CLAVIUS: UNDERSTANDING LANGUAGE UNDERSTANDING IN MULTIMODAL INTERACTION

Frank Rudzicz

Department of Electrical and Computer Engineering
McGill University, Montréal

September 2006

A Thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfilment of the requirements for the degree of
Master of Engineering

Abstract

Natural communication between humans is not limited to speech, but often requires simultaneous coordination of multiple streams of information - especially hand gestures - to complement or supplement understanding. This thesis describes a software architecture, called CLAVIUSwhose purpose is to generically interpret multiple modes of input as singular semantic utterances through a modular programming interface that supports various sensing technologies. This interpretation is accomplished through a new multi-threaded parsing algorithm that co-ordinates top-down and bottom-up methods asynchronously on graph-based unification grammars. The interpretation process follows a best-first approach where partial parses are evaluated by a combination of scoring metrics, related to such criteria as information content, grammatical structure and language models. Furthermore, CLAVIUSrelaxes two traditional constraints in conventional parsing - namely, it abandons forced relative ordering of right-hand constituents in grammar rules, and it allows parses to be expanded with null constituents.

The effects of this parsing methodology, and of the scoring criteria it employs, are analyzed within the context of experiments and data collection on a small group of users. Both CLAVIUSand its component modules are trained on this data, and results show improvements in performance accuracy, and the overcoming of several difficulties in other multimodal frameworks. General discussion as to the linguistic behaviour of speakers in a multimodal context are also described.

Résumé

La communication naturelle entre les humains n'est pas limitée à la parole, mais s'exige souvent de la coordination simultanée de plusieurs sources d'information - particulièrement les gestes manuelles - lorsqu'ils complétent ou ajoutent à la comprehension. Cette thèse décrit une architecture de logiciel, appelée CLAVIUS, dont le but est l'interprétation générique de plusieurs modes d'entrée comme des énoncés ayant des significations uniques par une interface de programmation modulaire qui soutient de diverses technologies de sensation. Cette interprétation est accomplie par un nouvel algorithme multi-fileté d'analyse grammaticale qui coordonne des méthodes asynchronés sur des grammaires d'unification fondées sur une représentation graphique. La procédure d'interprétation suit une approche de meilleur-en-premièr donc les analyses grammaticales sont évalués par une métrique combinatoire de marquage, liée à des critères tels que le contenu de l'information, la structure grammaticale et les modèles de la langue. De plus, CLAVIUSdétend deux contraintes traditionnelles dans l'analyse grammaticale conventionnelle - notamment, il abandonne que l'ordre des constituants mains-droits soit obligatoire dans les règles de grammaire, et il laisse que l'analyse peut se composer des constituants nuls.

Les effets de cette méthodologie d'analyse, et des critères de marquage qu'il utilise, sont analysés dans le contexte des évaluations et d'une collecte de données sur un group d'usagers. CLAVIUSet ses modules compositionnels sont entraînés sur ces données, et les résultats montrent des améliorations dans l'exactitude d'exécution, et une surmontage des plusieurs difficultés dans les autres systèmes semblables. La discussion générale quant au comportement linguistique des usagers dans ce contexte multimodale est aussi décrite.

ACKNOWLEDGEMENTS

Working on CLAVIUS has been all of these things at one time or another, among other things, often in parallel: invigorating, intriguing, exhausting, frustrating, gratifying and of course, fun. This section acknowledges those who have specifically had a direct influence in this work.

I am indebted to Professor Jeremy Cooperstock for providing me with the accommodating environment in which this research took place. Jeremy encouraged me to forge my own research directions, and emphasized the importance of academic rigour and publication.

The members of the Shared Reality Lab at McGill University have all at one point or another contributed in some way, either through collaboration or in casual discussion. I especially thank the other members of the Modeller's Apprentice team - François Rioux and Michael Wozniewski, who constituted one of the most productive and fun teams with whom I've ever worked. Part of §3.5.6 of this thesis is adapted from a prior publication of ours[**15**]. I also thank Stephen Spackman for early discussions during the inception of this project, and his assistance regarding relevant literature.

My family has been extremely supportive throughout my entire life and have contributed their patience, love, and respect to *that* ongoing project. They deserve my utmost thanks, and are really a whole lot of fun.

This thesis is dedicated to Melissa de Souza, my beautiful and wonderful fiancée. She is my ultimate source of inspiration and joy and has been the most motivating and encouraging individual in everything I've accomplished. This thesis marks one more day for me, my lovely ;).

This research was funded by la bourse de maitrisse of the fonds québécois de la recherche sur la nature et les technologies.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF NOTATIONS AND ABBREVIATIONS

MMI	**abbrev.** Multimodal Interaction	1
HCI	**abbrev.** Human-Computer Interaction	1
NLP	**abbrev.** Natural Language Processing	1
ASR	**abbrev.** Automatic Speech Recognition	4
RER	**abbrev.** Relative Error Reduction	5
HMM	**abbrev.** Hidden Markov Model	5
TDAG	abbrev. Typed Directed Acyclic Graph	10
FS	abbrev. Feature Structure.	10
Ψ_A	a particular DAG or feature structure	10
δ_i	an arc in some graph Ψ	10
$\|\Psi_A\|$	size of G_A, defined as the number of arcs in G_A	10
Φ_f	a feature in the feature set FEAT.	10
$\sqsubseteq$	the subsumption relation, defined between two feature structures	11
$\sqcup$	the Unification operator, with feature structure arguments	12
Γ	Grammar	15
Γ_i	the i^{th} rule in grammar Γ	15
LHS,RHS	Left-Hand Side, Right-Hand Side of a grammar rule $Gamma_i$, respectively	24
$\Xi^{[a]}$	Subspace a, where a is G(eneralizer), S(pecifier, active) or s(pecifier, inactive)	34
σ_i	Search node	35
γ_i	the set of constituents on the RHS of Γ_i	42
$\gamma_i^{(j)}$	the j^{th} RHS constituent of Γ_i	42
κ_i	The i^{th} scoring module used in computation of score, $Score(\Psi)$	46
ω_i	Weight associated with module κ_i in computation of score, $Score(\Psi)$.	46
Δ	A set of training data. Datum Δ_i encapsulates user input and transcription	64

CHAPTER 1

Introduction

Whether describing a scene, or objects and actions in space, humans make frequent use of gestures not only to supplement, but also to complement their speech. For instance, when providing directions, a speaker will often depict significant landmarks and the spatial relationships between them with their hands – the trajectory and distance between a fork in the road, for example, and a building.

The key concept is that the human language is not a decoupled process, but involves the simultaneous coordination of multiple streams of information – not just speech. This will be expanded upon shortly.

Multimodal interaction (MMI) represents a paradigm shift in engineering whereby traditional user interfaces are replaced by more intelligent software capable of incorporating these multiple components of natural user communication simultaneously in order to infer their underlying semantic meaning. Through the combined processing of multiple input **modes**, of which speech and gesture are just two examples, these systems have made significant breakthroughs in the fields of Natural Language Processing (NLP) and Human-Computer Interaction (HCI) .

In this thesis a new software framework called Clavius is presented that combines an extensible architecture for MMI with a new set of algorithms that expand on modern NLP techniques to handle some peculiarities of multimodal language. As a consequence of a modular framework, various divergent approaches to this task can be explored and incorporated within a single framework.

The following section introduces the field of multimodal interaction research, briefly overviewing popular or interesting approaches thereof, as well as current benefits and challenges (§1.1). Relevant psychological theory (§2.3) is then briefly touched upon before describing the goals and contributions of this thesis (§1.2).

1.1. Multimodal Human-Computer Interaction

A multimodal system is any one that allows users to express themselves using more than one channel of communication. Examples of the more prevalent of these include:

speech: : natural human speech recorded by microphone.

gesture: : gesticulation of the hands, as recorded by video or specialized equipment (ex. sensor-laden gloves) (see §2.3).

gaze/face: : the direction vector of the user's eyes (and therefore attention), and the shape of one's lips (indicative of the articulation of their speech). These are typically recorded by video-camera fixed on the user's face.

pen: : use of a small stylus on a 2D reactive surface, such as a PDA or tablet PC, often used for handwriting recognition (see §1.1.2.1).

typing: : traditional keyboard entry.

Multimodal interfaces have both taken advantage of and advanced the development of new hardware technologies. Current directions in MMI include expansion into more exotic hardware interfaces that incorporate haptic ("touch") information[**119**] [**64**], body position [**4**], and even directly measure brain activity [**103**].

1.1.1. Early History. The goal of supporting more expressive, efficient and flexible means of interacting with machines represents an ongoing effort in Human-Computer Interaction. Already, multimodal interfaces have allowed for increasingly challenging applications that are usable by more people in a wide range of types of physical environments, and it is expected that eventually this 'next step' towards human-like sensory perception in machines will allow our technological counterparts to interact with humans naturally and transparently in everyday environments.

Most research in multimodal human-computer interaction traces its origins to work done by Richard Bolt at the Massachusetts Institute of Technology (MIT) in the early 1980s [**14**]. That worked involved staged interactions within a "media room" in which a seated user interacted with abstract objects on a wall-sized display using rudimentary speech commands augmented with gesture, such as *"create a blue square <u>there</u>"*, and *"put <u>that</u> <u>there</u>"*. In these examples, arm gestures provide the computer with information otherwise absent in the referring words, namely co-ordinate locations, and the identity of objects. A technological breakthrough at the time, it demonstrated two key facets of multimodal interaction:

(i) that spontaneous multimodal expression provides an especially versatile and natural interface to the user, and
(ii) that pronouns used as verbal arguments in HCI achieve the same usefulness as in ordinary discourse by being pronounced in the presence of a deictic reference to an object that functionally defines their meanings.

Since that innovative early step, research in MMI has been divided primarily between exploration of the methods used to integrate information from the various modalities, and the empirical study of users in multimodal environments. For the most part, the abilities users possess in these environments have not changed drastically since these early systems, which often concentrated solely on direct manipulation of 2-dimensional maps or charts [**20**] [**21**].

Moreover, despite more than two decades of pursuant research, no dominant paradigm has yet emerged for the generic interpretation of multimodal interaction. A commonality amongst many divergent approaches, however, is an apparent reluctance to treat multimodal language as a unique linguistic phenomenon, which results in computational models that are retrofitted to older approaches, as examples throughout this thesis demonstrate.

1.1.2. Examples of MMI in Practise. A wide diversification and specialization in hardware interfaces allows for various permutations of modalities, including speech with pen input, speech with lip movements, speech with manual gesturing, and gaze tracking with manual input. They also allow for varying physical environments in which these can be deployed - from portable devices to large or immersive spaces. It is important to note that MMI systems cover a wide range of complexities, and that this may limit comparability to some degree.

1.1.2.1. *Mobile MMI.* As noted earlier, stylus input is indicative of multimodal mobile devices such as the PDA/Palm pilot, tablet PC, or cell phone, as shown in Figure 1.1. The small visual displays and cumbersome keypad input on these devices present limitations that cannot be overcome by voice browsers alone [**30**]. Combining speech with pen or keypad input, however, has led to increased access by a device to different types of programs and enhanced experiences in such mobile applications as stock trading or appointment scheduling. For example, stocks may be selected from a graphical list with the stylus, while commands applied to these selections may be spoken (eg. "sell"). Interaction with the stylus is then still possible while speech or audio feedback is being conveyed by the device – creating a degree of concurrency not possible in unimodal interaction.

Studies in such interaction scenarios indicate that flexible MMI allows users to manage changes in their cognitive load as the task complexity increases [**36**].

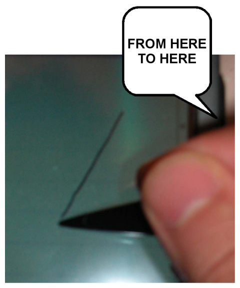


Figure 1.1: Example of mobile technology with stylus and speech interface.

1.1.2.2. *Large Screen MMI.* Interacting with large screens with arm gestures and speech has been employed chiefly to map-based applications in areas such as military intervention [**117**], crisis management [**100**], and weather narration [**57**]. This sort of arrangement typically involves a user standing directly before a large screen where arm gestures indicate virtual objects or locations on the screen, and speech conveys commands, as exemplified in Figure 1.2. Arms may be tracked by any combination of video-based tracking, infra-red (IR) sensors or wearable gyroscopic devices.

Unlike mobile MMI, large-screen displays are not encumbered by the same physical limitations. In this case, MMI is often used to *augment* the usual interaction paradigm by giving the user more locative freedom. Specifically, users are not necessarily constrained to sit in a particular spot with their hands resting on a keyboard but may move around the environment. This is made possible by distributing sensing devices, such as the microphones that record speech, around the environment – or at least by making these devices wireless, offering a degree of untetheredness.

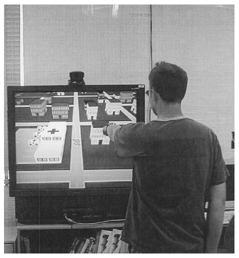


Figure 1.2: Example of large screen multimodal interface (from Sharma et al. [**100**]).

1.1.3. Applicability of MMI. A chief benefit of co-ordinating multiple modalities is the increased usability gained from compensating for any weakness in one modality with the strengths in others. For example, popular mobile phone manufacturers, such as Motorola, have included automatic speech recognition (ASR) software since the late 1990s to circumvent the difficulty of text input on a small keypad [**41**], or navigating through menus on a small screen [**46**]. There is also widespread evidence that suggests a general subjective preference of users to interact multimodally. For example, in separate studies of map-based tasks, researchers found that all subjects spontaneously interacted multimodally, and that between 70% [**28**] [**38**], and 95% [**79**] of those subjects preferred multimodal interaction to its unimodal counterpart.

Divergent approaches to recognition in MMI are discussed further in §2.2.

1.1.3.1. *Accessibility.* Multimodal user interfaces also have implications for accessibility, in particular for physically impaired individuals who may have only limited availability of their voice, hearing, sight, or motor control. Offering redundant pathways to information or control in software is also of importance to “situationally impaired” individuals. For example, voice entry of one’s credit card information to a telephony system may be desirable in the home, but not in a public environment. Speech commands also become relevant to motorists who need to look up and dial a phone number while their hands and vision need to be engaged elsewhere. This same type of multimodal exploitation is found in operating rooms, where members of the surgical team can access patient information verbally, and receive this information aurally and visually to maximize comprehension in an aseptic environment.

1.1.3.2. *Accuracy & Biometrics.* More technically, integrating co-related information from concurrent cross-modal information streams provides a benefit in recognition accuracy, specifically in MMI systems that perform this integration early in the process. For example, if an MMI interpreter has evidence that a user is pointing towards an image of a house, this information can be used to suggest that an ambiguous spoken utterance is “*this house*” rather than “*this horse*”. The error suppression achievable with a multimodal system, as compared with a unimodal one, can be in excess of 40% Relative Error Reduction (RER) [**78**].

Furthermore, in the sub-field of MMI concerned exclusively with the recognition process, information from modes extraneous to the interaction are used to disambiguate or more accurately recognize unimodal utterances. For example, recording a speaker’s lips with video, and associating measurements of quadrature values in the video’s optical flow[1] with acoustic features in an associated trained Hidden Markov Model (HMM) framework have demonstrated improvements in accuracy in speech recognition by at least 4% at Signal-to-Noise Ratio (SNR) = 5 dB and 13% at SNR = 10 dB [**105**] [**106**] [**71**] for connected digit grammars, given baseline audio-only accuracy of 41% and 58%, respectively. This differs from MMI in that the interaction mechanism is unimodal (eg. speech-only).

Combining unique properties of speech that can help to identify users with information from video for face recognition or lip-reading has also proven to be a useful mechanism for advanced security via biometrics [**29**].

1.1.3.3. *Complications.* There are, however, complications in unifying various methods of interaction, ranging from low-level signal processing to high-level semantic analysis. New modes added to a system have been shown to have a multiplicative effect on such things as ambiguity, computational complexity, and representation [**8**] [**22**]. Also, since MMI is a relatively young field of research, there does not yet exist a small de facto set of technologies or methodologies that can conclusively be shown to dominate all others.

One of the emerging themes in multimodal research involves the difficulty of re-applying techniques across application domains of potentially varying complexity. For example, multimodal discourse (possibly with some virtual avatar) [**115**] might require some degree of integrated planning, whereas a map-based querying will not [**78**]. This disparity in task complexity can be seen as a limiting factor in the generalization of multimodal techniques to new tasks, although distributed component models such as Multiplatform

[1]The optical flow is defined as the distribution of apparent velocities in the movement of brightness patterns in an image.

[**39**] based on message passing between processes have recently been gaining in popularity to deal with this dilemma [**18**].

These issues, among others, are discussed throughout this thesis.

1.2. Thesis Objectives and Contributions

Clavius is a generic interpretation layer between users and some arbitrary software that they use. It deciphers what a user says and does, for example, and passes this information along to the appropriate application. It provides a multimodal language-based integration engine within a generic framework that produces a structured semantic representation of user communication from raw data. Although its purpose is to recognize multimodal phrases, it is also amenable to unimodal user input, such as standard text- or speech-based processing.

This software was designed to meet the following broad goals:

Flexibility: The core recognition system must minimize the number of assumptions as to the nature of the application, or class of user input with which it will eventually be used.

Scalability: Augmentations to the abilities of the system must be made easily. This includes, for example, the ability to handle new types of input.

Efficiency: Chapter 2 will show that the speed of recognition has been an ongoing problem in MMI research. Clavius must address this issue meaningfully.

Structured semantics: Complex analysis of the meaning of phrases and sentences should be possible in order to draw inferences and constrain the process, as much as possible.

1.2.1. Contributions. The general approach in Clavius uses graph-based data structures (§2.1) that encode sensory information, syntactic structure and semantic knowledge within descriptions of a user's interaction with an application. The actual method used is a novel 'parsing' algorithm – the GS Algorithm – that integrates various approaches to multimodal interaction by means of a modular architecture. The GS Algorithm, and the architecture built to accomodate it form the two main innovations of this thesis. Some specific innovative aspects that overcome or address shortcomings in other work (see Chapter 2) include:

Mutual information: : Input from one mode can be used to inform interpretation in other modes. For example, if a word is spoken with a simultaneous gesture to an area, the nature of that gesture can be used to infer what the speaker said.

Distribution and collaboration: : Distributing the responsibility for different tasks across multiple machines or processing threads allows for a dynamic, robust, and efficient system.

Unified semantics: : Multimodal grammars allow a degree of semantic or world knowledge to be incorporated in the combination and interpretation of partial results.

Real-time recognition: : Interpretation takes place *continuously* as soon as new input is introduced, instead of waiting until sensory activity has ended (such as after a long pause). This means that the beginning of a sentence may be interpreted while the user is still uttering it. This is sometimes referred to as *interactive* parsing [**108**].

