C_n$ and $d_n$ are transmitted to the `HeardThis` behavior through incoming connections. It could hardly be considered local computation, if `HeardThis` had to constantly manage the connections to reflect the number of current concepts, but there is no need for that. Since the computed confidence value $C_{heard}$ should not be less than 0, we can simply create an `INPUTS` vector with a maximum number of elements $N$ in the initial state of the behavior (see figure 6-4 for the data members and state machine of `HeardThis`). We then let the active `Concept` behaviors compete to overwrite those connections only. So long as `HeardThis` receives the signals from the first $N$ winners of the competition, its confidence measure is computed in a principled way, as illustrated on figure 6-3.

Note that $C_n$ are used to compute $C_{heard}$ but $C_{heard}$ is needed to determine the




```
BEHAVIOR HeardThis {
  vocname myHeard;
  double cHeard;
  double inhibitionLevel;
  int threshold;
  int timer;

  INPUT(LINK(int), iInhibition, NULL);
  INPUTS(LINK(double*), inCandD);

  OUTPUT(LINK(double), oCheard, cCheard);
  OUTPUT(LINK(int), oRequest, cCMAPRequest);

  //...State machine...
};
INSTANCE(HeardThis);
```

Figure 6-4: The **HeardThis** behavior and its state machine.

update rule for the vocal label of concept $n$. However, this situation does not start an infinite regression, as there is a clear time difference between the activation of the two connections, so that when $C_{heard}$ is received back at **Concept**, the behavior has switched to another state. The interaction between a **Concept** behavior and **HeardThis** for a sample section of time is shown on figure 6-5, starting from time $t$ when a new phonemic string arrives and through time $t+5$ when the inhibition signal finally reaches **HeardThis**. It will be time $t+6$ (not shown) when the inhibition signal goes through to **ConceptMapManager** and its timer is reset. It has been determined experimentally that the average scan cycle execution for the complete module of protoverbal behaviors is approximately 520 msec, so that the system response in 6 steps takes approximately 3.1 seconds and is still fast enough to provide for a natural interaction.

### 6.2.3 Parameter update

All updates to the behavior's parameters take place in state **UpdateGains**. The vector of gains in each **Concept** is what determines the level of activation of that **Concept** when the relevant releasers are present (see section 5.1.1), since the behavior's **Receptors** output only boolean values when activated. When the behavior is created, the gains are set to default values, which must be updated to arrive at some consistent representation of releaser properties for that behavior. The signals that the **Concept** has

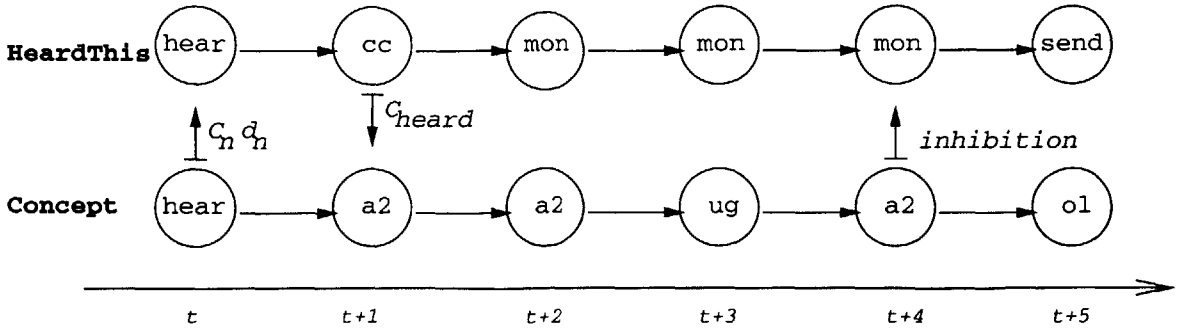



Figure 6-5: Timings in a sample sequence of state transitions of `HeardThis` and a `Concept`. The transitions are shown horizontally for each behavior. The abbreviations stand as follows: `a` for `Activate`, `ug` for `UpdateGains`, `ol` for `OutputLabel`, `cc` for `ComputeConfidence`, and `mon` for `Monitor`.

access to at this point in the state machine include its own activation $A$, the result of the matching process between its vocal label and the input phonemic string, its own confidence in its vocal label $C$, a measure of confidence in the accuracy of the string heard $C_{heard}$, and the current presence or absence of relevant releasers. Out of that information, the behavior must derive a reinforcement signal for the gains update rule. This signal $U_G$ will be computed as follows, where $d$ is the distance measure returned by $BestMatch()$, which is equal to 0 when the match is perfect:

$$U_G = \begin{cases} -\,k\,C_{heard}\,C\,d & : \quad A > \theta \\ \frac{C_{heard}\,C}{d+1} & : \quad otherwise \end{cases} \tag{6.5}$$

This rule allows for both Type I and Type II errors: when the behavior is active even though something other than its label is heard, and when the behavior remains inactive and its label was heard. However, in the first case reinforcement should be weaker since it is quite often the case that the speech the robot hears is not describing any immediate features of the environment. Therefore we scale this type of signal by a small number $k$. $U_G$ is then used in the update rule for each of the gains in the vector:

$$G^i_{t+1} = G^i_t + \alpha\, R^i_t\, G^i_t\, U_G \tag{6.6}$$

This will not change the value of $G^i$ (the $i$th element of the gains vector $G$) if its corresponding `Receptor` $R^i$ was not responding.