Interruptible Recognition: : At any time, Clavius may interrupt itself in order to return its best possible hypothesis, but may decide to deliberate further it believes that by doing so it can improve the correctness of its estimate.

Heuristics: : Since the problem of MMI is still in its infancy, and has many disparate approaches (see §2.2), a simple *scoring* paradigm that can unify various solution concepts (over various domains) is employed.

Automated Learning: : Clavius is highly customizable 'by hand', but is also amenable to training itself off-line according to interaction data.

In Clavius, unlike many speech recognizers [**65**], the goal is not constrained only to complete and fully grammatical sentences (for formal approximations to natural languages). In fact, Clavius can also accept 'sentences' that consist only of unexpected sentence fragments, but which still have internal structure and meaning.

1.3. Organization

The rest of this thesis is organized into four chapters. Chapter 2 defines some necessary linguistic and mathematical background forming the foundation of this work. Chapter 3 then presents a system-level overview of Clavius, describing component software structures and the flow of information. This is followed by a more detailed look at some of the algorithmic details of those component structures in Chapter 4, including an exposition on the GS Algorithm and the modular behaviours it uses. Finally, Chapter 5 explores the properties of Clavius experimentally through the analysis of multimodal interaction data with a test application.

Appropriate literature references are discussed where appropriate, according to subject. Since the content of the reference material is spread across fairly disparate domains, this provides a more logical progression through the thesis than having a separate chapter solely devoted to literature review.

1.3.1. Syntax Used in this Thesis. Although there currently exists no concise standard for representing multimodal phrases textually, the convention in Table 1.1 will be adopted for the remainder of this document. Other conventions will be defined as their underlying concepts are introduced. Any parametrization of symbols will take the form of attribute-value matrices, defined in Chapter 2.

Feature	**Description**	**Example**
Speech	Spoken words only	*This is this.*
Speech + point gesture	$\searrow$ refers to the maximal extent of a pointing gesture, and its position in the string reflects its temporal relation with spoken words. Any word or series of words <u>underlined</u> are semantically related to the nearest $\searrow$.	*<u>This</u>* $\searrow$ *is* $\searrow$ *<u>this</u>.*
Speech + area gesture	$\searrow$ and $\swarrow$ refer to the start and end points of a gesture outlining or circling a 2-dimensional area. Their placement in the string reflects their temporal relation with spoken words. Any word or series of words <u>underlined</u> are semantically related to the nearest $(\searrow, \swarrow)$ pair.	*Look in <u>this area</u>* $\searrow \swarrow$.
Speech + symbolic gesture	$\bigoplus_X$ refers to a symbolic gesture, specified by the subscript X. This incorporates mimetic, referential and modalizing gestures.	*Place* $\bigoplus_\triangle$. : draws a pyramid at the location of $\bigoplus_\triangle$.

Table 1.1: Convention used to represent multimodal phrases textually.

CHAPTER 2

Background and Related Work

Before discussing the architectural and algorithmic details of Clavius, some fundamental concepts must be defined formally, in order to motivate design decisions.

This chapter begins with a brief discussion of graph unification (§2.1), which serves as the fundamental operator of Clavius' grammar processor. It continues with a survey of existing approaches to multimodal interaction that diverge fundamentally over varying computational models (§2.2). From here, psychological aspects (§2.3) are discussed before concluding with particular linguistic features of multimodal interaction (§2.3.3).

Further linguistic and mathematical background necessary for the comprehension of this thesis are described in Appendix A.

2.1. Features and Unification

Constraint-based formalisms are typified in computational linguistics by the use of complex structures to describe various features of words, phrases, and sentences. There are several advantages of this approach relevant to MMI. For instance, expressing linguistic and semantic information declaratively frees the grammar from the control process, allowing for a greater degree of re-usability, should different combinations of modalities or mode-tracking software be necessary. Furthermore, expressing linguistic concepts structurally has generally led to smaller grammars than would be necessary otherwise [**88**]. This simultaneously helps reduce both the multiplicative effect of adding more modes on the size of a grammar, and also the generative capacity of that grammar, since irreconcilable differences in the topology or values of these feature structures can be used to prohibit combination of incongruous phrases or words in the search process. Reducing the size of the search space is a chief objective in multimodal research [**49**].

Pollard's Head-driven Phrase Structure Grammar (HPSG) [**86**], for example, is indicative of a constraint-based formalism popularly applied to linguistics in the form of a special representation format called *feature structures*.

2.1.1. Feature Structures. In Clavius, all user activity is described by collections of data structures called feature structures. A **Feature Structure** (FS) , over a set of features FEAT and a set of values VAL is a rooted, connected, typed directed acyclic graph (TDAG) (N, n_0, δ, θ) such that :

- N is a finite set of nodes,
- $n_0 \in N$ is the root of the graph,
- $\delta : N \times$ FEAT $\rightarrow N$ is a partial function. Specifically, the **arc** (or *directed edge*) $\delta_i = ((n_i, \Phi), n_j)$ is a directed connection from n_i to n_j having feature $\Phi \in$ FEAT . Here, in δ_i, n_i is the **parent** of n_j, and n_j is the **child** of n_i. An arc of feature Φ from n_i to n_j is sometimes represented by $\overset{\bullet}{n_i} \overrightarrow{\Phi} \overset{\bullet}{n_j}$.
- $\theta : N \rightarrow$ VAL is a set of tuples (n_i, v) assigning values $v \in$ VAL to nodes. $\theta(n) = v$ is sometimes used to mean the value of node n is v. Values of nodes can be of any type (integer, character,...), but are compared bitwise to determine equivalence. A node can also have `null` value, λ.

A particular **graph**, Ψ_A, is a complete specification of all variables in (N, n_0, δ, θ), distinguished from other graphs by its subscript. Each node n_i in Ψ_A *can* but does not necessarily *have to* be associated with a value (i.e., $||\theta|| \leq ||N||$). In fact, typically only the *leaf* nodes (nodes without any out-going arcs), have an associated value. The notation $n_i \rightarrow v$ refers to node n_i having value v. Nodes without associated values are called *anonymous* nodes.

In general, Clavius allows only one arc of type Φ to lead out of any node n_i. Formally, for any two arcs δ_j and δ_k leading from n_i such that $\delta_j = ((n_i, \Phi_j), n_j)$ and $\delta_k = ((n_i, \Phi_k), n_k)$, if $\Phi_j = \Phi_k$, then it must follow that $n_j = n_k$, and therefore that δ_j and δ_k are the same arc. In certain very specific instances, however, issues of implementation require that this constraint be waived, at the expense of additional computational structure, as discussed in §3.3.1.

The size of a particular feature structure $|\Psi_a|$ is defined as the number of directed edges present in Ψ_a. In §A.1.1.1 it was shown that if Ψ_a has a set N of nodes, then $\frac{|N|(|N|-1)}{2} \geq |\Psi_a| \geq |N| - 1$. This provides an important processing bound, namely that all of Clavius' algorithms that traverse graphs are bounded above by $O(|N|^2)$. This limit can in fact be lowered, as will be described later.

All non-root nodes $n_i,\ i = 1..(||N|| - 1)$ must be **reachable** from the root node n_0. That is, there must be a **path** from n_0 to n_j, which is recursively defined as a sequence of arcs, $S = \{\delta_0 ... \delta_j\}$, such that if $\delta_0 = ((n_0, \Phi_i), n_i)$, then either $n_i = n_j$ (for some Φ_i), or else there exists some path from n_i to n_j. The path from n_i to n_j is sometimes represented by $n_i \rightsquigarrow n_j$, where n_i is called the **ancestor** of n_j, and n_j is the **descendant** of n_i.

Furthermore, all paths in graph Ψ must be **acyclic**, meaning that there does not exist any path S_{ab} in Ψ such that two arcs exist where $\delta_a = ((n_a, \Phi_a), n_k) \in S_{ab}$ and $\delta_b = ((n_b, \Phi_b), n_k) \in S_{ab}$, and $\delta_a \neq \delta_b$. In

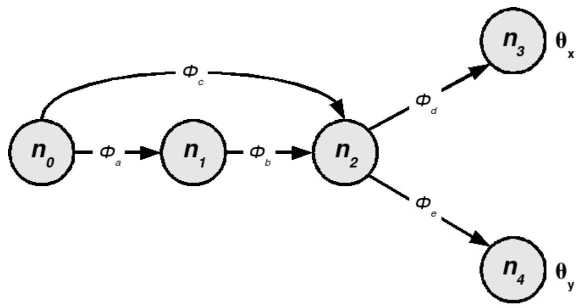


Figure 2.1: An example feature structure graph (N, n_r, δ, θ) where $N = \{n_0, n_1, n_2, n_3, n_4\}$, $n_r = n_0$, $\delta = \{((n_0, \phi_a), n_1), ((n_1, \phi_b), n_2), ((n_0, \phi_c), n_2), ((n_2, \phi_d), n_3), ((n_2, \phi_e), n_4)\}$, and $\theta = \{(n_3, \theta_x), (n_4, \theta_y)\}$

other words, for any path in Ψ, no node n_k in that path can be entered more than once. That is not to say that nodes in Ψ cannot have more than one parent, only that the parents of these nodes cannot also be their descendants. Nodes with more than 1 parent are called **re-entrant**. Although formalisms for cyclic feature structures exist in other disciplines such as uncertainty in artificial intelligence [**90**], these complicate issues of grammar representation and execution and therefore are almost never used in computational linguistics.

A visual example of a feature structure (typed directed acyclic graph) is provided in Figure 2.1, with one re-entrant node (n_2). Further examples of feature structures that represent linguistic entities (via grammar rules), are provided in §2.1.3.

2.1.1.1. *Subsumption of Feature Structures.* A morphism, g, in the context of feature structures is a mathematical operator that maps each node n in one graph to some node m in another, written as $g(n) = m$. A feature structure $\Psi_a = (N, n_0, \delta, \theta)$ is a *substructure* of another $\Psi_b = (N', n_0', \delta', \theta')$ (written $\Psi_a \leq \Psi_b$) if

(i) $\forall n \in N$ and $\Phi \in$ FEAT, if $\exists\delta(n, \Phi)$, then $\exists\delta'(g(n), \Phi)$ and $g(\delta(n, \Phi)) = \delta'(g(n), \Phi)$.
(ii) $\forall n \in N$, $\theta(n) = \theta'(g(n))$

Note that normally some hierarchical operator other than '$=$' is defined for point ii), but this is not the case in Clavius. This basically implies that all the topological and value information of one graph is 'contained' within another.

Feature structures can also be partially ordered according to the degree of their information content by a relation called **subsumption**. Subsumption is defined syntactically by graph morphisms that map all nodes of one graph to those in a subgraph of another. That is, a feature structure Ψ_b (defined above) subsumes another, Ψ_a, (written $\Psi_b \sqsubseteq \Psi_a$) , if and only if $\Psi_a \leq \Psi_b$ and $g(n_0) = n_0'$ for some morphism g. Namely, Ψ_a is a substructure of Ψ_b, where the root of Ψ_a is mapped to the root of Ψ_b by the same morphism that defines that substructure.

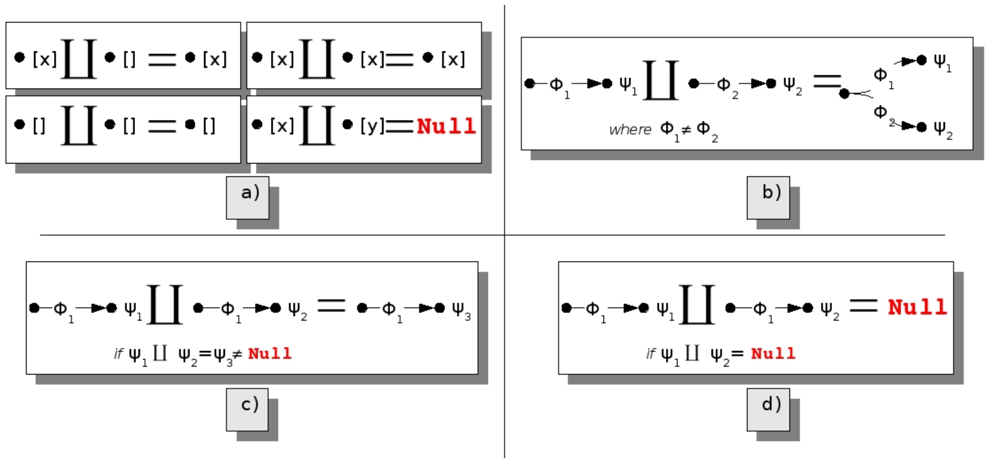


Figure 2.2: Prototypical unifications. In a) all atomic unifications succeed except those having different non-null values. In b) the resulting graph is the union of the information in the component graphs. In c) and d) if outgoing arcs of component graphs have the same feature, the respective subgraphs must also unify.

These are important relations that will be discussed further in §2.1.3 and §4.1.3.

2.1.2. Unification. Unification in NLP was developed separately and concurrently by Kay [**53**] [**54**] and Colmerauer [**24**], who approached it from slightly different purposes. Its use in Clavius follows the former more closely in that it concentrates on unification of structure rather than value terminals.

Unification, denoted by the operator $\sqcup$ serves as the fundamental operation in the Clavius search process. It serves both as an amalgamator of information in two distinct feature structures, and as a boolean test as to whether those structures *can* be combined properly. Since all user activity is represented by these structures, described more concretely in §2.1.3, it is important to be able to combine these, so that graphs representing words can be combined into graphs that form phrases and sentences.

The $\sqcup$ operator takes two FSs as arguments and produces a third. That is, $\Psi_C = \Psi_A \sqcup \Psi_B = \left(N^{(c)}, n_0^{(c)}, \delta^{(c)}, \theta^{(c)}\right)$ means that Ψ_C is the unification of Ψ_A and Ψ_B. It is a recursive operator and has the behaviour simplified in Algorithm 1, given $\Psi_A = (N, n_0, \delta, \theta)$ and $\Psi_B = (N', n_0', \delta', \theta')$.

In the above, SUBGRAPHAT(n) returns the subgraph rooted at node n, ROOTOF(Ψ) returns the root node of graph Ψ, and COPYGRAPH(Ψ) returns a new copy of the graph Ψ (that is, an exact morphism of graph Ψ). At the end of a successful unification, the resultant graph will subsume both of the input graphs, and will generally contain more structural and semantic information. For example, if two graphs can be said to describe a certain word, then their unification will provide more information about that word, if such a combination is possible. For clarification, examples of unification are shown in Figure 2.2.

Unification is obviously a commutative operation such that $\Psi_C = \Psi_A \sqcup \Psi_B = \Psi_B \sqcup \Psi_A$. This is contrasted with similar graph-combination algorithms in other multimodal systems, namely SMARTKOM[**3**],

Algorithm 1: UNIFY

```
Data: feature structures $\Psi_A$ and $\Psi_B$
Result: $\Psi_C = \Psi_A \sqcup \Psi_B$
begin
    if $||N|| = ||N'|| = 1$ then //$\Psi_A$ & $\Psi_B$ are atomic
        $\Psi_C \leftarrow$ Unify_Atomic $(\Psi_A, \Psi_B)$
        return $\Psi_C$
    if $||N|| = 1$ or $||N'|| = 1$ then //only one of $\{\Psi_A, \Psi_B\}$ is atomic
        $\Psi_C \leftarrow$ Unify_Mixed $(\Psi_A, \Psi_B)$
        return $\Psi_C$
    // Both $\Psi_A$ and $\Psi_B$ are complex feature structures
    for all $\delta_A = ((n_0, \Phi), n_i), \delta_B = ((n'_0, \Phi'), n'_i)$ s.t. $\Phi = \Phi'$ do //check arcs of all common
    outgoing features
        // If the nodes they point to unify, success, otherwise fail
        $\Psi_a \leftarrow$ SubgraphAt $(n_i)$
        $\Psi_b \leftarrow$ SubgraphAt $(n'_i)$
        $\Psi_c \leftarrow$ Unify $(\Psi_a, \Psi_b)$
        if $\Psi_c = \emptyset$ then
            return $\emptyset$
        else //Attach the subgraph to $\Psi_C$
            $n_c \leftarrow$ RootOf $(\Psi_c)$
            create $\delta_C = \left(\left(n_0^{(c)}, \Phi\right), n_c\right)$
    // Copy in all subgraphs led to by arcs not common to $\Psi_A$ and $\Psi_B$
    for all $\delta_A = ((n_0, \Phi), n_i)$ where $\nexists \Phi' = \Phi$ s.t. $\delta_B = ((n'_0, \Phi'), n'_i)$ do
        $n_c \leftarrow$ CopyGraph (SubgraphAt $(n_i)$)
        create $\delta_C = \left(\left(n_0^{(c)}, \Phi\right), n_c\right)$
    for all $\delta_B = ((n'_0, \Phi'), n'_i)$ where $\nexists \Phi' = \Phi$ s.t. $\delta_A = ((n_0, \Phi), n_i)$ do
        $n_c \leftarrow$ CopyGraph (SubgraphAt $(n'_i)$)
        create $\delta_C = \left(\left(n_0^{(c)}, \Phi'\right), n_c\right)$
end
```

Algorithm 2: UNIFY_ATOMIC

```
Data: feature structures $\Psi_A$ and $\Psi_B$
begin
    if $\theta(n_0) = \lambda$ then
        $\Psi_C \leftarrow \Psi_B$
        return $\Psi_C$
    if $\theta'(n'_0) = \lambda$ then
        $\Psi_C \leftarrow \Psi_A$
        return $\Psi_C$
    if $\theta(n_0) = \theta'(n'_0)$ then
        $\Psi_C \leftarrow \Psi_A$
        return $\Psi_C$
    // nodes have non-null, non-equal values
    return $\emptyset$
end
```

Algorithm 3: UNIFY_MIXED

```
Data: feature structures $\Psi_A$ and $\Psi_B$
begin
    if $||N'|| > 1$ then only $\Psi_A$ is atomic
        if $\theta(n_0) = \lambda$ then
            $\Psi_C \leftarrow \Psi_B$
            return $\Psi_C$
    else if $||N|| > 1$ then only $\Psi_B$ is atomic
        if $\theta(n'_0) = \lambda$ then
            $\Psi_C \leftarrow \Psi_A$
            return $\Psi_C$
    return $\emptyset$
end
```

where parse graphs are combined according to 'overlay'– a non-commutative approximation of the chart parser described in §2.2.3. The 'overlay' method conditionally unifies utterance parses against background knowledge or default expectations. This background knowledge is thereby used to automatically fill in for unspecified components of a partial parse.

Other common divergences from standard unification includes type-value hierarchies (so that values in θ can be subsumed), and the use of other graph-structured data as semantic frames-of-reference. These are not employed in Clavius, especially in the latter case, so as to minimize the use of 'external' information during graph recombination.