The simplicity of the update rules is possible because of the finite and very precisely described set of pre-existing releasers on Kismet. For a more scalable system, if we wish to maintain natural-time responsiveness, we will want to consider more sophisticated techniques.

## 6.3 Preliminary Results from "Fixed" Concepts

Since the robot Kismet already has an involved behavior system, developed in Breazeal (2000), we have added a number of "fixed" `Concepts` to shadow the workings of those behaviors and at the same time provide the protoverbal behavior functionality. In particular, we choose certain consummatory behaviors which already involve envelope displays on the robot. They are the `greet`, `flee`, `avoid`, `seek`, `orient`, and `express` behaviors. They are created at the `Initial` state of the execution of `ConceptMapManager` and their `Receptor` vectors are set up to respond to those releasers, to which their prototypes already respond. The only difference is that instead of sending commands to Kismet's motors, the shadow protoverbal behaviors send their vocal labels to the speech synthesizer. These fixed protoverbal behaviors have some correspondence with Halliday's meaning potentials (see Chapters 2 and 5). They are the finite set of grammatically non-mutable, fixed meanings which represent the basis of Kismet's formulaic protolanguage.

Figure 6-6 and table 6.1 show some data from an experiment, where a human teacher was explicitly trying to make the robot attach the label "look here" to the `orient` behavior. In table 6.1 we see an uneven performance. Poor phoneme recognition affects the results, although the main problem lies in the lack of a principled way to retain good matches. The update rules in the previous section performed irregularly on this run, with a good match to the desired result at cycle number 200 being discarded later in favor of poorer approximations. We can thus see poor compliance with the third category of initial requirements: that the update methods must

| cycle: | heard string: | template: | C: | d: |
|---|---|---|---|---|
| ..... | | | | |
| 100: | t h uh k ih ae | uh k iy | 0.3 | 3.5 |
| 200: | l uw d | l uh k ih | 0.9 | 1.2 |
| 300: | uh | h uh k ih | 1.1 | 2.8 |
| 400: | h eh n k ih t | uh k h ae | 0.8 | 4.1 |
| 500: | h uh k ih xr r | uh k h ih | 0.7 | 3.5 |
| ..... | | | | |

Table 6.1: Results of a label teaching scenario. Phoneme and word recognition is unreliable, which affects the results.

show not only adaptability but also persistence, once a good enough match has been found. Similar results were obtained for the `seek`, `greet`, `avoid`, and `flee` behaviors. The `express` behavior reverts to the grunting mechanism described in Chapter 5 as it represents the direct expression of the robot's motivational state.

Figure 6-6 shows plots of the `orient` protoverbal behavior's response in ragged lines superimposed on the values of some releasers from Kismet's pre-existing behavior system. The delay, especially clear on the top graph, comes from the execution time of the entire scan cycle and from the delays generated by the CORBA and YARP connections between the heterogeneous system components.

In the next section we present the mechanisms for automatically creating additional protoverbal behaviors in the module.