2.1.2.1. *Implementation of the Operator* $\sqcup$. The algorithm that implements $\sqcup$ accepts two feature structures, Ψ_A and Ψ_B, and returns either an empty graph and notification of error, or the unification of the two. The Clavius implementation is recursive and non-destructive, so that the two input graphs are preserved unchanged after unification. Other implementations will sometimes avoid a processing bottleneck incurred by the presence of re-entrant nodes by implementing a *destructive* version of unification that is *at least* linearly proportional in time to the size of the *larger* of the two graphs[**52**]. This, however, typically requires that copies be made of at least one of the input graphs, so that the originals are preserved. This results in a lower bound in time that is linearly proportional to the size of both graphs, or at best of the smallest, if such a distinction can be made in $O(1)$.

The Clavius implementation of Algorithm 1, is $O\left(\max\left(|\Psi_A|, |\Psi_B|\right)\right)$ since every edge of each graph is visited exactly once on successful unification. The best case, however, is $\Omega(1)$ which represents merely the length of one path in each graph from the root node to a value, since the algorithm terminates as soon as an inconsistency is detected. In the typical case this is a *much* more efficient approach than that presented by Jurafsky and Martin [**52**] because, for reasonably complex grammars, the majority of unification attempts will fail.

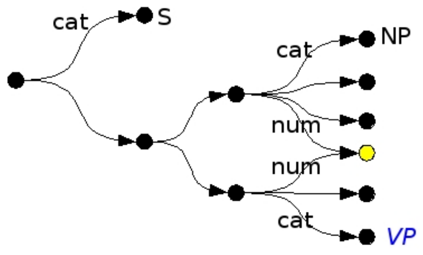


Figure 2.3: Simplified example graphical representation of the grammar rule S $\leftarrow$ NP VP, with the added constraint that the number (NUM) of the noun phrase and verb phrase must agree.

In Clavius, each grammar Γ is associated with a sorting function, β, that uniquely orders the set of all features in a grammar (for instance, by the order that they first appear in Γ). This ordering is essentially arbitrary and serves only to simplify implementation details. If feature Φ_i precedes arc Φ_j according to β, this is written $\Phi_i \prec_\beta \Phi_j$. Since in general outgoing arcs are constrained from a node to have distinct features, β can also be said to order the arcs from a node within some graph Ψ, although again this is purely pragmatic.

In addition to internal (but arbitrary) ordering of graph arcs, the other unique aspect to the implementation of $\sqcup$ in Clavius is its handling of re-entrant nodes. Specifically, during unification, a hash table (RE-ENTRANTHASH is generated on the fly keyed by nodes from input graphs Ψ_A and Ψ_B) tracks their associated node in output graph Ψ_C. For instance, if n_x is a re-entrant node in Ψ_X, and Ψ_X and Ψ_Y are being unified to create Ψ_Z, then RE-ENTRANTHASH$\{n_x\} = n_z$ gives the node $n_z \in \Psi_Z$ that corresponds to n_x. This hash is useful in several ways, especially since it avoids repeated traversals of the same subgraph and keeps re-entrant nodes in input graphs from being translated to multiple nodes in output graphs.

2.1.3. Unification Grammars. All lexical-level terminals (spoken words and gestural atoms) in Clavius are represented as DAGs. Furthermore, all grammar rules such as some i^{th} rule Γ_i S $\leftarrow$ NP VP[1] are represented by DAGs, as exemplified in Figure 2.3, and discussed in §3.3.1). Therefore, *parsing*[2] consists of localized unification of grammar-rule graphs with either lexical terminal graphs, or some other constituent-level graph. This process is elucidated in Chapter 4.

There are several advantages to using unification of feature structures as a methodology for multimodal language parsing, some of which are described in the following subsections. A common representation paradigm for both grammar rules *and* potential parses of user input simplifies the implementation, and allows for subsumption and substructures to be taken advantage of by the GS Algorithm.

[1]This rule states that a sentence (S) consists of a noun phrase (NP, ex. "the Force") and a verb phrase (VP, ex. "is with you")
[2]See Appendix A for more background.

2.1.3.1. *Agreement.* Various agreement phenomena in English constrain the lexical morphology (§A.2.1) in order to help partially distinguish grammatical from ungrammatical phrases. For example, present-tense verbs will often take a terminating *-s* if their subject is third-person singular (eg. *this train* **leaves**), but not if the subject has some other form (eg. *these trains* **leave** , *I* **leave**). The problem is further exacerbated by whether the noun is nominative (*I*,...) or accusative (*me*,...), resulting in an explosion in the size of any representational grammar.

The solution is to parametrize grammar rules, and to use these extra arguments as constraints. For example, S ← NP(*sing*) VP(*sing*) can be used to signify that both the noun and verb phrases have to have singular person. Agreement between information in disparate nodes becomes especially relevant in constraining the search. For instance, one may wish to prohibit unification of a gesture towards an image of a single virtual object with a spoken plural noun phrase such as "*these objects*".

2.2. Divergent Approaches in MMI

In the few decades since Bolt's seminal work (§1.1), there has not yet emerged a single paradigm that accommodates all potential uses of multimodal interaction, or all theoretical methodologies thereof, despite some vary broad recent attempts by the W3C to create a unified standards base [**25**]. Varying techniques have been applied to solving the problem of combining information across disparate modes such as nearest-neighbour temporal alignment, neural networks [**27**], salience measurement [**33**], and biologically-inspired rhythm tracking [**114**].

Several *additional* divergent formalisms that are more relevant to the development of Clavius are discussed in subsections §2.2.1 through §2.2.4, below.

2.2.1. Semantic Fusion. Given the availability of unimodal speech and gesture recognizers, a popular methodology has simply been to combine the information held in the 'best' unimodal hypotheses of utterances after the fact. For example, once a speech recognizer has a list of its n-best interpretations of a sentence, a multimodal component may attempt to cross-correlate their component information with complementary information in unimodal gestural interpretations from an independent tracker at the utterance level.

An increasingly popular approach to this integration process has been to merge attribute/value data structures recursively [**112**] according to predefined rules, although the exact approach used to correlate such complementary information varies on the representation schemes of the recognizers used. Such a methodology, though, invariably depends on complete sentence-level hypotheses to be determined before hand, requiring that such processing be done only after an utterance has been completed. The implications of using a top-down approach as a means of rectifying problems of accuracy are further compounded by the lack of shared processing across modes that could greatly refine or direct the search process during its initial pass

[**101**]. Integration at the semantic level can also be restrictive, especially if there is a fundamental lack of understanding of the human speech/gesture production mechanism. Despite these limitations, such systems can be trained very easily on unimodal data, which are more prevalent and easy to collect.

Classic examples of systems based on semantic fusion include Put That There [**14**], ShopTalk [**21**], QuickSet [**22**], and CUBRICON [**73**].

2.2.2. Prosodic Co-Analysis. Pitch/accent association in English often underlines discourse-level aspects of information flow. The *fundamental frequency*, F_0, is the physical aspect of speech corresponding to audible pitch, inversely derived by the time between two successive glottal closures [**40**]. Efficiently estimating F_0 often involves counting zero-crossings of the speech signal within short time periods, although a more accurate method is to take the inverse Fourier transform of the energy spectrum $|S\left(e^{j\omega}\right)|^2$ [**77**].

The shape of the F_0 contour can be used to decipher the delimitation of phrasal units, but more importantly can be used by the speaker to emphasize certain words, thereby shifting their semantic content [**76**]. For instance, by shifting emphasis on the end of the phrase *It's the baker, Mister Jones.*, the speaker either informs the listener as to the identity of a third-person baker in the appositive case, or merely inflects as to the person being addressed in the vocative case, as demonstrated in Figure 2.4.

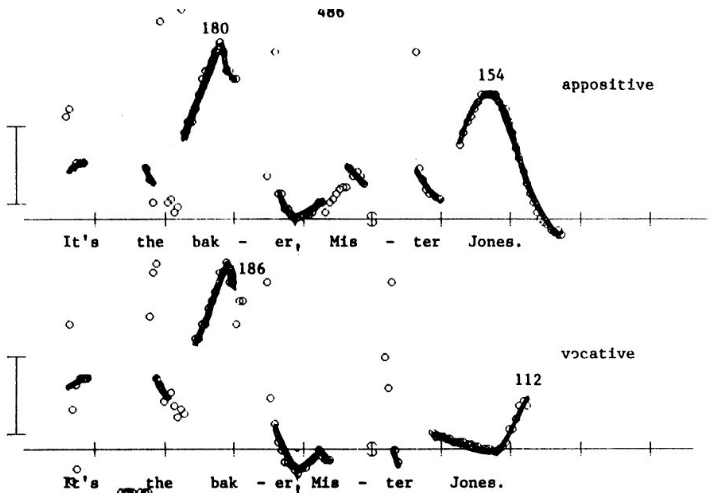


Figure 2.4: Sentence meaning as a function of F_0 contour (frequency vs. time) for appositive and vocative utterances (from O'Shaughnessy [**76**]).

This approach has been used statistically in MMI to provide a model relating the temporal alignment of active hand gestures to words having prominent speech segments (ex. the word "this" in "*erase* $\searrow$ *<u>this</u>*") [**57**]. It is adopted to Clavius as discussed in §4.12.

2.2.3. Deterministic Unification. Unification of feature structures (§2.1) is often implemented in MMI within the context of semantic fusion, although this is not theoretically necessitated. It differs slightly from common semantic fusion, though, in that it restricts all unimodal recognizers to be represented within a common feature structure framework, typically a unification grammar [**50**][**51**] [**43**]. The prototypical approach to multimodal parsing with unification grammars is shown in Algorithm 4 (from Johnston [**49**]).

Algorithm 4: ChartParse

```
begin
    while $\exists$ edge in AGENDA do
        remove best edge from AGENDA, and make it current_edge
        for each EDGE $\in$ CHART do
            if current_edge $\cap$ EDGE $= \emptyset$ then
                create set NEWEDGES $=$ ($\cup$ current_edge *EDGE) $\cup$ ($\cup$ EDGE*current_edge)
                add NEWEDGES to AGENDA
    add current_edge to CHART
end
```

Obstacles to this approach have included computational complexity (effectively all admissible edges are created) and tractability. Chapters 3 and 4 discuss how these obstacles, and others met by other implementations, can be overcome within the unification model in Clavius.

2.2.4. Weighted Finite State Machines. Partially in order to overcome perceived issues of computational complexity in unification-based parsing, some research groups abandoned this approach in favour of finite state machines [**50**] [**51**]. Finite state machines have been applied extensively to many aspects of language processing, including phonology, morphology, chunking, parsing, machine translation, and general speech recognition. Their adaptability from data, their effectiveness of decoding, and their compositional calculus make them an attractive general-purpose representational framework.

In a multimodal interface with n modes, a finite-state automaton over $n{+}1$ tapes is necessary to construct a joint interpretation, where the $n + 1^{th}$ tape represents the semantic output, which may take the form of a compositional predicate calculus. Fortunately, such machines can be encoded directly from context-free grammars, but they also abandon several advantages of feature structures – namely the ability to represent complex semantic information.

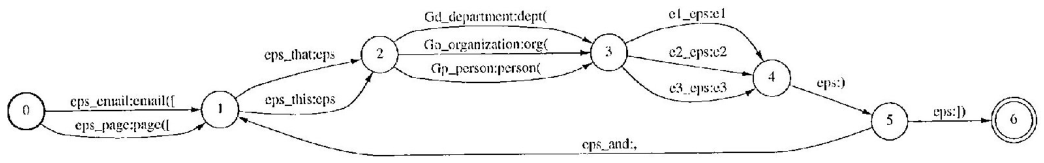


Figure 2.5: Transducer relating gesture, G, and speech,S, to meaning, M, $\tau\colon (G \times W) \rightarrow M$ (from Johnston and Bangalore [**50**]).

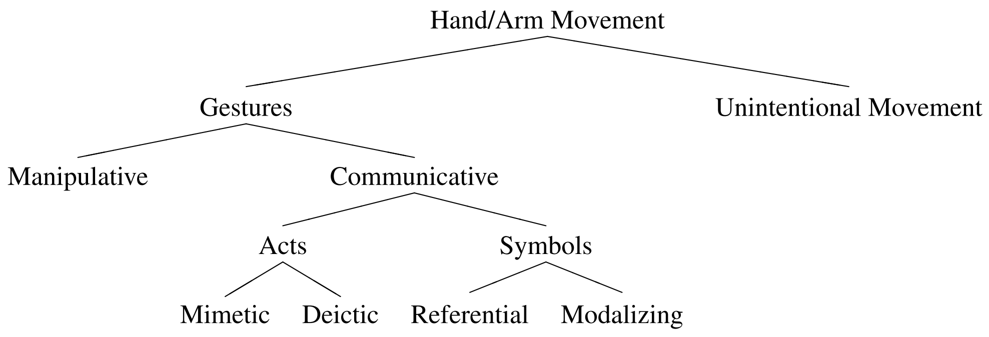


Figure 2.6: A hierarchical ontology of human gesticulation.

2.3. Psychological Aspects of Multimodal Human-Human Communication

Singular conceptual representations – especially those related to spatial descriptions – can be manifested in variety of combinations of speech and gesture [**59**]. For instance, Cassell and Prevost [**19**] show that roughly 50% of the meaningful semantic components of spatial descriptions between humans are expressed either by speech or by gesture exclusively while the other 50% were encoded *redundantly* across both. This suggests, at least, that humans regularly split conversation over multiple channels – but in which situations does this occur, and how is it useful?

Figure 2.6 shows a hierarchical decomposition of human gestures according to their usage and meaning (from Pavlovic [**83**]). **Deictic** gestures, in particular, are generally used in conjunction with speech to indicate an object or location, given the context of the speaker's frame of reference. This lessens the cognitive load on both the speaker and hearer, especially in spatial descriptions [**70**], in accordance with Zipf's Principle of Least Effort [**121**]. Specifying the identity of referred objects in this way produces shorter sentences requiring fewer assumptions to be made as to the knowledge of the hearer. Gesticulation might also be useful in helping the speaker conceptualize the subjects of their communication, and to retrieve the appropriate lexicography in their description [**60**]. That is, hand and arm movements seem to facilitate both spatial working memory and speech production [**55**]. Naturally, encouraging such an important verbal aid in human-computer interaction should therefore be desirable.

2.3.1. Anaphora.

Series of actions are not often performed in isolation and users may typically need to refer to objects repeatedly in subsequent commands. This is especially pronounced in NP/pronominal relationships across sentences, as in :

Put ↘this, and ↘this in ↘ here ↙. Colour them blue.

where the noun phrase $(_{NP}\,(_{PRP}this)\,and\,(_{PRP}this))$ refers to the same collection of objects as $(_{PRP}them)$. A multimodal system must be able to infer equivalence of this type if it is to take advantage of its users' natural speaking patterns. Approaches to this phenomena in NLP have included simple surface-level heuristics [**10**] and deeper use of semantics [**42**]. In MMI, restricting the language to query or command-based interaction has allowed researchers to implement simple context trees to localize information [**21**].

2.3.2. Form vs. Meaning. In speech, the distinction between *lexical* and *sentential* meaning is that the relation between the word form and its semantic realization is arbitrary in the former, and compositional in the latter. For instance, the lexical meaning of *dog*, *chien*, and *hund* are all the same, though their surface-structure differ. On the other hand, the difference sentential meaning of "*the man ate the lobster*" versus "*the lobster ate the man*" depends on relative ordering of constituents to instantiate verb/object arguments for the verb.

Psycholinguistic [**68**] and engineering [**56**] studies suggest that even deictic gestures also do not exhibit a one-to-one mapping of form to meaning. Multimodal interaction in general deals with highly ambiguous input spaces, where the emphasis must lie in functional grammatical constructions.

2.3.3. Peculiarities of Multimodal Language. A significant dichotomy between multimodal and unimodal spoken language is the use of locative constituents, which are almost always instantiated as prepositional phrases depicting location, as in *"I left the cake out [in the rain]$_{LOC}$"*. Surprisingly, some sources suggest that the canonical English S-V-O-LOC [3] word order is almost completely supplanted in pen/voice interaction by a LOC-S-V-O word order. Two separate research teams found corroborating evidence that for a fixed set of speakers, $\tilde{}95\%$ of locative constituents were in sentence-initial position during multimodal interaction, compared to $\tilde{}96\%$ in sentence-final position (as in the example above) for unimodal speech [**80**] [**79**] [**7**]. The Clavius data collection, described in Chapter 5, confirms higher rates of sentence-initial locatives, though generally within imperative command structure.

2.3.3.1. *Conversational Multimodality.* As previously mentioned, use of gesture in multimodal conversation generally leads to simpler sentences. This can come with the cost of an added degree of dysfluency in practise, however. Specifically, vocative dysfluencies ("*uh*", "*err*",...) must be considered in addition to gestural dysfluencies – namely hesitations or false-starts during gesticulation.

Current systems implementing multimodally conversing virtual agents, while still in very early development, have resulted in generally encouraging or favourable impressions by test users, although these still appear to be most taken by the novelty of MMI [**11**].

[3] SUBJECT-VERB-OBJECT-LOCATIVE

CHAPTER 3

Clavius System Architecture

Clavius is a generic multimodal dialogue system that employs an extensible best-first search strategy over stochastic/semantic data structures in order to multiplex multiple streams of human activity into a control stream to the application to which they refer.

This chapter covers an architectural overview and a component analysis of the system. It begins with broad usage scenarios (§3.1) and a high-level, structured decomposition of software components (§3.2), followed by a more pragmatic discussion of their functionality (§3.3 - §3.8). An overview of the deployment methodology (§3.9) is then provided, and is concluded by notes regarding the general approach to software engineering (§3.9.1). Algorithmic details, specifically those related to mathematical and linguistic theory, are covered in Chapter 4.

3.1. Towards Generic Multimodal Understanding

The purpose of Clavius is to act as an interpretation layer between an arbitrary software application and its users. Clavius is the result of an effort to combine sensing technologies for several modality types, speech and video-tracked gestures being the most important among them, within the immersive virtual enclosure [**15**] [**94**] at McGill University's Shared Reality Environment, shown in Figure 3.1.

Although Clavius is not constrained to certain modality combinations, application types or interaction schemas, it is expected to be used generally in command-based interaction, where the stereotypical usage scenario involves the relocation and manipulation of virtual objects, as exemplified by the sequences in Table 3.1. These consist of simple commands on objects, and their immediate operation by the application.

3.2. Systematic Component Decomposition

Figure 3.2 shows a high-level breakdown of the most significant software components of Clavius, and the communication paths between them, inspired initially by work from Neel and Minker [**23**], and discussed further in the indicated sections. Note that, although all components within Clavius are customizable at

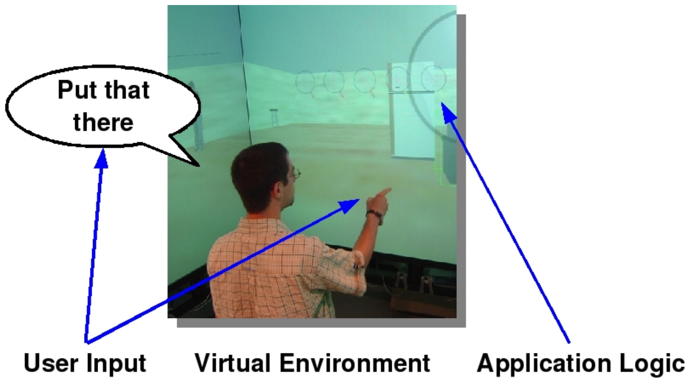


Figure 3.1: The Shared Reality Environment.

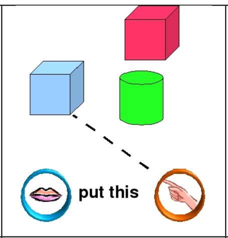
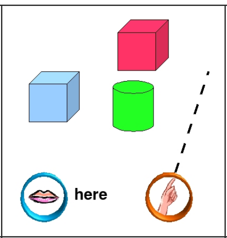
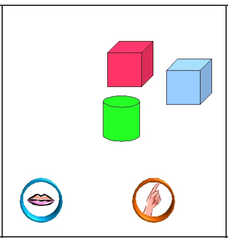
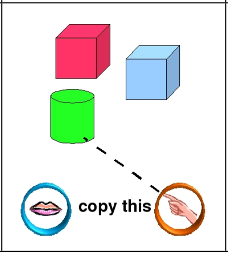
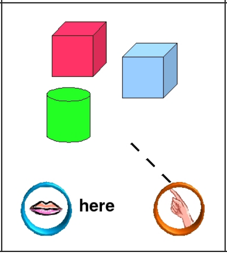
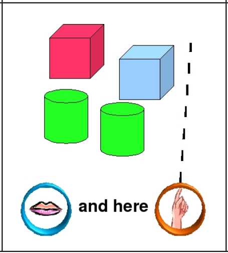
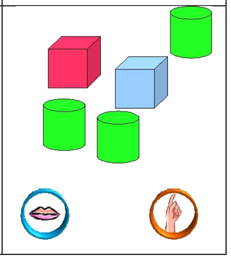
Translocation				
Copy and Pasting				

Table 3.1: Example interaction sequences with a virtual blocks world using speech coupled with pointing gestures.

startup by an XML[1]-based configuration manager (§3.4), the chief operating characteristics of language and behaviour are determined by the application designer at runtime externally from the core system, making it highly customizable and scalable. Approaching the problem in this way also helps to localize functionality into smaller modular components [**72**].

Subsections §3.2.1 to §3.2.3 give a simplified hypothetical chain of operation through the system between the time sensory data is first introduced, and the eventual construction of a valid hypothesis for the intended meaning of a user's utterance. In the example covered in the following subsections, it is assumed that a hypothetical user has uttered the multimodal phrase "*put* ↘*this* ↘*here*" in order to move a virtual object around an immersive environment.

[1]eXtensible Markup Language.

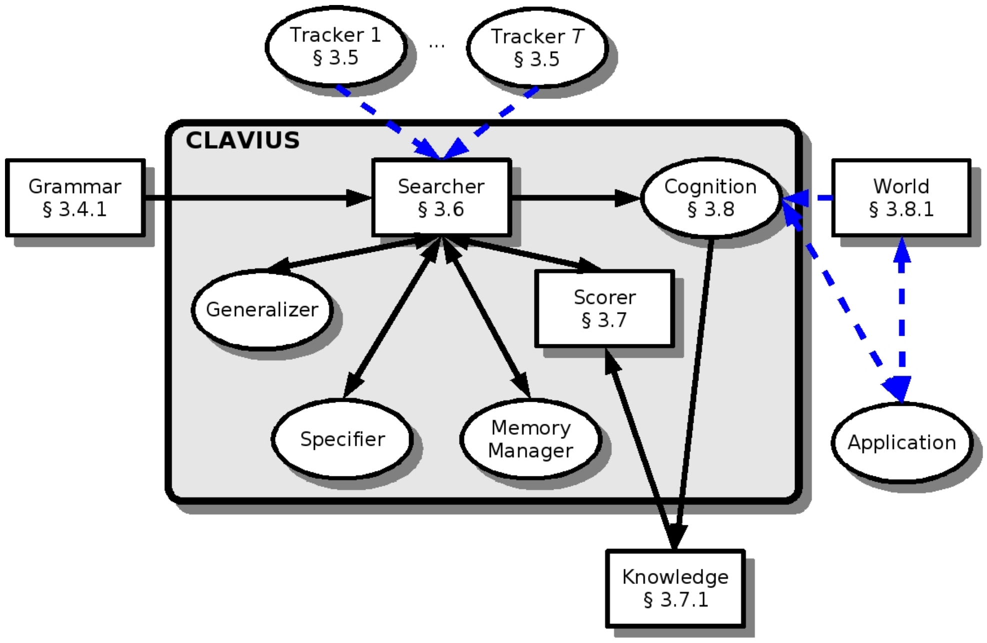


Figure 3.2: Simplified information flow between fundamental software components in Clavius. Rectilinear and ovoid elements represent static objects and dynamic threads, respectively. Hashed arrows represent network communication.

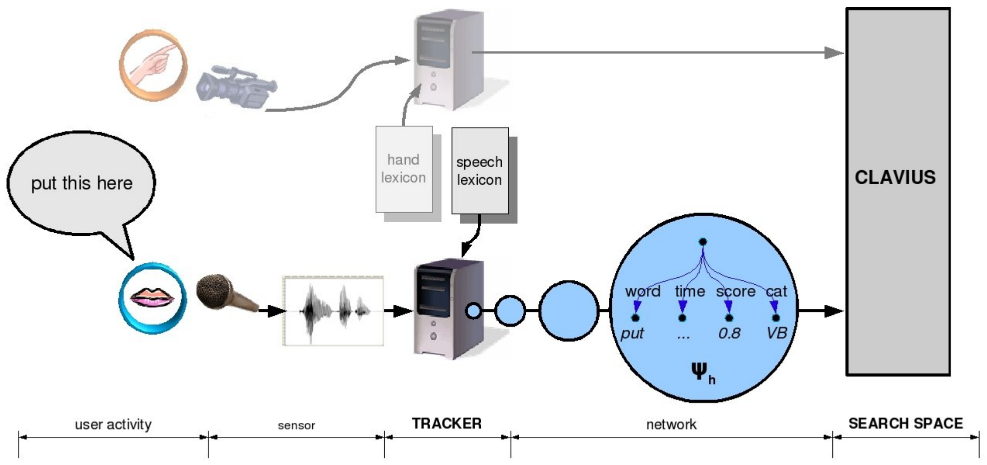


Figure 3.3: During tracking, user activity is sensed and analyzed by the tracker before being tokenized and sent to Clavius' search space.

3.2.1. Stage 1 : Tracking. **Trackers** are independent programs that sense and monitor a particular input modality, such as speech, and perform rudimentary interpretation of activity in that modality. Specifically, a tracker will classify a given input signal into one of the equivalence classes provided to it by a 'lexicon', or vocabulary. For example, a speech tracker monitoring the signal level on a microphone recognizes individual words, and a gesture tracker monitoring video will recognize particular gestures.

These interpretations are sent in the form of fully qualified graphs, or '**tokens**' to Clavius for further processing over a TCP/IP network connection according to a pre-determined XML format (see §3.5.1). Trackers are responsible for providing each token with a likelihood score. Given the above example, if the speech tracker is only 80% confident that a word it hears is *put*, then the token it sends to Clavius will have a score of 0.8, as exemplified by Ψ_h in Figure 3.3.

A collection of T trackers monitor user activity asynchronously and independently of one another over the M modalities used by an application. Typically, $T = M$, so each modality is monitored by a unique dedicated tracker, but Clavius allows for $T > M$, so that certain modalities can be monitored by more than one tracker. Implications of this capability are discussed in §3.5.3. Trackers will typically be implemented as stand-alone programs run on machines directly attached to appropriate sensing equipment, and they can be implemented in any language capable of TCP/IP network communication.

This component is covered in greater detail in §3.5.

3.2.2. Stage 2 : The Searcher. Upon the receipt of token Ψ_h for *put*, Clavius inserts this hypothesis into the **searcher**. The Searcher consists of specialized data structures and the concurrent operators acting upon them to determine the best interpretation for an utterance, given the input. The **search space** is composed of numerous substructures containing hypothetical partial parses of which a *subset* potentially lead to a sentence-level interpretation. These partial parses are built according to a specialized multimodal **grammar**, based on the input tokens provided to the search space by the trackers. In the ongoing hypothetical scenario, if at this point three tokens have been inserted into the search space by the trackers – namely Ψ_h : (VB put), Ψ_i : (POINT $\searrow$), and Ψ_j : (DT $this$), then the task of combining these into grammatical phrases falls upon two concurrent processes: the **generalizer** and the **specifier**.

3.2.2.1. *Generalization and Specification.* Generalization provides the 'bottom-up' component of the GS Algorithm, according to the terminology of Jurafsky and Martin [**52**]. This process takes partial parses from the search space and attempts to attach these to the right-hand sides (RHS) of grammar rules. For example, as shown in Figure 3.4, if Γ_r : ObjectReference $\leftarrow$ DT POINT is a rule in the grammar, and the parse Ψ_j : (DT$this$) exists in the search space, then a new parse Ψ_k : (ObjectReference(DT $this$)(POINT)) is created that is then put back into the search space. Naturally, this process leads towards sentence-level parses.

Similarly, specification provides the 'top-down' component of the GS Algorithm. This part of the process combines two partial parses having the same top-level category in order to amalgamate their respective information content. Continuing with the above example, at this point the generalization process has taken place as shown in Figure 3.4, and the partial parse Ψ_k it has generated exists within the search space. Also, a

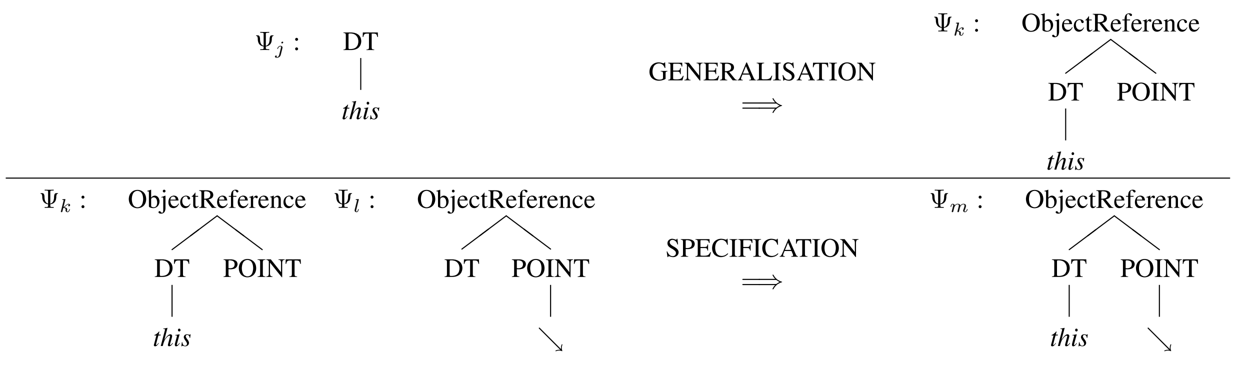


Figure 3.4: Simplified Generalisation and Specification examples.

similar process has gone into producing Ψ_l : (ObjectReference(DT)(POINT $\searrow$)) in the search space – that is, it is the result of another generalization step based on the input token for the gesture "$\searrow$". The Specifier then combines Ψ_k and Ψ_l to form a new partial parse Ψ_m which combines the information of the two, as shown in Figure 3.4.

The combined process of the GS Algorithm is described in detail in §4.1. Both generalization and specification choose which partial parses to consider based on the individual **scores** that all such graphs possess, discussed below.

3.2.2.2. *Memory Management.* Partial parses remain in the search space indefinitely unless forcibly removed. A **memory manager** decides if any parses should be removed, and if this is the case – it performs the expulsion. For example, if the speech tracker from Stage 1 has also produced hypotheses that, instead of "put", the user said "pat" or "pot", then graphs representing these alternatives may be deleted when it becomes clear that neither will form part of an acceptable sentence-level hypothesis. This process is described in greater detail in §4.1.5.

3.2.3. Stage 3 : Application Interface. While the three threads of the Searcher work to form partial parses, a fourth thread – **Cognition** – monitors the potential sentences that they have so far produced. Given various types of ambiguity, there may be several divergent explanations for a user's input. For example, if there is some acoustic or sensory noise obscuring the input, **Cognition** may need to decide between hypotheses "*Cut <u>his</u> $\searrow$ hair.*" and "*Put <u>this</u> $\searrow$ there $\searrow$*".

At a certain point, Cognition will select the latter from the search space and send this accepted hypothesis to the application in the form of a graphical structure simplified in Figure 3.5. The application, on receiving this interpretation, will perform whatever action is appropriate – such as moving the indicated object to the indicated location. This is normally also accompanied with a change of state information shared between Clavius and the application.

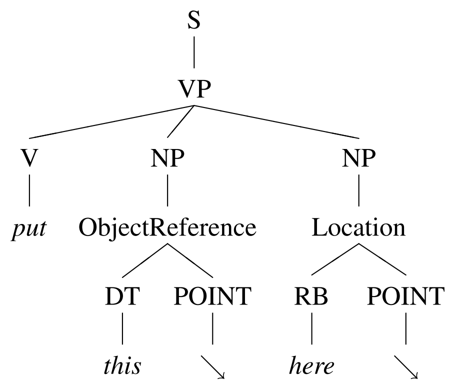


Figure 3.5: A simplified grammatical expansion of the phrase *put* ↘*this* ↘*here*.

Further detail on the engineering aspects behind each component are provided in the sections below.

3.3. Data Representation

The fundamental data type in the Clavius framework is the directed acyclic graph, whose characteristics were introduced in §2.1. Every node in a graph has reserved memory space to hold arbitrary data types as the atomic 'value' at that node. This is accomplished by allocating with the `void *`, and encapsulating the data type and size within each graph node.

Important data types include:

ERRCODE: is an 8-bit mask that is the return value of almost all functions. `0x00` represents function success, `0xFF` represents function failure (error), and `0x01` represents null output.

SCORE: this is a double-precision floating point that is used in the ordering of all partial parses.

TIME: the time format is not strictly enforced by Clavius, but must be consistent between all trackers and must be amenable to ordering using the `strcmp` string-comparison system call. In practise, time has been represented by an integer string of the form "s:u", where s and u are the seconds and (remainder) microseconds from Epoch (00:00:00 UTC, January 1, 1970).

3.3.1. Clavius-specific constraints on DAGs. Certain assumptions were made in the implementations of graphical structures and associated operators in Clavius, based on the fact that these structures would primarily be used to represent phrases and words. Therefore, certain features are required to be present in all graphs in the search space. Those features, and the meaning of their values, are:

cat: The grammatical category associated with the graph.

score: The raw score associated with the graph.

time: The time spanned by the encompassing graph.

content: Any semantic variables associated with this graph. The value of this feature is an arbitrary subgraph of the grammar engineer's design – and can point to a leaf node of null value. Its presence, however, is assumed for the purposes of several utility algorithms.

The value of the time feature is an internal node rooting of a binary tree having subfeatures **t_start** and **t_end**, as indicated in Figure 3.6.

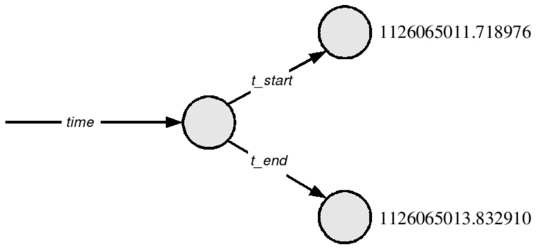


Figure 3.6: Required structure of TIME feature (with example time values).

Additionally, all graphs implementing grammar rules are required to include the following arc features:

rhs: This arc points to an internal node representing the set of constituents on the right-hand side (RHS) of the rule it represents.

constituent: A constituent of a grammar rule is an element of the RHS of a rule, such as B and C in A ← B C. To represent multiple constituents in a feature structure, an exception to the regular condition that all outgoing arcs of an internal node have distinct features ($\Phi_i \neq \Phi_j$) (§2.1.1) is necessary. Therefore, the RHS node n_j ($\delta_i = ((n_i, \text{RHS}), n_j)$) can have multiple outgoing arcs of type CONSTITUENT, for each element on the RHS of the rule.

Certain modules associated with the scoring of partial parses require that additional constraints be placed on the structure of DAGs. These are discussed where appropriate in sections §4.5 and §4.10.

3.4. Configuration Management

All software components query a central configuration manager that is instantiated at runtime with command-line parameters and a specified XML-based configuration file format. All configuration data is maintained in key-value pairs for fast hash-based retrieval. Among the important system-level configuration details are the IP network addresses of all components, the location and login details (username/password) for all relevant databases. Other configuration details are described where appropriate throughout this thesis.

3.4.1. Multimodal Grammars. The most significant configuration of Clavius' run-time is the grammar, which is also specified in XML format by the interaction engineer. This grammar is essentially an order-invariant list of grammar rules, as shown in the example Clavius grammar in Appendix B.

3.5. Input and Tracking

As described in §3.2.1, trackers are implemented separately from Clavius. This approach was taken specifically to allow alternate trackers to be swapped in and out without altering the complex internal structure of Clavius. Trackers will typically be modifications of existing software that have been designed for specific sensory applications.

Each tracker is instantiated independently of Clavius, and must be active and awaiting input when Clavius is started. All device driver handling is performed by the tracker, which is also responsible for the scoring of its own hypotheses, according to its own lexicon and configuration.

3.5.1. Communication Protocol and Lexica. Lexica are specialized vocabularies in Clavius that describe 'words' in terms of structured argument/value pairs. These are specified in XML, as exemplified in Figure 3.7, and serve as the primary run-time descriptor for trackers.

Trackers essentially select from the provided lists of lexical entries to describe their monitored input, and send the appropriate `clav_token` XML elements to Clavius over TCP/IP. These XML elements can optionally be compressed for network transmission, though this is not necessary. The format of the received tokens is validated by Clavius upon receipt according to the same XML lexica used to configure the trackers.