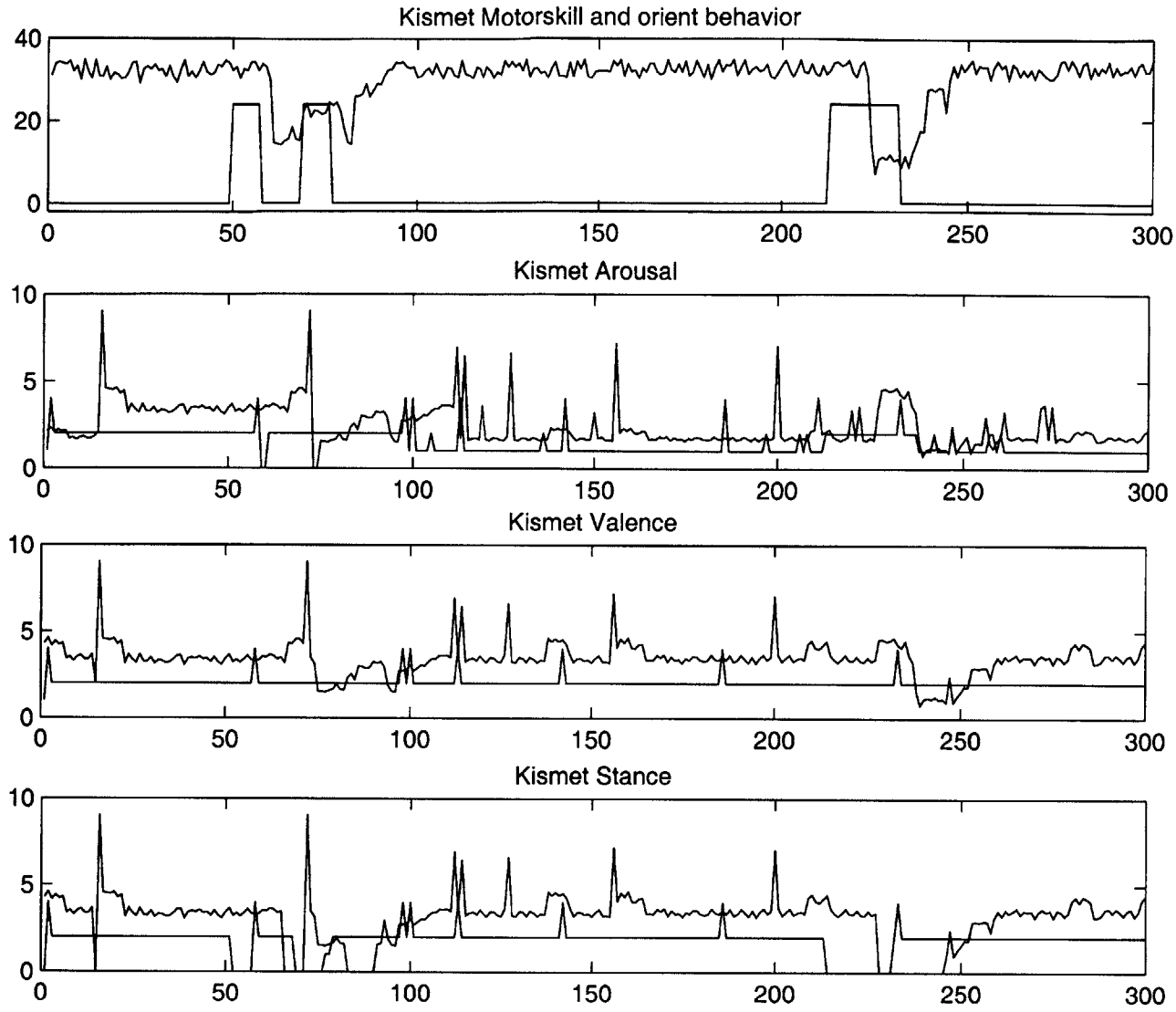



Figure 6-6: Some data from a label teaching scenario. Top graph shows the response of the `orient` behavior (the ragged line) superimposed with the robot's `Motorskill` variable. It follows a similar pattern after a delay. Bottom graphs show the arousal, valence, and stance releasers superimposed with the response of relevant `Receptors` of the `orient` behavior.

## 6.4 ConceptMap Methods

As was mentioned earlier, due to the nature of a Zac system, all functions must be called from within the state machine of a behavior which is active in the Mesh. The programmer does not have direct access to the main execution loop. Therefore, if we want to create and manage concepts at runtime, this must be through calls originating from another behavior. `ConceptMapManager` is the single behavior which acts as a very high-level manager of concepts. Described as such it sounds like precisely the homunculus which we are trying to avoid by creating this distributed architecture of behaviors. However, the functionality of `ConceptMapManager` is very tightly tied in with that of `HeardThis` and should perhaps be implemented as two extra states within that behavior. That would perhaps be more acceptable in terms of the spirit of the project. The current implementation has singled out `ConceptMapManager` mainly for the sake of readability.

### 6.4.1 Architecture and operation of the map

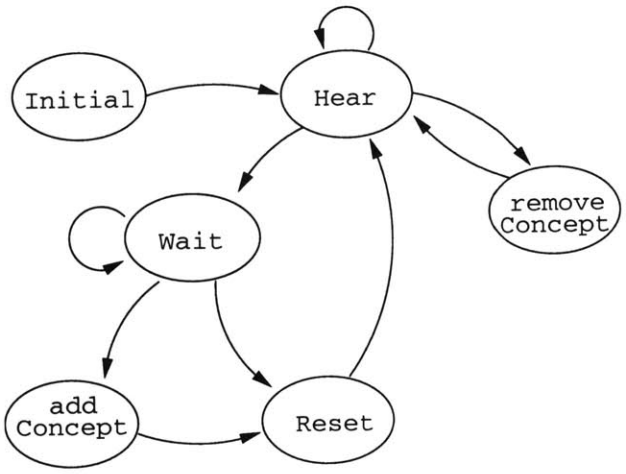
We define the `class ConceptMap`, extending the STL template class `map`, with methods which implement (and hide from the programmer) the BALZac initialization phase of `Concepts`:

```
class ConceptMap : public map <vocname, BALZACProcess_Concept *>
{
 public:
  //...Constructors, Destructor...

  void Insert (vocname name, BALZACProcess_Concept * ptrConcept);
  int DeleteByName (vocname name);
  void DeleteOne (map<vocname, BALZACProcess_Concept *>::iterator i);
  void DeleteN (map<vocname, BALZACProcess_Concept *>::iterator s,
                map<vocname, BALZACProcess_Concept *>::iterator f);

};
extern ConceptMap conceptMap;
```

Recall that BALZac `BEHAVIOR Name` statements get automatically translated to




```
BEHAVIOR ConceptMapManager {
  ConceptMap *ptrConceptMap;
  BALZACProcess_Concept *b;

  //Inputs a new concept request
  INPUT (LINK(int), iRequest, NULL);

  STARTUP {
    ptrConceptMap = &conceptMap;
  }

  //...State machine...
};
INSTANCE(ConceptMapManager);
```

Figure 6-7: The `ConceptMapManager` behavior and its state machine. `ptrConceptMap` is the behavior's handle on the only global object of type `class ConceptMap`.

declarations and definitions of `class BALZACProcess_Name : public BALZACProcess`, so that `ConceptMap` extends a map of pointers to behavior objects. The vocal labels of concepts serve also as the key in the `ConceptMap` and may be used to remove a concept from the Mesh by finding the vocal label key. The single object of that class, called unimaginatively `conceptMap`, will store pointers to all `Concept` behaviors, i.e., all non-grunting, non-babbling protoverbal behaviors. The single `ConceptMapManager` behavior has a pointer data member initialized to point at `conceptMap`. The state machine of `ConceptMapManager` operates as shown in figure 6-7. It initializes `conceptMap` to the small set of fixed concepts, which correspond to Kismet's existing behaviors. These are the same behaviors as described in the experiments in section 6.3. After initialization, `Hear` is the default looping state, which shadows the execution of the `HeardThis` state of the same name. When a new speech input arrives, the transition is taken to the `Wait` state, which increments a timer. If the `iRequest` is received in this state, it inhibits the default generation of a new `Concept`, making the behavior transition to the `Reset` state, which resets the timer and goes back to the `Hear` loop. Otherwise, if the timer runs out and there is no inhibitory signal at `iRequest`, the transition to the `addConcept` state is taken, where the `ConceptMap::Insert()` method is called. From there, the timer is reset unconditionally, and the behavior returns to the looping `Hear` state.

We have already seen in section 6.2 the methods by which individual concepts compute their activation and priority, match their vocal label to phonemic input, and update their parameters and their vocal label. Here we look over the "big picture", examining what happens as the entire system of protoverbal behaviors runs on the robot. The following section deals with the mechanism by which new instances of the `Concept` behavior are generated.

### 6.4.2 Addition of new concepts

If no inhibition signal is received in `iRequest`, by default `ConceptMapManager` creates a new entity for every time new speech input is received. This is done by calling the `ConceptMap::Insert()` method with the perceived speech string as the key. This function takes care of adding a new entry with that key to the map, as well as of the BALZac initialization routines for the newest addition. However, certain tasks still fall on the programmer. Input and output ports must be initialized and connected together, so they become part of the Mesh. The vector of `Receptors` must be populated with elements with specific boundary functions, which represent their "receptive fields". All this is accomplished from `ConceptMapManager`'s `addConcept` state.

Inputs and outputs are connected to their targets directly using the BALZac `DIRECT_CONNECTION` construct (where `ptrConcept` is a pointer of type `BALZACProcess_Concept *` which points to the newly created map entry:

```
DIRECT_CONNECTION <double> (ptrConcept, POINT(ptrConcept, iCheard),
                            cCheard, 1.0, 0);
DIRECT_CONNECTION <char *> (ptrConcept, POINT(ptrConcept, oVocal),
                            cSayThis, 1.0, 1);
```

Note that `oMatchResult` is not initialized at this point since which free-standing connection it attempts to overwrite is only determined during the execution of the particular `Concept` instance, and depends on its activation level.

When a new protoverbal behavior is automatically generated from `ConceptMapManager`, its vector of `Receptors` is initialized to include one for every kind of releaser that is

present in the system. All receptor gains are set to a default value. This exhaustive assignment will be remedied later as the gains are updated individually within each behavior.

## 6.5 Overview

Kismet's perceptual, motor, behavioral and motivational systems all produce releasers, which can be picked up by a protoverbal behavior's `Receptor` if it is within the boundaries, to which that `Receptor` is sensitive. Each behavior computes its activation based on the active `Receptors` and corresponding gains. If a behavior thus "recognizes" a state of the world, i.e., if its activation level reaches above a threshold, the behavior attempts to write its vocal label to the outgoing connection. Several behaviors may compete to establish the content of the connection; the one with the highest priority wins the competition, and its vocal label may then be spoken by the robot, depending on the value of the appropriate drive (see Chapter 5 for a discussion of drives and conditions on speech).

Independently, whenever something is heard, each behavior starts looking for a match, and `ConceptMapManager` attempts to create a new `Concept` instance with a vocal label that was most recently heard. Inhibitory signals prevent this addition in the cases where a good match is already available. Such a mechanism should be able to form rudimentary perceptual categories in scenarios of naming.

At the time of writing, we have not been able to pursue further experimentation with the perceptual space of the robot with a view to acquiring new categories as described above. The focus of the project thus far has been on establishing a framework in which meaningful protoverbal interaction can take place. In Chapter 7 we discuss the results obtained from the development of such a framework, the many limitations of the protolanguage module as it currently stands and directions for future research.

# Chapter 7

# Conclusions, Limitations, and Future Work

## 7.1 Discussion of Results

Currently, we have only been able to run a limited number of short experimental interactions with the robot to test the Protolanguage module. In particular, we have no interesting test results from the working of `ConceptMap` which would show that the robot has acquired a new concept from heard speech and perceptual features. A number of new `Concept` objects were created, but none resulted in a persistent entity with a definitely circumscribed response field.