In cases where arguments are not specified by the lexicon – such as the (x, y) co-ordinates in the example in Figure 3.7 – the tracker is also responsible for filling these with appropriate values.

```
...

<clav_token>
   <mode> SPEECH </mode>
   <cat>  DT </cat>
   <num> s </num>
   <word> a </word>
</clav_token>

...

<clav_token>
   <mode> MOUSE </mode>
   <cat> DEIXIS </cat>
   <x> </x>
   <y> </y>
   <left> </left>
   <middle> </middle>
   <right> </right>
</clav_token>

...
```

Figure 3.7: Example lexical entries for speech and mouse trackers. Note that empty fields in lexica (for instance, X or Y, above) are typically filled at run-time with values from the tracker.

3.5.2. N-Best Lists. 'Noisy' modes such as speech or vision-based tracking are often difficult to interpret and there may be multiple possible hypotheses for the same input. At this stage of the recognition process, there is no way to make use of contextual information to help decide between ambiguous possibilities, so in theory they should all be considered by Clavius. In practise, though, only a subset of the competing hypotheses survive to be sent to Clavius in the form of an n-best list. Since each tracker determines the scores of each token, these can be ordered by decreasing likelihood and either the best n are sent for some fixed cutoff n, or n can be dynamic, with the threshold set by some minimum allowable score.

Once sent to Clavius, the n-best list is broken up into its component tokens and inserted individually into the search space. Additional constraints must then be enforced by Clavius in order to ensure that these tokens are considered as disjunctive alternatives for the same input, as discussed in §4.4.

3.5.3. Strategies for Redundant Tracker Combination and Selection. As previously mentioned, the Clavius architecture allows for $T > 1$ trackers to monitor the same mode simultaneously. This is, of course, not the expected deployment scenario – but it is possible that the interaction engineer might have multiple pieces of tracking software for the same tracked mode, and reason to assume that putting them in parallel will aid the recognition process.

All trackers are meant to identify their mode to Clavius by a 'mode key'. Clavius identifies redundant trackers by grouping them into sets by common mode key. When a tracker in one of these groups reports input to Clavius, it is then *possible* (though not necessary) for Clavius to wait a specified time for the other redundant trackers to report their interpretations. At this point, Clavius will apply one of the following combination mechanisms, according to its configuration:

hard merge: Clavius essentially selects one of the T trackers as being the best indicator of actual user input, and inserts *only* the tokens from this tracker into the search space. The tracker selected has the highest average score among its n best hypotheses, and can therefore be considered as the most 'confident' in its prediction.

soft merge: n-best lists of tokens from all redundant trackers are concatenated into a single list. Redundant tokens in this list are those representing the exact same hypothesis (and hence having the exact same argument/value structure), but coming from different trackers. These redundant tokens are identified and replaced by a single token having the same argument structure, and a score representing the average of all redundant tokens. The resulting list contains no redundant tokens, and the n-best of this list are then sent to Clavius. This mechanism provides a good heuristic combining the confidence in a hypotheses across multiple trackers.

no merge: All hypotheses from all trackers are inserted immediately into the search space by Clavius. This method is not preferred, since the distribution of scores becomes uncorrelated across tokens and redundant tokens will clutter the search space.

In all cases, the same mechanisms used to prevent multiple uses of ambiguous input from n-best lists are used to prevent multiple uses of redundant input from multiple trackers, as discussed in §4.4.

3.5.4. Continuous Speech (dictation) Tracker. Continuous speech dictation provides the recognition of individual words, while the source speech is still being uttered according to the lexicon is shown in Table 3.4.

The CMU Sphinx-4 framework [**63**] is the latest addition to Carnegie Mellon's suite of open source speech recognition tools and serves as the basis for speech recognition in Clavius. It is a collaborative effort, jointly designed by CMU, Sun Microsystems Laboratories, and Mitsubishi Electric Research Laboratories.

Sphinx includes an acoustic trainer (limited and fully continuous) and various speech decoders capable of n-best list generation and phoneme recognition. It is capable of bigram and trigram language models – which will become relevant to specialized constraints discussed in §4.8 and uses limited subphonetic HMM topologies on phones, diphones and triphones.

The main Sphinx-4 decoder is based on the conventional Viterbi search algorithm and beam search heuristic, and uses a lexical-tree search structure similar to its predecessors but with improvements in speed and accuracy [**62**]. A high-level decomposition of the framework is shown in Figure 3.8, where the Linguist and Decoder components of the recognizer have been modified for Clavius.

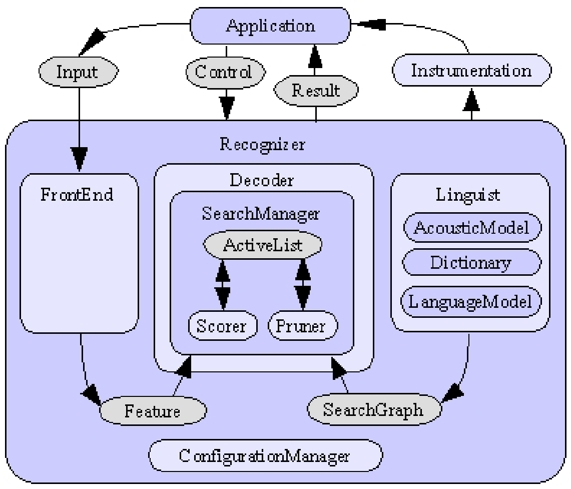


Figure 3.8: Architecture of the CMU Sphinx-4 framework [**110**]

3.5.4.1. *Sentence recognition vs. true dictation.* CMU Sphinx only begins recognition of raw PCM[2] recorded speech once a 'timeout' has been observed – that is, a short period of low signal activity on the microphone. The result is that CMU Sphinx can only provide the n-best *sentence-level* hypotheses for its input, defeating some benefits of the Clavius architecture. Although 'true' commercial dictation systems exist that can report on word-level hypotheses as soon as they occur, such as Dragon system's NaturallySpeaking [**6**], these were not available to this project.

[2]Pulse Code Modulation.

Prosody Lexicon		
Word	Arguments	Purpose
Shift	direction	either a raise or drop in pitch
	amount	the degree of difference in F_0 detected

Table 3.2: The prosody lexical schema.

For the purposes of the current Clavius implementation, the CMU Sphinx framework was used to *emulate* a dictation system. Specifically, once a sentence is recognized by Sphinx, it is split up into its component words which Sphinx can time stamp. If a given word is polysemous and belongs to $n > 1$ parts of speech (the word "box" can be both a noun, and a verb, for example), then n graphs are encoded in the appropriate XML format, and sent individually to Clavius as an n-best list. In this way, the need for part-of-speech tagging is completely circumvented and the parsing algorithm is left to select from the available word-tag associations itself, according to the relative scores of graphs using the different interpretations.

3.5.4.2. *The JSGF Grammar Format.* CMU Sphinx also uses a different grammar format from Clavius, and therefore a unimodal 'speech-only' grammar must be extracted from the multimodal Clavius grammar. Sphinx uses the Java Speech Grammar Format (JSGF) which is a popular, BNF-style representation scheme [**5**]. Currently, the responsibility of translating the Clavius grammar to JSGF format for the benefit of the CMU Sphinx-based speech tracker lies with the interaction engineer.

3.5.5. Continuous Speech (prosodic) Tracker. Acoustic prosody has been used in MMI as a means of identifying important time- and semantically correlated expressions across modes [**57**] where dramatic shifts in a speaker's pitch (fundamental frequency, F_0) can be indicative of important activity in another mode.

The autocorrelation algorithm [**13**] measures F_0 within windows of 1200 samples, $s[0..1200]$, for speech sampled at 44.1 kHz. Given s, the algorithm shifts these samples towards the right in increments δ, from $\delta = 1$ to $\delta = 600$, providing new sample sets $s_\delta[0..1200+\delta]$. For each shifted sample set, the autocorrelation function $r(s) = \sum_{i=600}^{1200} |s[i] - s_\delta[i]|$ measures the absolute difference between the original and shifted sample sets. As the shift approaches F_0, the value of $r(s)$ decreases, and thus the δ giving the minimum $r(s)$ recorded is taken as the value of F_0 for the windowed speech[3].

Given changes in F_0 over the speech signal, dramatic shifts can be determined heuristically as changes above some minimum threshold α, within a small window of time t, where α and t are chosen at configuration time. Such a shift constitutes the lexical entity of this tracker, as shown in Table 3.2 and further discussed in §4.12. This approach is an approximation to the method used by Kettebekov, where prominent segments are identified according to the maximum gradients of the pitch slope [**57**].

[3]Normally, the differential of the autocorrelation function would be used to detect minima

3.5.6. 2D/3D Hand Position Tracker. In Clavius, the desired mode of gestural interaction involves untethered deictic arm gestures in an immersive virtual environment. Tracking the distance and angle between the user's head and their hand (or hands) for this purpose is accomplished via an overhead camera. For instance, to track the 2D top-down head position, image differencing (background removal) is performed between the current video frame and a reference frame averaging pixels of the 3 previous frames. This pixel difference can be assumed to correspond primarily to the user's body, the centre of which can be assumed to correspond to the 2D position, (x_h, y_h), of the user's head [**109**].

In order to locate the hands in the pixel difference map, skin colour segmentation is used, as shown in Figure 3.9. Conditions for skin colour (based on Peer and Solina [**85**]) have been determined empirically for the illumination conditions in the SRE environment, dependent on camera sensitivity, and adhering to the following constraints in RGB space:

$$\begin{aligned} \{r,g,b\} \in \text{SKIN} \leftrightarrow\ & (r > 70) \wedge (g > 30) \wedge (b > 20) \\ & \wedge (\max[r,g,b] - \min[r,g,b] > 55) \wedge (|r-g| > 15) \wedge (r > g) \wedge (r > b) \end{aligned} \tag{3.1}$$

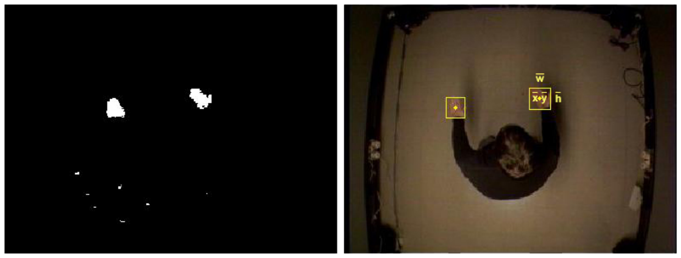


Figure 3.9: Example of hand segmentation. Binary mask resulting from the skin segmentation process (left). Bounding boxes (right) are created around the two largest blobs of connected skin pixels. The bounding box is defined by its centre coordinate, width and height.

Given the overhead positions of the head and right-most hand,(x_r, y_r), assumed to be the dominant hand of the user, it becomes trivial to calculate the distance $d = \sqrt{(x_h - x_r)^2 + (y_h - y_r)^2}$ and angle $\theta = \arccos \frac{|y_h - y_r|}{d}$ between them, assuming the user is facing the *top* of the frame (normal to $y = 0$).

Once d and θ have been calculated, rudimentary gesture recognition can occur. Three types of gestural segments are important to Clavius, namely *preparation*, *stroke/point*, and *retraction* as suggested by McNeill [**68**]. Estimating these three gestures can be done heuristically by comparing measurements of d with customizable thresholds D_{inner} and D_{outer} as follows:

preparation: occurs when a hand leaves the state of rest at the side of the body, passing from $d < D_{inner}$ to $D_{inner} \leq d < D_{outer}$.

Deictic Gesture Lexicon		
Word	Arguments	Purpose
Deixis	type	(preparation | point | retraction)
	h_x	x-position of head in video frame
	h_y	y-position of head in video frame
	d	distance of head to hand, in pixels, in video frame
	θ	angle between head and hand in video frame

Table 3.3: The 2D deictic gesture lexical schema.

stroke/point: occurs when a hand nears its maximal distance from the body, according to the user's reach, passing from $D_{inner} \leq d < D_{outer}$ to $D_{outer} \leq d$.

retraction: : occurs when a hand returns to a state of rest, passing from $D_{inner} \leq d < D_{outer}$ to $d < D_{inner}$.

These gestures are encoded as soon as they occur (according to d) and sent to Clavius, in the format indicated in Table 3.3. Note that a user may begin a preparation gesture, but may retract before a stroke/point gesture, if they are unsure of their target.

Unfortunately, problems including occlusion, varying illumination, false positives and poor pixel connectivity lead to somewhat unreliable tracking at the time of empirical analysis. Furthermore, geometric errors incurred by projecting vectors through inaccurate head and hand positions onto the virtual environment are compounded by the use of a second camera used to deduce 3D positions. For these reasons, the mouse tracker of §3.5.8 is currently the primary source of deictic gesture recognition.

3.5.7. Keyboard Tracker. The keyboard tracker is a drop-in replacement for the continuous speech dictation tracker (§3.5.4) in that its arguments are the same, with the same semantics. Specifically, the **POS** (part-of-speech) argument specifies the part of speech of the tokenized word, and **lex** the lexicography of the word, as shown in Table 3.4. A token is sent when the space bar is hit and a non-empty string of characters has been entered since the last space. If the POS of a word is ambiguous – for instance, the word *colour* can be either a noun or a transitive verb – then *each* possible interpretations are sent to Clavius in an n-best list, thereby circumventing the need to perform POS tagging. The scores associated with each token are meant to be the unigram prior probability of the word $P(W)$ in some corpus, although these may be initialized at the discretion of the application designer if no suitable corpus yet exists.

3.5.8. Mouse Tracker. The mouse tracker reports on two types of activity by the mouse, as shown in Figure 3.10. First, a single-point **deixis** is identified by a mouse click, and described by the position of the mouse within a window, and the status of its buttons. If the mouse is displaced by a minimum distance before a depressed button is released, then this is identified by the tracker as a drawn **area**, and is reported once the button returns to its 'up' state. The arguments of each of these lexical entries is given in Table 3.4. Every

Mouse Lexicon		
Word	Arguments	Purpose
Deixis	x	x-position in window
	y	y-position in window
	left	status of left button (up|down)
	middle	status of middle button (up|down)
	right	status of right button (up|down)
Area	x	x-position of area's left edge
	y	y-position of area's top edge
	width	pixel width of the drawn area
	height	pixel height of the drawn area

Speech/Keyboard Lexicon		
Word	Arguments	Purpose
Word	POS	part-of-speech (ex. VB, NN, ...)
	lex	the word itself (ex. *go*, *house*, ...)

Table 3.4: The mouse and speech/keyboard lexical schemas.

token it sends has a score of 1.0, and does not have multiple interpretations for the same input (n-best list of length 1).

An example of the usage of this lexicon is shown in the grammar in Appendix B.

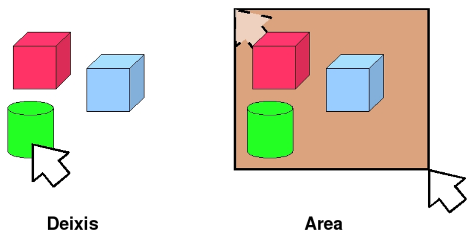


Figure 3.10: Examples of deixis and area-drawing by the mouse. Clavius – not the tracker – will identify that the cylinder is selected by the deixis and that all three objects are selected in the drawn area.

3.6. Search Space

Internally, Clavius can be defined as a pseudo-blackboard system, since it involves a number of processes that work simultaneously and asynchronously on a shared data space. Blackboard systems are typically characterized by the distributed nature of their subsystems and are especially useful in approximating solutions to problems for which no deterministic solution is known [**16**]. Similarly, Clavius was designed to be highly modular and distributed, as the eventual system was expected to require significant processing power.

All partial parse hypotheses reside in the SEARCH SPACE, which is partitioned into the following three subspaces , according to the processing threads of §3.2.2.1:

$\Xi^{[G]}$: The Generalizer's Active Subspace.

$\Xi^{[S_{Active}]}$: The Specifier's Active Subspace.

$\Xi^{[S_{Inactive}]}$: The Specifier's Inactive Subspace.

These subspaces are named according to the process that accesses their members, and whether their members can be transferred to another subspace. For example, graphs in $\Xi^{[S_{Active}]}$ are selected by the SPECIFIER thread, and then moved to $\Xi^{[S_{Inactive}]}$. Table 3.5 summarizes the permissions granted to the indicated processing threads. This distinction is further clarified in §4.1.

	Read	*Write*	*Delete*
$\Xi^{[G]}$	Generaliser, Memory Manager	Generaliser, Specifier, Trackers	Generaliser, Memory Manager
$\Xi^{[S_{Active}]}$	Specifier, Memory Manager	Generaliser	Specifier, Memory Manager
$\Xi^{[S_{Inactive}]}$	Specifier, Memory Manager, Cognition	Specifier	Memory Manager

Table 3.5: Read, write, and delete permissions granted to threads, according to subspace.

Since for more complex applications the number of partial hypotheses is likely to become very large, it becomes expedient to partition the search space further. For algorithmic reasons that will become clear in §4.1, and because the CATEGORY of a graph can be determined in $O(1)$ (because of the restrictions in §3.3.1), each subspace is implemented as a hash table [**58**] of subsets of graphs, keyed by the CATEGORY of those graphs, as exemplified in Figure 3.12. Specifically,

DEFINITION 3.6.1. *for the hash table h holding subspace $\Xi^{[A]}$ containing the set of graphs $\{\Psi_{A,1}, \Psi_{A,2}, ..., \Psi_{A,|h|}\}$, where $|h| = \left|\Xi^{[A]}\right|$ – the element $h\,(c)$, contains the subset $\{\Psi_{A,j} |\, \text{CATEGORY}(\Psi_{A,j}) = c\}$.*

3.6.1. Search Nodes. All partial parse graphs are encapsulated within container SEARCH NODES within the search space, to ensure that each graph is handled independently of, and separately from all others within the context of data structure management.

A *search node*, or *node* refers to the atomic object, σ_i that encapsulates a partial-parse graph Ψ_i. This distinction is particularly useful to abstract the mechanisms for data storage away from the linguistic mechanisms of unification parsing. Although a particular partial hypothesis and its encapsulating search node can often be referred to interchangeably, the search node also allocates additional 'blackboard' space used to facilitate the computation of scores, further discussed in Chapter 4.

3.6.2. Randomized Search Trees. The subsets that comprise each subspace can become very large themselves, especially for 'higher' grammar categories (such as sentences), and it is therefore important that they be organized to make certain operations as efficient as possible. **Treaps** are specialized binary search trees that are efficiently height-balanced. Specifically, every node, v, in the treap has both a *key* and a *priority*, and is organized with respect to other nodes such that:

- If v is in the left subtree of u, then $\text{KEY}(v) < \text{KEY}(u)$
- If v is in the right subtree of u, then $\text{KEY}(v) \geq \text{KEY}(u)$

- If v is a child of u, then $\text{PRIORITY}(v) < \text{PRIORITY}(u)$

An example treap is shown in Figure 3.11. If the priorities associated with each node $x_1, x_2, ..., x_n$ of a treap are uniform randomly distributed, then the resulting treap is height balanced at $O(\lg n)$ in expectation, as discussed in [**98**] and [**26**].