Instead, the primary focus and objective of the project was to establish a framework in which concepts can be implemented as processes. Results from the development of the fixed protoverbal behaviors, presented in Chapter 5 and section 6.3 show that the system can engage in natural interaction by following simple update rules that govern the behavior of its components. However, the preliminary results in label acquisition have been disappointing due to many factors. One is the simplistic nature of the rules. Another one is the high level of noise in the workings of the robot, which, although undoubtedly contributing to the impression of naturalness and spontaneity on the robot's human users and teachers, is hard to deal with at the system level.

## 7.2 Limitations

In the protolinguistic system itself, there are currently problems in phoneme recognition, which lacks adequate accuracy, and has resulted in poor performance of label updates and proliferation of spurious "new concepts". A great limiting factor throughout the project is the lack of proper word segmentation from fluent speech. Matching strings with different windows is not a robust solution to this problem, which is very hard in the case where no words are known a priori. Even though we give an overview in chapter 2 of the kinds of supralinguistic statistical information that human infants make use of in word segmentation, we have not as of yet implemented any such extensive mechanism on the robot.

There are many limitations in the manner in which acquisition mechanisms are implemented. Simple update rules rely on the availability of very specific releasers. Thus far we also fail to use any exogenic reinforcement signals. Obvious candidate signals, to be used in future work are the outputs of the affective intent classifier developed in Breazeal & Aryananda (2000). We have also neglected to provide a solution to the issue of dying concept removal. With the algorithms as presented in this thesis, a great proliferation of behaviors which only become active very rarely could easily happen. Finally, the system as it stands now is not very tolerant to noise which abounds in the robot's sensorimotor mechanisms.

## 7.3 Future Work

Research in the immediate future will focus on developing the current system to the point where we can demonstrate interesting results in new concept acquisitoin. This would be a fuller system, where the aforementioned issues of reinforcement signals, learning models and algorithms, useless behavior management, and noise tolerance are addressed directly. On the speech recognition side of the problem, it would be interesting to pursue the problem of word segmentation from the natural cues available to infants. What makes this approach hard is that it seems contingent on extremely

accurate phoneme recognition, at which machines are still not very good.

The pragmatic approach opens new possibilities in machine language acquisition and use. There is a range of possible future research directions, the most important one being the transition to full linguistic ability from protolanguage. This would be characterized by an ability to recognize objects in a profound way, e.g., the ability to distinguish two objects in a scene as having distinct identities. That ability and understanding of spoken object labels are related and develop at the same time in human children (Xu & Carey 1995). It would also be characterized by the introduction of simple grammatical constructs and combinations of meanings.

Finally, future research will concentrate on learning new and appropriate verbal behaviors, and on a method for systematically evaluating the results that may be achieved. This may involve proper sociological experiments with groups of naive subjects interacting with the robot under separate conditions. In one condition, the robot would be running only the standard demonstration software, including turn-taking and conversational "babble" as its only vocal output. In the other condition, the robot would be running the protoverbal behavioral system and the learning algorithms presented here. We could assess the naturalness of interaction, as well as success on some cooperative task that may require meaningful communication. We may also assess the overall state of the robot's emotional system throughout the interaction.

Perhaps we can imagine the perfect demonstration scenario that would show the benefits of the pragmatic approach as something along the following lines. Kismet makes a vocalization that is interpreted as a request by the human teacher. The teacher therefore produces something, a toy, that she thinks is what the robot asked for. In a principled way the robot should display happiness if the world now contains the attributes of the behavior that satisfies the request. The robot would display sadness or frustration when the request is not satisfied by the toy, and repeat the vocalization until satisfied. Such a sophisticated interaction is unfortunately well beyond the scope of this thesis.

# Appendix A

# The Phonemes of American English

Table A.1: The Phonemes of American English: Vowels. Using their ARPA representation (also used in DecTalk4.1). Rows by place of articulation, and columns by manner of articulation.

| | High | Mid | Low |
|---|---|---|---|
| Front | iy ih | ey eh | |
| Central | ux | ah ax | ae |
| Back | uh uw | ow ao | aa |

Table A.2: The Phonemes of American English: Consonants. Using their ARPA representation (also used in DecTalk4.1). Rows by manner of articulation, and columns by place of articulation.

| | Labial | Dental | Alveolar | Palatal | Velar |
|---|---|---|---|---|---|
| Stop (Occlusive) | p b | | t d | | k g |
| Fricative | f v | th dh | s z | sh zh | |
| Nasal | m | | n | | ng |
| Semivowels | y w l r | | | | |
| Affricates | jh ch | | | | |
| Others | hh hv dx nx q | | | | |

# Appendix B

# Syntax of BALZac Script

| | | |
|---|---|---|
| **<Program>** | :== | **{<BALZac-unit> \| <c-unit>}** |
| **<BALZac-unit>** | :== | **<Lateral-unit> \|<Declare> \| <Template> \| <BALZac-Behavior> \| <Instance> \| <Instance-Init> \| <Connection-vector>** |
| <Lateral-unit> | :== | (<Behavior> \| <Connection> \| <Object>)";" |
| <Behavior> | :== | "BEHAVIOR" <Identifier> "{"<Behavior-body>"}" |
| <Behavior-body> | :== | {<Behavior-unit>} [<State-machine>] |
| <Behavior-unit> | :== | <c-unit> \| <Behavior-connection> |
| <Behavior-connection> | :== | (<Port> \| <Port-status> \| **<Port-vector>**)";" |
| **<Declare>** | :== | **"DECLARE" "{" {<c-unit>} "}" ";"** |
| **<Template>** | :== | **"TEMPLATE" "<" "class" <Identifier> ">" {<Behavior> \| <BALZac-Behavior>} ";"** |
| **<BALZac-Behavior>** | :== | **<Behavior> "INSTANCE"** |
| **<Instance>** | :== | **"INSTANCE" ["<" <Identifier> ">"] "(" <c-term> "," <c-term> ")" ";"** |
| **<Instance-Init>** | :== | **"INSTANCE_INIT" "(" <Identifier> "," <c-term> "," <c-term> "," <c-term> "," <c-term> ")" ";"** |
| <Port> | :== | ("INPUT" \| "OUTPUT") "(" <c-term> "," <c-term> "," <c-term> ")" |
| <Port-status> | :== | ("INPUT_STATUS" \| "OUTPUT_STATUS") "(" <c-term> "," <c-term> "," <c-term> ")" |
| **<Port-vector>** | :== | **("INPUTS" \| "OUTPUTS") "(" <c-term> "," <c-term> ")"** |
| **<Connection-vector>** | :== | **"CONNECTIONS" "(" <c-term> "," <c-term> "," <c-term> "," <c-term> "," <c-term> ")"** |
| <State-machine> | :== | <State> {<State>} |
| <State> | :== | "@" <Identifier> {**<B-unit>** \| <Next>} |
| **<B-unit>** | :== | **<c-term> \| <Add-unit> \| <Del-unit> \| <Connect-unit>** |
| **<Add-unit>** | :== | **("ADD_INPUT" \| "ADD_OUTPUT") "(" <Identifier> ["," <c-term>] ")" ";"** |
| **<Del-unit>** | :== | **"REMOVE" "(" <c-term> ["," <c-term>] ")" ";"** |
| **<Connect-unit>** | :== | **"DIRECT_CONNECTION" "(" <c-term> "," <c-term> "," <Point> "," <Identifier> "," <c-term> "," <c-term> ")" ";"** |
| <Next> | :== | ["DEFAULT"] "NEXT" [<Identifier>] ";" |
| <Object> | :== | "OBJECT" "(" <c-term> {"," <c-term>} ")" |
| <Connection> | :== | ("CONNECT" \| "CONNECT_STATUS") "(" <Link> "," <Identifier> "," <Point> "," <c-term> ")" |
| <Link> | :== | <c-term> \| ("LINK" "(" <c-term> ")") |
| <Point> | :== | "POINT" "(" <Identifier> "," <Identifier> ")" |

Table B.1: BALZac Script syntax in EBNF notation. Changes from the original Zac are shown in bold letters.

# Bibliography


Aryananda, L. (2001), Online and Unsupervised Face Recognition for Humanoid Robot: Toward Relationship with People, *in* 'Proceedings of the IEEE-RAS International Conference on Humanoid Robots'.

Aslin, R., Woodward, J., LaMendola, N. & Bever, T. (1996), Models of Word Segmentation in Fluent Maternal Speech to Infants, *in* J. Morgan & K. Demuth, eds, 'Signal to Syntax: Bootstrapping From Speech to Grammar in Early Acquisition', Lawrence Erlbaum Associates: Mahwah, NJ.

Bard, E. & Anderson, A. (1994), 'The unintelligibility of speech to children: effects of referent availability', *Journal of Child Language* **21**, 623–648.

Bazzi, I. & Glass, J. (2000), Modeling Out-of-Vocabulary Words for Robust Speech Recognition, *in* 'Proc. 6th International Conference on Spoken Language Processing', Beijing, China.

Bloom, P. (2000), *How Children Learn the Meaning of Words*, Cambridge: MIT Press.

Blumberg, B. (1994), Action-Selection in Hamsterdam: Lessons from Ethology, *in* 'From Animals To Animats'.

Boliek, C., Hixon, T., Watson, P. & Morgan, W. (1996), 'Vocalization and breathing during the first year of life', *Journal of Voice* **10**(1), 1–22.

Boysson-Bardies (1999), *How Language Comes to Children: From Birth to Two Years*, MIT Press, Cambridge MA.

Breazeal, C. (1998), A Motivational System for Regulating Human-Robot Interaction, *in* 'Proceedings of the American Association of Artificial Intelligence (AAAI-98)'.

Breazeal, C. (2000), Sociable Machines: Expressive Social Exchange Between Humans and Robots, PhD thesis, MIT Department of Electrical Engineering and Computer Science.

Breazeal, C. & Aryananda, L. (2000), 'Recognition of Affective Communicative Intent in Robot-Directed Speech', *Proceedings of Humanoids 2000.*

Breazeal, C. & Scassellati, B. (1999), A context-dependent attention system for a social robot, *in* '1999 International Joint Conference on Artificial Intelligence'.

Breazeal, C. & Scassellati, B. (2000), 'Infant-like Social Interactions between a Robot and a Human Caretaker', *Adaptive Behavior.*

Breazeal, C., Edsinger, A., Fitzpatrick, P., Scassellati, B. & Varchavskaia, P. (2000), 'Social Constraints on Animate Vision', *IEEE Intelligent Systems.*

Brent, M. & Siskind, J. (2001), 'The role of exposure to isolated words in early vocabulary development', *Cognition* **81**, B33–B44.