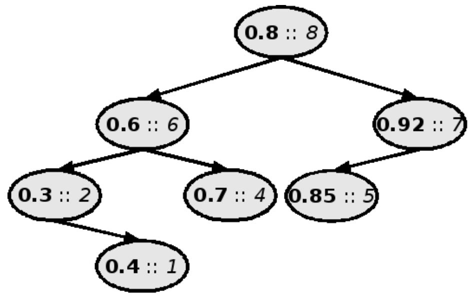


Figure 3.11: Example treap. Each SEARCH NODE is labelled $(k :: p)$, where k is the KEY of the node, and p is its PRIORITY.

The most common operations on sets of parses in Clavius are summarized in Table 3.6, along with their asymptotic running times, given treap size N. Treaps are especially efficient at these operations, given that search nodes need to be ordered. Since a pointer indicates the node with the highest priority in a treap, reverse depth-first traversal from that node effectively orders the nodes in the treap by decreasing (or non-increasing) order. Most of these operations are used heavily by the Generalizer and Specifier, although trimming a node is only performed by the memory manager, although frequently. Trimming a node refers to removing all nodes below a given node. All operations must maintain the treap properties.

Operation	**Asymptote**
Find best node	$O(1)$ (amortized)
Find the M-best nodes	$O(M)$ (expected)
Insert new node	$O(\lg N)$
Delete given node	$O(\lg N)$
Trim a given node	$O(1)$

Table 3.6: The most frequent set operations in Clavius, and their asymptotic behaviour as implemented by treaps (of size N).

Furthermore, the value pair $(key, priority)$ required of search nodes can easily be instantiated by the score and timespan of their component graphs, respectively, since these values *must* be present in all search graphs (§3.3.1). Setting $key = score$ orders the treap by score (a reverse post-fix depth first traversal lists the nodes in order of decreasing score). Setting $priority = time$ (or, at least, end-time) means that all nodes below a given node are *older* than that node. Therefore, if a given node is “too old” to be considered for a partial parse, so too are all nodes below it.

Parallel set operations such as intersection, union, and difference on treaps of size m and n have also been shown to run asymptotically in $O(m \lg(\frac{n+m}{m}))$ expected work and polylogarithmic depth [**12**]. This has potential applicability in future expansions to the internal data management of Clavius.

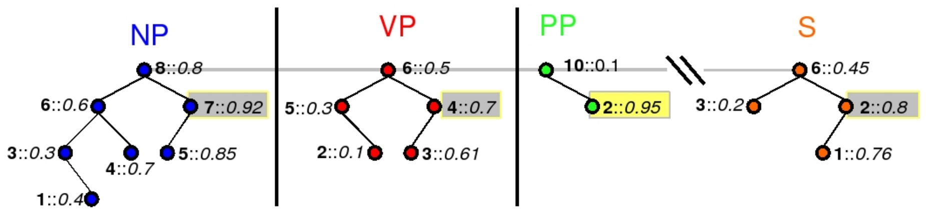


Figure 3.12: Example subspace, where each element is a treap keyed by their common category (ex. NP, VP, ...). The highest scoring node in each treap, and the best node overall, is highlighted.

3.7. Scorer and Partial Ordering

Upon being created, all partial parses are assigned a score on $\Re_{[0..1]}$ approximating their likelihood of being part of an accepted multimodal sentence. This score is effectively a linear combination of scoring modules, which are defined in mathematical detail in §4.2. The scorer software component of the Clavius architecture is essentially a static object that responds to requests of other components. For instance, the Generalizer thread may pass a graph Ψ to the Scorer to be evaluated, and the Scorer, in turn, will pass this graph to its various dynamically loaded knowledge sources for further evaluation.

3.7.1. Knowledge Sources. Each knowledge source is an independent code module designed to measure a unique aspect of the given graph Ψ according to its particular metric of multimodal language (see Chapter 4). These are compiled into independent object code modules, which are then dynamically loaded using the `dl_open` directive by the Scorer. Since the Scorer calls each source dynamically to score Ψ, each must implement the same function declarations, including initialization and scoring. The intention is to allow interaction engineers to write new scoring modules specifically for their project, without needing to change Clavius.

This is discussed in detail in §4.2.

3.8. Cognition and Application Interface

Clavius decides when to stop searching for a better sentence-level hypothesis and to simply return its best so far (see §4.1.4 for details). This essentially allows the system to trade deliberation time for more accurate results, and vice-versa. The class of algorithms having this feature is called *Anytime algorithms*, and has been used by intelligent systems in signal processing, and mobile robotics[**120**].

The **Cognition** module decides when a sentence Ψ_B represents the correct interpretation of user input and communicates this directly with the **Application** over the network. This encodes the semantic information of the sentence along with a classification of the type of sentence, constituting a dialogue act.

3.8.1. State and World Interface. The state of the world is held in a MYSQL database according to a relational database model whose structure is defined by the interaction engineer. This database is updated with any state information from the **Application** the engineer deems appropriate, and is accessed by certain knowledge sources in order to make informed decisions about the partial parses they evaluate. This is discussed in further detail in §4.1.4.

3.9. Deployment Overview

Deployments of multimodal systems are often highly application-specific [**74**], and hence are limited in their scalability and applicability in arbitrary computing environments. They are typically bisected into two main architectural subcateogires: *feature-level* deployments or "early fusion" – in which signal-level recognition in any mode affects and influences the course of recognition in the others [**82**][**84**], and *semantic-level* deployments or "late fusion" – which consists of an independent collection of N recognizers and an $N + 1^{th}$ component that unifies their output (see CUBRICON [**73**], for example). Early fusion may be more appropriate for more temporally synchronized input modes [**66**][**93**] due to the possibility to exploit codependencies early, although the late fusion approach may be more easily scaled up and trained – mainly due to the preponderance of available unimodal data. Clavius is an example of feature-level early fusion.

Clavius is a modular collection of libraries and resources that can be deployed in a number of ways, and essentially combines aspects of both feature-level and semantic-level deployments. It is grammar-based, and places almost no assumptions on the type of application or physical interface with which it is used – and is hence highly generic.

There are essentially two extremes of deployment methodology with the Clavius architecture that vary in the degree of their concurrency. In the first, all processing occurs on a single host – including all tracking and application logic, as shown in Figure 3.13. In the second model, processing threads are maximally spread over multiple machines, as shown in Figure 3.14. In the latter case, each tracker would ideally be run on a dedicated machine attached to the appropriate sensing equipment. Since the generalization and specification processes are so decoupled, multiple instances of either function can be spread over an arbitrary number of machines in this scheme – improving concurrency and hence execution time. The mutex-controlled critical sections of Clavius that provide protection to the search space does not need to be modified as the number of accessing methods scale up – hence each partial parse will only be accessed by one thread at a time.

Regardless of the deployment scheme used, all communication between Clavius and the trackers, the application, and the world database must occur over TCP/IP, even if exclusively on the local host. Furthermore, all machines must have their internal clocks synchronized to between the nearest 200μs and 10ms, which is possible through the Network Time Protocol [**87**] – an externally run service. No additional network security has yet been implemented.

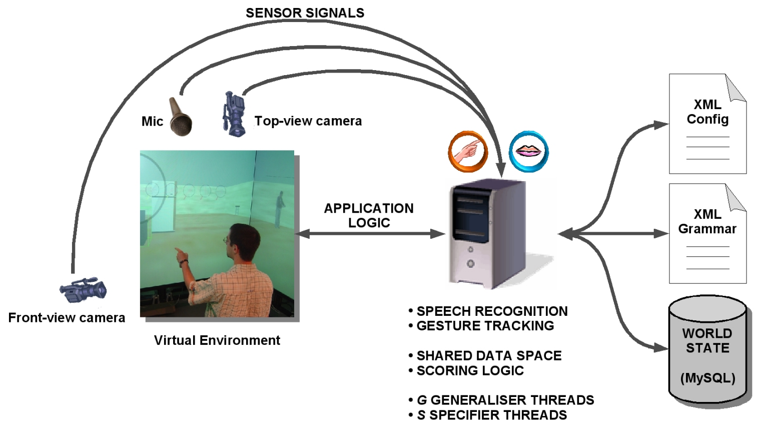


Figure 3.13: Simple deployment layout, with one central processor for all tasks and data management.

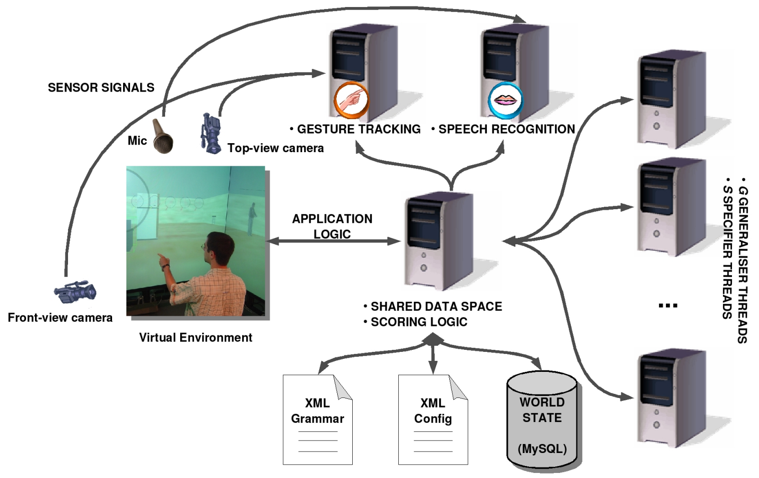


Figure 3.14: Distributed deployment layout, where one central server allocates work orders and maintains data integrity, one processor is assigned to each modality, and $D \geq 1$ *additional* processors execute an arbitrary number of Generalizer and Specifier threads on the shared space.

3.9.1. Software Engineering. The core framework was developed for the Linux platform with the version 2.6 kernel, with further deployment information outlined in §5.1.5. Clavius has undergone four phases of development, roughly a) preliminary research (spring 2005), b) development (summer 2005), c) quality assurance and bug-fixing (fall 2005), and d) experiment and analysis (winter 2006).

All code within the core of Clavius is written in C, with trackers and testbed applications written in C++ and JAVA. The final size of the code base is approximately **10900** lines of C code in the core of Clavius, with approximately **1580** lines of original JAVA code for ancillary test applications, and an additional **1631** lines of configuration, including grammars.

In Clavius, a procedural programming model was employed, although aspects of object oriented programming were accomplished via opaque types, and modular structures. To save time and management overhead, all memory allocation is performed at startup by the use of 'free lists'.

Table 3.7 lists the software components that are required to build and run Clavius.

Component	**Description**
MySQL ≥ 4.1	World state. Spatial extensions used to compare world objects.
libxml ≥ 2.6	Decoding of all network communication.
POSIX threads	Required for multithreaded architecture.
GNU sockets	Used for all communication by Clavius with components.
Network Time Protocol server	Synchronization of parser and tracker timestamps.

Table 3.7: Necessary software components for compiling and running Clavius.

CHAPTER 4

Algorithmic and Linguistic Issues

This chapter delves deeper into some of the concepts and components introduced in Chapter 3, concentrating on their algorithmic, mathematical, and linguistic consequences. First, the GS Algorithm is examined in greater detail in §4.1, concentrating on implementation details. Then, §4.2 describes the general mechanism of applying scores to partial parses, before the various approaches designed for that purpose are discussed in §4.3 - §4.12. Finally, §4.13 discusses some consequences of the GS Algorithm.

4.1. The GS Algorithm

The GS Algorithm consists of multiple concurrent, asynchronous processes operating on the search space. It was inspired initially by bottom-up multidimensional chart parsing [**49**], before issues of computational efficiency with this approach drove development towards a best-first dynamic programming strategy, such as those employed in the Viterbi and CYK algorithms [**52**].

The GS Algorithm employs a high degree of parallelism. Multiple instances of the two chief processors – Generalization and Specification – are used simultaneously to explore the search space bi-directionally, as discussed in the following subsections.

4.1.1. Multimodal Bi-Directional Parsing. Clavius' parsing strategy combines *bottom-up* and *top-down* approaches, but differs from other approaches to bi-directional chart parsing [**97**][**91**] in several key respects, introduced in the following subsections before being presented in practice in §4.1.2 and §4.1.3.

4.1.1.1. *Asynchronous Collaborating Threads.* A defining characteristic of the approach in Clavius is that graphs Ψ are selected *asynchronously* by two concurrent processing threads, rather than serially in a two-stage process. In this way, processes can be distributed across multiple machines, or dynamically prioritized. Generally, this allows for a more dynamic process where no thread can dominate the other. In typical bi-directional chart parsing the *top-down* component is only activated when the *bottom-up* component has no more legal expansions [**1**].

4.1.1.2. *Unordered Constituents.* Clavius incorporates the unconvential approach of placing no mandatory ordering constraints on the set of constituents γ_i of any rule Γ_i, hence the rule $\Gamma_{abc} : A \longrightarrow$ B C parses the input " C B". Temporal ordering can easily be maintained, however, as explained in §4.10.2.3.

4.1.1.3. *Partial Qualification.* Whereas existing bi-directional chart parsers maintain fully-qualified partial parses by incrementally adding adjacent input words to the agenda, Clavius can construct parses that instantiate only a subset of their constituents, so Γ_{abc} also parses the input "B", for example.

4.1.2. Generalization. A GENERALIZER thread monitors the best partial parse, Ψ_g, in $\Xi^{[G]}$, and creates new parses Ψ_i for all grammar rules Γ_i having CATEGORY(Ψ_g) on the right-hand side. Effectively, these new parses are instantiations of the relevant Γ_i, with one constituent unified to Ψ_g. This provides the impetus towards sentence-level parses, as simplified in Algorithm 5 and exemplified in Figure 4.1. It abstracts information in a given graph Ψ_g to 'higher-level' grammar categories, so if DEPTH$(\Psi_i) = d$, then its generalization will have depth $d + 1$.

Naturally, if rule Γ_i has more than one constituent $(c > 1)$ of type CAT(Ψ_g), then c new parses are created, each with one of these being instantiated.

Algorithm 5: GENERALIZE

```
Data: Subspaces $\Xi^{[G]}$, and $\Xi^{[S_{Active}]}$, and Grammar $\Gamma$
Result: New partial parses written to $\Xi^{[G]}$, chosen parses moved to $\Xi^{[S_{Active}]}$
while data remains in $\Xi^{[G]}$ do
    $\sigma_g :=$ Best_node $\left(\Xi^{[G]}\right)$
    $\Psi_g :=$ Graph_of $(\sigma_g)$
    foreach rule $\Gamma_i \in \Gamma$ s.t. Cat $(\Psi_g) \in$ RHS $(\Gamma_i)$ do
        $((n_{root}, \Phi_{RHS}), n_{rhs}) :=$ Get_RHSNode $(\Gamma_i)$
        foreach $j$ such that $n_{rhs}$'s $j^{th}$ outgoing arc is $\delta_j = ((n_{rhs}, \Phi_{constituent}), n_j)$, and Cat $(\Psi_g) =$ Cat $(n_j)$ do
            $\Psi_i := \left[\bullet \overrightarrow{\text{RHS}} \bullet\right]$
            for $k = 1..||$ RHS $(\Gamma_i)||$ do each constituent of $\Gamma_i$
                if $k = j$ then desired constituent
                    add $\Psi_g$ to $k^{th}$ slot of $\Psi_i$'s RHS
                else
                    add null arc to $k^{th}$ slot of $\Psi_i$'s RHS
            $\Psi_i :=$ Unify $(\Gamma_i, \Psi_i)$
            if $\exists \Psi_i$ then
                $\sigma_i :=$ Encapsulate $(\Psi_i)$
                Apply Score $(\Psi_i)$ to $\sigma_i$
                Insert $\sigma_i$ into $\Xi^{[G]}$
    Move $\sigma_g$ into $\Xi^{[S_{Active}]}$
```

Since the GENERALIZER is activated as soon as input is added to $\Xi^{[G]}$, the process is *interactive* [**108**], and therefore incorporates the associated benefits of efficiency. This is contrasted with the all-paths bottom-up strategy in GEMINI [**32**] that finds all admissible edges of the grammar.

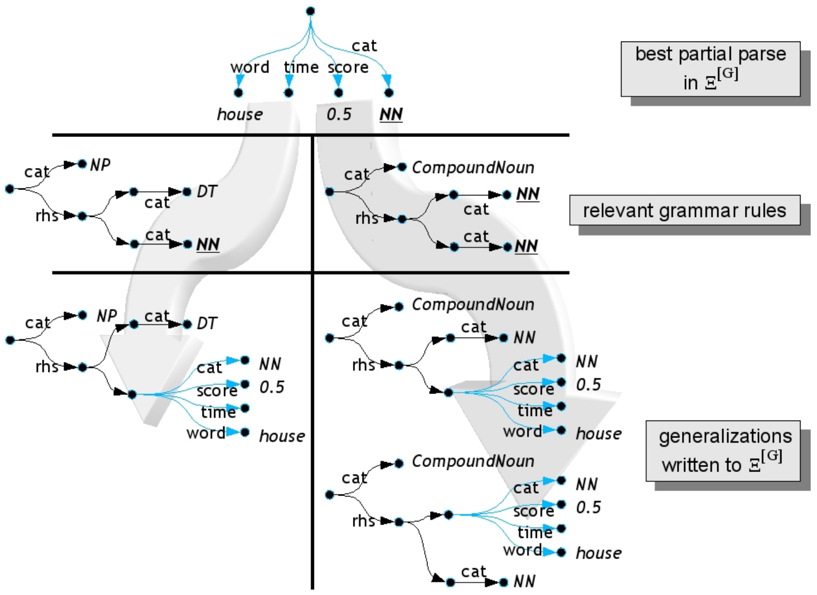

best partial parse in $\Xi^{[G]}$
relevant grammar rules
generalizations written to $\Xi^{[G]}$
word
time
score
cat
house
0.5
NN
cat
NP
rhs
DT
CompoundNoun

Figure 4.1: Example of GENERALIZATION.

4.1.3. Specification. A SPECIFIER thread provides the impetus towards complete coverage of the input, as simplified in Algorithm 6 (see Figure 4.2). It combines parses in its subspaces that have the same top-level grammar expansion but different instantiated constituents. The resulting parse merges the semantics of the two original graphs only if unification succeeds, providing a hard constraint against the combination of incongruous information. The result, Ψ, of specification *must* be written to $\Xi^{[G]}$, otherwise Ψ could never appear on the RHS of another partial parse. Consequences of this approach are discussed in §4.4 and §4.6.

Specification is necessarily commutative and will always provide more information than its constituent graphs if it does not fail, unlike the 'overlay' method of SMARTKOM [**3**], which basically provides a subsumption mechanism over background knowledge. In Clavius, the specified graph Ψ_U (Algorithm 6) subsumes both of its component parses, $\Psi_U \sqsubseteq \Psi_s$ and $\Psi_U \sqsubseteq \Psi_j$. According to the nature of subsumption, the resulting graph will always contain more information than its components – and hence cover more of the user input in this case.

4.1.4. Cognition. The COGNITION thread serves a dual role as the interface between Clavius and both the application it controls, and the world state database that they share. It is primarily a monitoring

Algorithm 6: SPECIFY

```
Data: Subspaces $\Xi^{[S_{Active}]}$, and $\Xi^{[S_{Inactive}]}$, Grammar $\Gamma$, and Beam width $W$
Result: New partial parses written to $\Xi^{[G]}$, chosen parses moved to $\Xi^{[S_{Inactive}]}$
while data remains in $\Xi^{[S_{Active}]}$ do
    $\sigma_s :=$ Best_node $\left(\Xi^{[S_{Active}]}\right)$
    $\sigma_j :=$ Best_node $\left(\Xi^{[S_{Inactive}]}\right)$
    $\Psi_s :=$ Graph_of $(\sigma_s)$
    $\Psi_j :=$ Graph_of $(\sigma_j)$
    $w := 1$
    while $\exists\Psi_j$ and $w \leq W$ do
        if ActiveConstituents $(\Psi_s) \cap$ ActiveConstituents $(\Psi_j) = \emptyset$ then
            $\Psi_U :=$ Unify $(\Psi_s, \Psi_j)$
            if $\exists\Psi_U$ then
                $\sigma_U :=$ Encapsulate $(\Psi_U)$
                Apply Score $(\Psi_U)$ to $\sigma_U$
                Insert $\sigma_U$ into $\Xi^{[G]}$
            $\sigma_j :=$ predecessor of $\sigma_j$ in $\Xi^{[S_{Inactive}]}$
            $\Psi_j :=$ Graph_of $(\sigma_j)$
            $w := w + 1$
    Move $\sigma_s$ into $\Xi^{[S_{Inactive}]}$
```

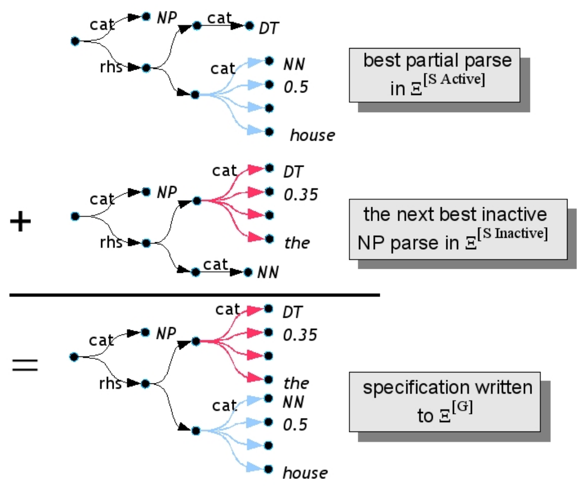

cat
NP
cat
DT
rhs
cat
NN
0.5
house
best partial parse in $\Xi^{[S\ Active]}$
+
cat
NP
cat
DT
0.35
rhs
the
cat
NN
the next best inactive NP parse in $\Xi^{[S\ Inactive]}$
=
cat
NP
cat
DT
0.35
rhs
the
cat
NN
0.5
house
specification written to $\Xi^{[G]}$

Figure 4.2: Example of SPECIFICATION.

tool, augmented with networking code. The main functions it performs are summarized in the following subsections, each of which is implemented by an interruptible thread.

4.1.4.1. *Sentence Acceptance.* The only task that the COGNITION thread must necessarily perform is the acceptance of interpretations for user input, and the transmission of this interpretation to the application that input controls.

Specifically, whenever a new *sentence-level* parse, Ψ_B is inserted into the Specifier's inactive subspace, $\Xi^{[S_{Inactive}]}$ *and* it is the highest-scoring sentence-level parse, COGNITION is signalled and begins a countdown of T μs, specified by system configuration, but typically between 2 or 3 seconds. If a new best *sentence-level* parse during this time, the countdown is restarted.

When a countdown is started, the average score, s_μ of the N *next* best sentence-level parses in $\Xi^{[S_{Inactive}]}$ is taken, for configuration parameter N. At the end of the countdown T, Ψ_B has not been superseded, but its N best predecessors might comprise new alternate hypotheses. The average of *these* scores, s'_μ is taken and compared with the original average s_μ. If $s'_\mu - s_\mu > \theta$ for some parameter θ (the difference will never be negative), this implies that the search process is improving its estimates at the 'high end', and that further deliberation may yield a better Ψ_B. If this is the case, the countdown is again restarted until the scores of the N best competitors no longer approach $\text{SCORE}(\Psi_B)$ by the minimal amount.

Once this occurs, Ψ_B is been accepted, and COGNITION sends it in the XML format of Clavius grammars (Appendix B), to the APPLICATION over the network.

4.1.4.2. *Queries.* The COGNITION module connects to and reads from the MySQL WORLD database, as specified by its configuration. All scoring modules that make use of this information send properly formatted SQL queries to COGNITION that in turn performs the database lookup, and sends the results to the issuing module. The COGNITION module also prohibits modification of this database.

4.1.4.3. *Dialogue Awareness.* If implemented, special scoring modules that make use of the flow of dialogue between the human user and their application can query COGNITION to indicate the dialogue act most recently issued by the application, and the user (assuming the previous recognition was correct). This is discussed in §4.9.

4.1.5. Memory Management. There are several reasons to perform memory management to trim the size of the search space. First, there is the obvious fact that finite memory would be exhausted as new parses accumulate in $[\Xi_{Inactive}]$, and that the resulting treaps would be excessively costly to traverse and manage. Another basis for memory management is to simply aid the Generalization and Specification processes by removing hypotheses from their consideration that are unlikely to be of consequence to them. Unlike the former, the latter rationalization implies evaluation of the partial parses to determine candidacy for removal.

Despite these grounds for the automatic reduction of the search space, Clavius does *not* reset the search spaces to their initial empty states once a sentence has been approved by the COGNITION module. Maintaining such intermediate data once a hypothesis has been accepted is an uncommon practise, but is useful for two reasons: to correct for possible mistakes, and to allow for unifications with anaphoric (§2.3.1) entities.

The memory management thread traverses all treaps in each subspace in a prefix depth-first search (nodes are examined before its children). When it encounters search node σ_m, it may do one of the following:

(i) if TIME_START(σ_m) <CURRENT_TIME$-\theta_T$ for some parameter θ_T, then σ_m and all nodes below it (by virtue of the treap property) are too old for consideration, and the whole subtreap rooted at σ_m is removed (trimmed - §3.6.2) in $O(1)$.

(ii) if the difference between the score of the best graph in the treap and SCORE(GRAPH_OF(σ_m)) is greater than some parameter θ_S, then σ_m's score may be considered to low for consideration. Therefore, the σ_m and the entire left subtreap of σ_m can be removed.

4.2. Parse Scoring and Domain-Centric Knowledge

The score of *every* non-lexical (tracker) partial parse Ψ, SCORE(Ψ), is a weighted linear combination of $|S|$ independent scoring modules, or KNOWLEDGE SOURCES:

$$\begin{aligned} Score(\Psi) &= \sum_{i=0}^{|S|} \omega_i \kappa_i(\Psi) \\ &= \omega_1\kappa_1(\Psi) + \omega_2\kappa_2(\Psi) + ...\omega_{|S|}\kappa_{|S|}(\Psi) \end{aligned} \tag{4.1}$$

Each module presents a score function $\kappa_i : \Psi \to \Re_{[0..1]}$ according to a unique criterion of multimodal language, weighted by a vector $\overrightarrow{\Omega}$, where each component ω_i is also on $\Re_{[0..1]}$. Furthermore, enforcing $\sum_{i=0}^{|S|} \omega_i = 1$ ensures that $0 \leq Score(\Psi) \leq 1$ for any Ψ, thus normalising the score space, and controlling the partial ordering of parses.

The criteria of multimodal language explored in this model are outlined in §4.3 - §4.12. In Clavius, these criteria generally represent a particular and precise aspect of multimodal communication, and individually define a partial ordering over the set of all Ψ. Their co-representation in this form is meant to provide a transparent and extensible mechanism in which new discoveries about MMI could be easily added to the system, and one in which disparate approaches can be combined.

Apart from the obvious priority scheme existing between the modules via their weights, all κ_i are considered to be independent from each other. This implies that the partial ordering of Ψ (according to grammar Γ) by any κ_j does not effect the ordering by κ_k, $k \neq j$. Data localization and abstraction has ensured that this is pragmatically the case within the current implementation of Clavius, however §5.3 describes how criteria can nevertheless sometimes operate against one another, despite having similar goals by 'preferring' divergent parse classes.

Although the assumption of fixed linear independence between criteria is common in practise, it is not necessarily theoretically optimal. For instance, researchers at IBM have demonstrated that weights automatically assigned to audio and visual components of MMI based on audio-stream fidelity (the degree of voicing in the audio) at run-time improves accuracy by 7% in 'clean' speech and by 26% in 'noisy speech'(noise at 8.5dB SNR)[**37**] when compared to the audio-only WER. They also show another technique, more similar to Clavius, where audio and visual streams are independently trained offline, and have their relative weights jointly trained to minimize WER multimodally, achieving 5% relative improvement over N-best re-scoring. This approach is similar to other weighting schemes for audio-visual fusion in speech recognition [**17**] [**111**], with characteristic results.

Finally, values for the set $\overrightarrow{\Omega}$ can currently be trained *off-line* using supervised MLE on acquired training data annotated with desired parses, as explained in §5.1.7.1. This model does not readily allow the weights to be trained or updated *on-line* during user interaction, although incremental automatic weight update mechanisms could be incorporated in the future.

4.2.1. Hard Constraints. A side effect of only using $\kappa_i : \Psi \to \Re_{[0..1]}$ is that it precludes scoring modules from explicitly forbidding partial parses from consideration. If some graph Ψ represents some impossible parse, and only a single κ_i recognizes this, returning a score of 0 still allows for a $\text{SCORE}(\Psi) > 0$ if $w_j, kappa_j(\Psi) > 0$ for at least one $j \neq i$.

Therefore, some scoring modules implement 'hard constraints' that expressly forbid unification outright, returning $\kappa_i(\Psi) = -\infty$ in those exceptional cases, therefore $\text{SCORE}(\Psi) = -\infty$, and Ψ is not added to the search space, so all parses under consideration still have a score in $\Re_{[0..1]}$.

An example of hard constraints in practise is given in §4.4.

4.2.2. Module Parametrization. Scoring modules can accept externally-specified parameters that control their behaviour. This is of primary importance, as it allows training each knowledge source according to the specific domain of their criterion. These may include parameters to Gaussians, thresholds or HMM transition weights, for example. When modules take externally-specified arguments, these are differentiated by their superscript, such as $\kappa_i^{(\sigma)}$, and are introduced below where appropriate.

4.3. Temporal Alignment, κ_1

A pronounced correlation in time exists between co-referring speech and gestural events, notably between demonstrative pronominals and deixis. The κ_1 module, therefore, partially orders parses Ψ whose constituents cover similar timespans in proximity and scope. A stochastic mechanism is used to measure the similarity of timespans, where each timespan [$\text{TIME\_START}(\Psi)$,$\text{TIME\_END}(\Psi)$] is modelled with a Gaussian $N(\mu, \sigma^2)$, as shown in Figure 4.3, such that

$$\mu = \frac{\text{TIME\_START}(\Psi) + \text{TIME\_END}(\Psi)}{2}$$

and

$$\sigma^2 = \begin{cases} \kappa_1^{(\sigma_{def})} & \text{if TIME\_START}(\Psi) = \text{TIME\_END}(\Psi), \\ \kappa_1^{(c)}(\text{TIME\_END}(\Psi) - \text{TIME\_START}(\Psi)) & \text{otherwise.} \end{cases}$$

where the chief parameterization is specified by $\kappa_1^{(c)}$. Timespans are modelled by Gaussians in this way so that the similarity amongst both their lengths and locations can be simultaneously compared, while also using a smooth nonlinear function to account for possible measurement error in timing.

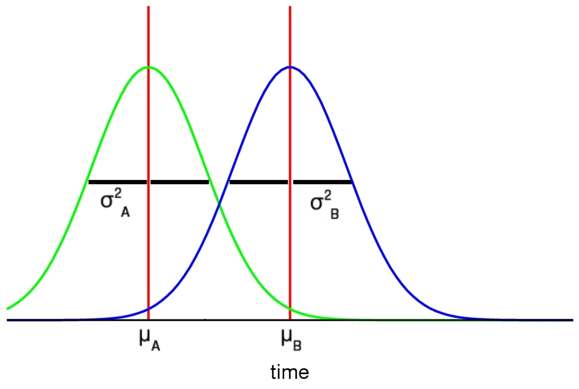


Figure 4.3: Two gaussian curves $N_A\left(\mu_A, \sigma_A^2\right)$ and $N_B\left(\mu_B, \sigma_B^2\right)$ representing timespans for constituent parses Ψ_A and Ψ_B, respectively.

The *Kullback-Leibler distance*, or *Relative Entropy*, is a frequently used information-theoretic distance measure between two continuous probability densities on $x \in X$, p and q [**67**]:

$$KL\left(p \| q\right) = \int p(x) \log \frac{p(x)}{q(x)} dx \tag{4.2}$$

However, equation 4.2 is non-symmetric as κ_1 requires. A symmetric version, for Gaussian distributions $N_A\left(\mu_A, \sigma_A^2\right)$ and $N_B\left(\mu_B, \sigma_B^2\right)$, whose derivation is given in §A.1.3, obeying the triangle inequality used in κ_1 is:

$$\begin{aligned} SKL\left(N_A \| N_B\right) &= SKL\left(N_B \| N_A\right) \\ &= \frac{(\sigma_1^2 - \sigma_2^2)^2 + ((\mu_1 - \mu_2)(\sigma_1^2 + \sigma_2^2))^2}{4\sigma_1^2\sigma_2^2} \end{aligned} \tag{4.3}$$

Unlike the amount of overlap between N_A and N_B, $SKL\left(N_A \| N_B\right)$ can be computed very quickly (in $O(1)$). Since $SKL : N_A \times N_B \rightarrow \Re_{[0..\infty]}$ and is a measure of distance, $e^{-SKL(N_A \| N_B)}$ is used as a measure of similarity on $\Re_{[0..1]}$ where identical timespans have maximal score:

$$\begin{aligned} e^{-SKL(N_A \| N_A)} &= e^{-0} \\ &= 1 \end{aligned}$$

Therefore, $\kappa_1(\Psi)$ is the average similarity between all of Ψ's *existing* constituents, $\text{RHS}(\Psi)$:

$$\kappa_1(\Psi) = \frac{\sum\limits_{\substack{\Psi_A,\, \Psi_b \,\in\, \text{RHS}(\Psi) \\ \Psi_A \neq \Psi_B}} e^{-SKL(N_A \| N_B)}}{\text{NUM\_PAIRS}(\text{RHS}(\Psi))}, \tag{4.4}$$

where $N_i\left(\mu_i, \sigma_i^2\right)$ is parametrized by the timespans in Ψ_i, as described above. The result is that phrases with co-referring cross-modal utterances are more highly scored. Further consequences are discussed in §5.2.2.

4.4. Common Ancestry Constraint, κ_2

A consequence of accepting n-best lexical hypotheses for each word is the risk that two competing hypotheses for the same input might be unified into a single phrase during SPECIFICATION. That is, if Ψ_A and Ψ_B represent alternatives for the same input, no partial parse Ψ should exist such that $\Psi_A \leq \Psi \wedge \Psi_B \leq \Psi$.

The κ_2 module expressly prohibits such simultaneous substructures by providing a simple *hard-constraint* (§4.2.1). Metadata encoded in each word-level graph Ψ by the trackers indicate a unique ID, $k(\Psi)$ for each n-best list. If Ψ_A and Ψ_B are lexical alternatives, then $k(\Psi_A) = k(\Psi_B)$. When Ψ is generalized, creating parse Ψ', then $k(\Psi') = k(\Psi)$.

Given this metadata, the κ_2 module can compute $\{k(\Psi_A)\} \cap \{k(\Psi_B)\}$, and therefore

$$\kappa_2(\Psi) = \begin{cases} 1.0 & \text{if calling thread is GENERALIZER}, \\ 1.0 & \forall \Psi_A \neq \Psi_B \in \text{RHS}(\Psi), \{k(\Psi_A)\} \cap \{k(\Psi_B)\} = \emptyset, \\ -\infty & \text{otherwise.} \end{cases} \tag{4.5}$$

.

When $\kappa_2(\Psi) = -infty$, then even if $\omega_2 = 0$, $Score(\Psi) = -\infty$ and is not added to the search space. When graphs Ψ_X and Ψ_Y are specified to form graph Ψ, this module encodes in the latter the metadata $k(\Psi) = \{\{k(\Psi_A)\} \cup \{k(\Psi_B)\}\}$, allowing for the comparison in Equation 4.5 to properly prohibit illegal parses.

4.5. Probabilistic Declarative Grammar, κ_3

More common grammatical constructions are emphasized by transforming the multimodal grammar into a probabilistic one according to the Probabilistic Context-Free Grammar (PCFG) formalism. Specifically, given grammar rules $\Gamma_{i,j} : N^{(j)} \leftarrow \zeta^{(i)}$, for sequences of terminals and non terminals, $\zeta^{(i)}$, each is augmented with a probability, $P(\Gamma_{i,j})$ approximating its likelihood of being used in a parse, under the constrain that:

$$\forall i \sum_j P\left(N^{(j)} \leftarrow \zeta^{(i)}\right) = 1 \tag{4.6}$$

That is, the probabilities of all rules with $N^{(j)}$ on the LHS sum to 1. The value $\kappa_3(\Psi)$ is the product of the probabilities of all rules used in a parse:

$$\kappa_3(\Psi) = P(\Gamma(\Psi)) \cdot \prod_{\Psi_c \in \text{RHS}(\Psi)} \kappa_3(\Psi_c) \tag{4.7}$$

, where $\Gamma(\Psi)$ represents the top-level Γ_i expansion in Ψ. The κ_3 module requires that Clavius XML grammars be augmented with features $\Phi = P_\Gamma$ with floating point values representing the associated probabilities. Currently, these values are linear, although logarithmic representation may be used instead to avoid numerical underrun errors.

Probabilities applied to individual features Φ of graphs have been used in exponential probabilistic models [**48**], but not yet in Clavius.