Brock, D. P. (2001), A Language Use Approach to Human-Computer Interaction, Technical report, Navy Center for Applied Research in Artificial Intelligence.

Brooks, R. A. (1986), 'A Robust Layered Control System for a Mobile Robot', *IEEE Journal of Robotics and Automation RA-2.*

Brooks, R. A. (1990), The Behavior Language User's Guide, Memo 1227, Technical report, MIT, Artificial Intelligence Laboratory.

Brooks, R. A., Breazeal, C., Marjanovic, M., Scassellati, B. & Williamson, M. M. (1999), The Cog Project: Building A Humanoid Robot, Springer-Verlag Lecture Notes in Computer Science.

Burnham, D., Francis, E., Kitamura, C., Vollmer-Conna, U., Averkiou, V., Olley, A. & Paterson, C. (1998), Are You My Little Pussy-Cat? Acoustic, Phonetic And Affective Qualities Of Infant- And Pet-Directed Speech, *in* 'Proc. 5th International Conference on Spoken Language Processing', Vol. 2, pp. 453–456.

Cassell, J., Bickmore, T., Billingshurst, M., Campbell, L., Chang, K., Villjálmsson, H. & Yan, H. (1999), Embodiment in conversational interfaces: Rea, *in* 'Proceedings of CHI '99', pp. 520–527.

Cassell, J., Sullivan, J., Prevost, S. & Churchill, E., eds (2000), *Embodied Conversational Agents*, MIT Press.

Chomsky, N. (1986), *Knowledge of Language: Its Nature, Origin, and Use*, New York: Praeger.

Cole, R., Mariani, J., Uszkoreit, H., Zaenen, A. & Zue, V., eds (1995), *Survey of the State of the Art in Human Language Technology.*

Fitzpatrick, P. (1997), A novel behaviour-based robot architecture and its application to building an autonomous robot sentry, Master's thesis, University of Limerick, Ireland.

Fitzpatrick, P. & Metta, G. (forthcoming), YARP, Technical report, MIT, Artificial Intelligence Laboratory.

Gat, E. (1992), Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling Real-World Mobile Robots, *in* 'Proceedings of AAAI-92', San Jose, CA, pp. 809–815.

Gesture & Narrative Language Group at the Media Lab, M. (2001), http://gn.www.media.mit.edu/groups/gn/projects/kiosk/index.html.

Glass, J., Chang, J. & McCandless, M. (1996), A probabilistic framework for feature-based speech recognition, *in* 'Proc. International Conference on Spoken Language Processing', pp. 2277–2280.

Gorin, A., Petrovksa-Delacrtaz, D., Riccardi, G. & Wright, J. (1999), Learning Spoken Language without Transcriptions, *in* 'Proc. IEEE Automatic Speech Recognition and Understanding Workshop', Colorado.

Halliday, M. (1975), *Learning How To Mean: Explorations in the Development of Language*, Cambridge: MIT Press.

Harnad, S. (1990), 'The Symbol Grounding Problem', *Physica D* **42**, 335–346.

Hauser, M. D. (1996), *The Evolution of Communication*, MIT Press, chapter 5: Ontogenetic Design and Communication.

Hazen, T. & Bazzi, I. (2001), A Comparison and Combination of Methods for OOV Word Detection and Word Confidence Scoring, *in* 'Proc. International Conference on Acoustics', Salt Lake City, Utah.

Hirschberg, J., Litman, D. & Swerts, M. (1999), Prosodic Cues to Recognition Errors, *in* 'ASRU'.

Holzman, M. (1997), *The Language of Children*, 2nd ed. edn, Blackwell Publishers.

Jusczyk, P. (1997), *The Discovery of Spoken Language*, Cambridge: MIT Press.

Jusczyk, P. & Aslin, R. (1995), 'Infants' Detection of the Sound Patterns of Words in Fluent Speech', *Cognitive Psychology* **29**, 1–23.

Jusczyk, P., Friederici, A., Wessells, J., Svenkerud, V. & Jusczyk, A. (1993), 'Infants' sensitivity to the sound patterns of native language words', *Journal of Memory and Language* **32**, 402–420.

Klopfer & Hailman (1965), *An Introduction to Animal Behavior, Ethology's First Century*, chapter The Study of Instinct.

Malcolm Staney, G. M. (1998), Baby Ears: A Recognition System For Affective Vocalizations, *in* 'Proceedings of the 1998 International Conference on Acoustics, Speech, and Signal Processing (ICASSP)'.

Matsusaka, Y. & Kobayashi, T. (1999), Human Interface of Humanoid Robot Realizing Group Communication in Real Space, *in* 'Proc. Second International Symposium on Humanoid Robots', pp. 188–193.

McCune, L., Vihman, M., Roug-Hellichius, L., Delery, D. & Gogate, L. (1996), 'Grunt communication in human infants (Homo sapiens)', *Journal of Computational Psychology* **110**(1), 27–36.

Oates, T., Eyeler-Walker, Z. & Cohen, P. (2000*a*), 'Toward Natural Language Interfaces for Robotic Agents: Grounding Linguistic Meaning in Sensors', *Proceedings of the 4th International Conference on Autonomous Agents* pp. 227–228.

Oates, T., Eyler-Walker, Z. & Cohen, P. (2000*b*), Toward Natural Language Interfaces for Robotic Agents: Grounding Linguistic Meaning in Sensors, *in* 'Proceedings of the 4th International Conference on Autonomous Agents', pp. 227–228.

Pepperberg, I. M. (1990), 'Referential mapping: A technique for attaching functional significance to the innovative utterances of an African Grey parrot', *Applied Psycholinguistics* **11**, 23–44.

Pepperberg, I. M. (1997), *Social Influences on Vocal Development*, chapter 9: Social influences on the acquisition of human-based codes in parrots and nonhuman primates.

Pepperberg, I. M. (n.d.), Categorical Class Formation by an African Grey Parrot, *in* Zentall & Smeets, eds, 'Stimulus Class Formation in Humans and Animals', North Holland Publishers.

Pinker, S. (1999), *Words and Rules: The Ingredients of Language*, Basic Books.

Ristad, E. (1993), *The Language Complexity Game*, Cambridge: MIT Press.

Roug-Hellichius, L. (1998), Communicative Grunts: An Adaptive Phonetic Phenomenon?, *in* 'Proceedings of the Eleventh Swedish Phonetics Conference', Stockholm.

Roy, D. K. (1999), Learning Words from Sights and Sounds: A Computational Model, PhD thesis, MIT.

Roy, D. K. (2000), Integration of speech and vision using mutual information, *in* 'Proceedings of the International Conference on Speech and Signal Processing', Istanbul, Turkey.

Seidenberg, M. S. (1997), 'Language Acquisition and Use: Learning and Applying Probabilistic Constraints', *Science* **275**, 1599–1603.

Shafer, V. L. & Shucard, D. W. (1995), Is the Right Hemisphere of the Brain Functionally Specialized to Process Prosodic Information from Birth?, *in* D. MacLaughlin & S. McEwen, eds, 'Proceedings of the Boston University Conference on Language Development', Vol. 19, pp. 563–574.

Siskind, J. M. (1995), Robust Lexical Acquisition Despite Extremely Noisy Input, *in* D. MacLaughlin & S. McEwen, eds, 'Proceedings of the Boston University Conference on Language Development', Vol. 19, pp. 587–598.

Song-Yee Yoon, Bruce M. Blumberg, G. E. S. (n.d.), 'Motivation Driven Learning for Interactive Synthetic Characters'.

Speidel, G. E. & Nelson, K. E., eds (1988?), *The Many Faces of Imitation in Language Learning*, Springer-Verlag.

Steels, L. (1999), *The Talking Heads Experiment, vol.I Words and Meanings*, Special pre-edition for LABORATORIUM, Antwerp.

Steels, L. & Kaplan, F. (2001), Bootstrapping Grounded Word Semantics, *in* T. Briscoe, ed., 'Linguistic evolution through language acquisition: Formal and Computational Models', Cambridge University Press.

Tincoff, R. (2001), 'Infants' Attention to Speech and Nonspeech Vocalizations and Its Relation to Word Learning', Poster presented at the conference of the Society for Research in Child Development.

Trevarthen, C. (1979), Communication and cooperation in early infancy: a description of primary intersubjectivity, *in* M. Bullowa, ed., 'Before Speech: The beginning of interpersonal communication', Cambridge University Press.

Varchavskaia, P., Fitzpatrick, P. & Breazeal, C. (2001), Characterizing and Processing Robot-Directed Speech, *in* 'Proceedings of the IEEE-RAS International Conference on Humanoid Robots'.

Waxman, S. R. (1995), Characteristics of word learners at 12 and 30 months: Early emergence and modification of the noun-category linkage, *in* D. MacLaughlin & S. McEwen, eds, 'Proceedings of the Boston University Conference on Language Development', Vol. 19, pp. 667–678.

Werker, J., Lloyd, V., Pegg, J. & Polka, L. (1996), Putting the Baby in the Bootstraps: Toward a More Complete Understanding of the Role of the Input in Infant Speech Processing, *in* J. Morgan & K. Demuth, eds, 'Signal to Syntax: Bootstrapping From Speech to Grammar in Early Acquisition', Lawrence Erlbaum Associates: Mahwah, NJ, pp. 427–447.

Wray, A. (2000), A protolanguage with no declaratives and no names, *in* 'Proceedings of the 3rd Conference on the Evolution of Language'.

Xu, F. & Carey, S. (1995), Do Children's First Object Kind Names Map onto Adult-like Conceptual Representations?, *in* D. MacLaughlin & S. McEwen, eds, 'Proceedigns of the Boston University Conference on Language Development', Vol. 19, pp. 679–688.

Zlatev, J. (1999), 'The Epigenesis of Meaning in Human Beings, and Possibly in Robots', *Lund University Cognitive Studies.*

Zue, V., Glass, J., Plifroni, J., Pao, C. & Hazen, T. (2000), 'Jupiter: A Telephone-Based Conversation Interface for Weather Information', *IEEE Transactions on Speech and Audio Processing* **8**, 100–112.