4.5.1. Independence Assumptions. There are certain assumptions implicit in κ_3's probabilistic grammar model. Namely, given the parse Ψ_A with constituent substructures Ψ_i and Ψ_j ($\Psi_i \neq \Psi_j$):

Place invariance: $\kappa_3(\Psi_i)$ is independent of Ψ_i occurring before or after Ψ_j.

Ancestor-free: $\kappa_3(\Psi_i)$ and $\kappa_3(\Psi_j)$ are independent of their use in the production of Ψ_A.

Context-free: The rule expanding Ψ_A from Ψ_i and Ψ_j expands to a unique nonterminal $N^{(A)}$ in the grammar.

In this model, the probability of a sentence $w_{1m} = w_1\, w_2\, \ldots w_m$, is $P(w_{1m}) = \sum_{\Psi} \kappa_3(\Psi)$ for all Ψ that cover w_{1m}, which might be grammatically ambiguous.

4.5.2. Training. Training the probabilities of a PCFG can be done in two ways, depending on the data. For unannotated data, the Inside-Outside algorithm can be used to generate a new grammar Γ representative of the input set [**67**]. This approach is not well-suited to Clavius, as it will typically not be able to include semantic constraints, for instance. Therefore, all $P(\Gamma_{i,j})$ are trained according to maximum likelihood estimation on annotated training data, as discussed in Chapter 5.

4.6. Input Coverage, κ_4

For reasons discussed in §5.3, it is necessary to explicitly apply a preference for longer sentences – or those that cover as much of the input as possible. This can be accomplished easily by a $\Re_{[0..1]}$ function of the number of words in a given parse Ψ. Specifically,

$$\kappa_4(\Psi) = \frac{2}{1 + e^{-\kappa_4^{(c)} \textsc{NumberOfWordsIn}(\Psi)}} - 1 \tag{4.8}$$

The function NUMBEROFWORDSIN can be computed in $O(1)$ given metadata similar to that used in §4.4. The parameter $\kappa_4^{(c)}$ can be instantiated according to the expected length of sentences used in an application, as shown in Figure 4.4, although this does *not* change the ordering of parses by κ_4 – just the range of its contribution to the overall $Score$. For instance, smaller values of $\kappa_4^{(c)}$ may be used for longer, more conversational sentences, and larger $\kappa_4^{(c)}$ for shorter command-driven sentences.

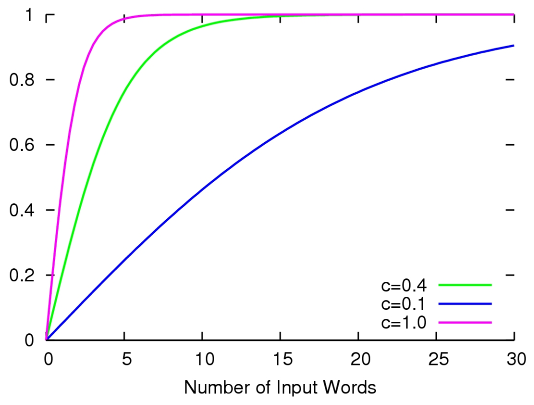


Figure 4.4: Different curves of κ_4 given varying parameters $\kappa_4^{(c)}$.

This approach is similar to the effect of the reduce-reduce conflict resolution in GEMINI [**32**], in that parses covering more input are preferable to shorter ones, and are examined first.

4.7. Information Content, κ_5

Many problems in NLP can be cast as classification problems where semantic or statistical context predicts membership in linguistic classes. Important problems of relevance to Clavius, such as part-of-speech tagging, PP-attachment, parsing and categorization have been approached via *Information Theory* as an extension to regular conditionally probabilistic models[**89**] [**9**]. The use of information theory is especially applicable to Clavius, since the feature structures it uses are inherently amenable to semantic evaluation.

The κ_5 scoring module uses a maximum joint entropy model to combine diverse pieces of contextual evidence in estimating the probability of linguistic classes co-occurring within a given linguistic context. In

Clavius, this is useful in maximizing the coverage of the input early in the search process, in conjunction with κ_4 §4.6.

As previously discussed in §3.3.1, all sentence-level graphs must have a single arc with feature **content** encapsulating high-level semantic variables. The κ_6 module associates probabilities $P(\Phi_S \rightarrow v)$ for all (Φ_S, v) pairs of features Φ_S and values v in subgraphs rooted by **content** in a data set. For example, $P(\text{action} \rightarrow \text{delete})$ reflects the proportion of $\Phi_S = \text{action}$ features having the value $v = \text{delete}$ in a data set. In the same way, multivariate probabilities $P(\Phi_x \rightarrow v, \Phi_y \rightarrow w)$ are also computed for pairs of co-occurring feature-value pairs.

For X and Y being the sets of **content** feature-value pairs in Ψ_X and Ψ_Y, respectively – the joint mutual information between these parses is:

$$I(\Psi_X;\Psi_Y) = -\sum_{(\Phi_x \rightarrow v_x)\in X}\ \sum_{(\Phi_y \rightarrow v_y)\in Y} P\left(\Phi_x \rightarrow v_x, \Phi_y \rightarrow v_y\right) \lg \frac{P(\Phi_x \rightarrow v_x)P(\Phi_y \rightarrow v_y)}{P\left(\Phi_x \rightarrow v_x, \Phi_y \rightarrow v_y\right)} \tag{4.9}$$

where $I(\Psi_X;\Psi_Y) = I(\Psi_Y;\Psi_X)$ [**67**]. This metric is useful only during the SPECIFICATION step, where a shifted sigmoid (with parameter $\kappa_6^{(c)}$) similar to the one used in §4.6, gives:

$$\kappa_5(\Psi) = \begin{cases} 1.0 & \text{if calling thread is GENERALIZER,} \\ \frac{2}{1+e^{-\kappa_5^{(c)} I(\Psi_X;\Psi_Y)}} - 1 & \text{for SPECIFIER called for graphs } \Psi_X \text{ and } \Psi_Y \end{cases} \tag{4.10}$$

.

Since $I(\Psi_X;\Psi_Y)$ is a measure of how many bits would be needed to encode the combination of Ψ_X and Ψ_Y, κ_6 therefore promotes parses that maximize information across constituents.

Consequences are discussed in §5.3.

4.8. Multimodal Language Model, κ_6

There has not been very much research extending language models to multiple dimensions[1]. In Clavius, the κ_6 module uses a bigram probability table according to the lexica associated with a given grammar. For a grammar with M modes ($m_1, m_2, \ldots, m_M$), where the i^{th} mode has $||m_i||$ lexical entries, a 'word' $\vec{w}$ is represented by an M-dimensional vector, where the j^{th} component is either one of the lexical entries in m_j, or λ – the 'empty' (null) string. The interpretation of $\vec{w} = \langle w_1, w_2, \ldots, w_M \rangle$ is that over a non-zero period of time, each component w_j co-occurs with the others.

For example, Figure 4.5 shows how κ_6 breaks down a partial parse for the phrase "paint $\searrow$ orange" into 7 2-dimensional 'word' segments and Table 4.1 defines these segments.

[1] See §A.2.4 for more on language modelling.

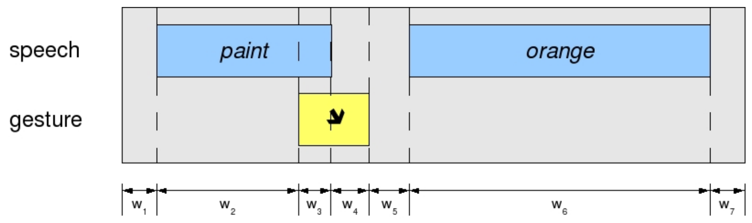


Figure 4.5: Example 7-word breakdown of "paint $\searrow$ orange", under κ_6 representation.

$\vec{w_1}$	$\vec{w_2}$	$\vec{w_3}$	$\vec{w_4}$	$\vec{w_5}$	$\vec{w_6}$	$\vec{w_7}$
$\langle\lambda,\lambda\rangle$	$\langle paint,\lambda\rangle$	$\langle paint,\searrow\rangle$	$\langle\lambda,\searrow\rangle$	$\langle\lambda,\lambda\rangle$	$\langle orange,\lambda\rangle$	$\langle\lambda,\lambda\rangle$

Table 4.1: Two dimensional representation of multimodal segments in Figure 4.5

Given this new segmented representation of the random variables, the task of language modelling in Clavius becomes trivial. For each instantiation of the interpreter, an $N \times N$ matrix is constructed, where

$$N = \prod_{i=1}^{M} (||m_i|| + 1) \tag{4.11}$$

Each row and column of this table represents an M-dimensional vector for a specific permutation of the symbols in all modes. Cell (i, j) of this matrix is $P(\vec{w_i}|\vec{w_j})$ – the probability that a multimodal 'word' ($\vec{w_i}$) immediately follows another ($\vec{w_j}$). Initial training of these probabilities naturally follows standard techniques of n-gram learning on finite data, although the increased complexity of additional dimensions may limit its effect if data is scarce. In this case, additional adjustment can be accomplished by discriminative training methods [**96**] [**61**], discussed in §A.2.4.

Therefore, given a partial parse Ψ covering (ordered) word sequence $< s > \ \vec{w_1}\,\vec{w_2} \ldots \vec{w_n} \ < /s >$ (for sentence-start and -end tags $< s >$ and $< /s >$, respectively,

$$\kappa_6(\Psi) = P(\vec{w_1}| < s >) \cdot P(< /s > |\vec{w_n}) \cdot \sum_{i=2}^{n} P(\vec{w_i}|\vec{w_{i-1}}) \tag{4.12}$$

This product can be converted to a sum of logarithms to avoid numerical underruns.

4.9. Dialogue Model, κ_7

The κ_7 scoring module in $Clavius$ is tightly coupled to the grammar and is based on the assumption of a command-driven dialogue between the human user and their application. The interaction engineer adds the feature <DIALOGUE_TAG> to all sentence-level rules in their grammar, with values indicating the type of dialogue event they represent, such as those indicated in Table 4.2.

Forward Types	**Example**
assertion	*"I always fly first class"*
action-directive	*"Book me a flight $\searrow$ here"*
info-request	*"Who lives in $\searrow$ that castle?"*
offer	*"I can meet you at 3"*
commit	*"I'll come to your party"*
conventional-opening	*"Hello"*
conventional-close	*"Goodbye"*

Backward Types	**Example**
accept	*"Yes, I can meet you at that time."*
signal-nonunderstanding	*"What did you say?"*
acknowledgement	*"OK"*
answer	*"Nobody lives there."*

Table 4.2: Subset of dialogue events types, from [**45**].

The application may respond and participate actively in the dialogue, producing its own dialogue events. The κ_7 module uses this information, and procures it from the application via the COGNITION thread (§4.1.4). Given that the user's previous utterance was recognized as being of type τ_{i-2}, and that the application's previous utterance was of type τ_{i-1}, then

$$\kappa_7(\Psi) = \begin{cases} 0.0 & \text{if } \Psi \text{ is not a sentence-level parse,} \\ P(\tau_i | \tau_{i-2}\ \tau_{i-1}) & \text{for } \Psi \text{ being of dialogue event of type } \tau_i \end{cases} \tag{4.13}$$

.

This is a state-based metric similar to Hidden Markov methods, where the conditional probability is trainable via maximum likelihood. This module is, of course, designed for conversational agents, from which appropriate data would be more readily procured.

4.10. Functional Constraints, κ_8

The phrase *"pour the glass into the water"* has no meaning, nor is the multimodal interpretation *"remove this $\searrow$ triangle"* likely if the pointing gesture, $\searrow$, indicates a square, say, or empty space. Constraints that prohibit or inhibit these types of phrases cannot always be expressed declaratively, and depend instead on dynamic state or common-sense information.

Associating functional constraints with grammar rules has been done in HPSG and other unification grammar formalisms [**118**]. Specifically, grammar rule Γ_i can be associated with set of functions $F_i = \{f_{i,1}\ f_{i,2}\ \ldots f_{i,n}\}$ by the interaction engineer, where each function in F_i evaluates to a score on $\Re_{[0..1]}$, and are combined by a parenthesized structure of terms (Table 4.3). Each function can take parameters consisting of nodes in the DAG that implements Γ_i (Figure 4.7).

Essentially, these functions are instantiations of a λ-calculus whose purpose is to help guide the search process by evaluating any aspects of the world or dialogue that cannot be declared statically in the grammar. This evaluation includes the application of hard constraints, common-sense knowledge (§4.10.2.4) and other semantic comparison (§4.10.2.3). The functions are executed at run-time ("applicative expressions") by the κ_8 module when scoring the graph Ψ that incorporates them.

4.10.1. Instantiation and Definition. The configuration parameters to the κ_8 module include the location and name of the object code implementing the functions in the grammar (ex. `\$CODEBASE/constraints.o`). This object code must be compiled with shared, position-independent and dynamically loadable flags.

Inclusion of the κ_8 module necessitates that a ¡CONSTRAINTS¿ arc be present in the grammar for each rule, and forbids execution if it is not present. If there are no constraint functions associated with Γ_i, then this node will be empty. If constraint functions exist, they must be specified in the format provided in Table 4.3.

$$\begin{array}{l|l}
\text{constraint} \longleftarrow \text{term} \mid \emptyset & \text{fcn} \longleftarrow \text{fname (PARAMS)} \\
\text{term} \longleftarrow \text{(term)} & \text{term} \longleftarrow \text{fcn} \\
\text{term} \longleftarrow \text{term + term} & \text{term} \longleftarrow \text{term * term} \\
\multicolumn{2}{l}{\text{PARAMS} \longleftarrow \emptyset \mid \$p \mid \hat{o} \mid \text{PARAMS , PARAMS}}
\end{array}$$

Table 4.3: The description language for constraint functions, including functions (fcn), terms, and parameterization (PARAMS).

The two operators, $+$ and $*$ are soft-addition and multiplication respectively. Namely, $+$ returns the summation of the scores of its operands, or 1.0, if this sum exceeds 1.0, while $*$ returns the product of the scores of its operands. In this way, the values of all terms are constrained to be on $\Re_{[0..1]}$, and the operators become essentially soft logical operators, approximating $\vee$ and $\wedge$ in Boolean arithmetic. Parentheses have the normal semantics and execution precedence over the operators. $\kappa_8(\Psi)$, therefore, returns the $\Re_{[0..1]}$ evaluation of its highest term upon execution of its functions.

Figure 4.6 provides an example of this in practise.

```
...

<params>
    \$dx <dx> 100 </dx>
    \$dy <dy>  42 </dy>
</params>
<constraints>
      <![CDATA[
             (f_1( \$x_1, \$y_1, \$dx, \$dy ) + f_2( \$x_2, \$y_2 ))
            *
             f_3( \$x_3, ^o_1 )
         ]]>
</constraints>

...
```

Figure 4.6: Example use of functional constraints in the Clavius XML grammar format.

Incorporation of the functional constraints into the DAG implementing the grammar rules is exemplified in Figure 4.7. Parameters to each function are always re-entrant nodes whose values are defined elsewhere in the graph structure, hence these nodes may be roots of complex substructures.

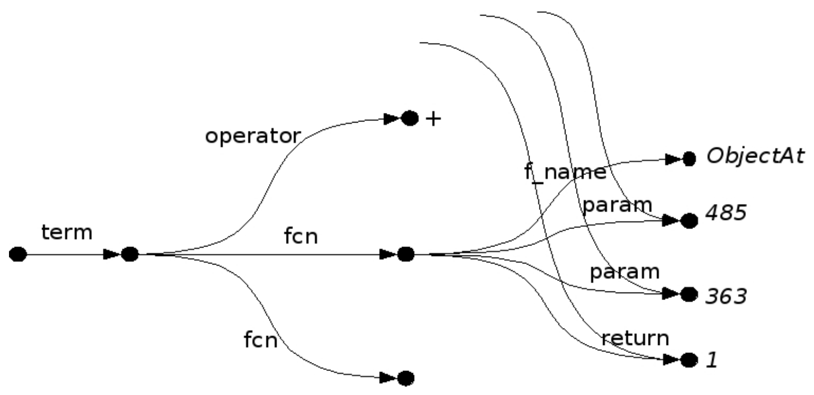


Figure 4.7: Example of graphical incorporation of the functional constraint OBJECTAT, with parameters instantiated by re-entrant nodes.

4.10.2. Aspects of Functional Constraints. Technical details of κ_8 are discussed in the following subsections.

4.10.2.1. *Delayed Execution.* Functional parameters may only become available midway through the search process. For example, a function f parametrized by the location of a deixis event, but associated with a grammar rule Γ_i that has not yet been unified with a deixis parse cannot be executed until those parameters become instantiated. In these cases a 'default' score $\kappa_8^{(\lambda)}$ can be returned by f until its parameters become available. This approach is similar to that in Ait-Kaci and Nagr [**2**].

4.10.2.2. *Unique Execution.* Once a κ_8 function is executed, its return value is encoded as the value of its rooting node. This is a notable exception to regular DAG semantics (§2.1.1), but is useful in avoiding repeated calculations and increasing efficiency, since the parameters of an executable function will not change through unification. That is, if function f has been executed for graph Ψ, and Ψ is later generalized to Ψ^G, or specified to Ψ^S, the new fs in Ψ^G and Ψ^S will not be re-evaluated.

4.10.2.3. *Temporal Constituent Ordering.* Certain linguistic entities must occur in a particular temporal order in order to be evaluated – such as the arguments of a transitive verb, for instance. Clavius, however, does not require such temporal ordering. An important function of κ_8 is to be able to enforce these sorts of constraints through temporal constraint functions that compare the times of occurrence of parse constituents.

Table 4.4 shows relations between time intervals, and the functions that implement them, as inspired by Nilsson [**75**]. For example, the basic relation

$(\forall x, y)\,[\text{T\_MEETS}(x, y) \equiv (\text{T\_END}(x) = \text{T\_START}(y)]$

is easily defined. Since the time spanned by each parse is stored in a predefined location, all these functions can be computed in $O(1)$.

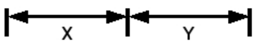
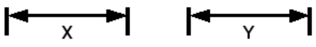
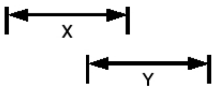
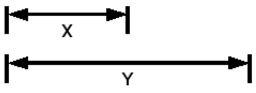
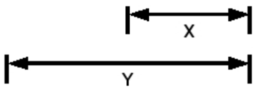
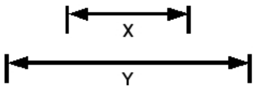
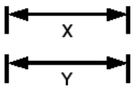
Relation	Function, relative to X	Function, relative to Y
	T_MEETS(X, Y)	T_MET_BY(Y, X)
	T_BEFORE(X, Y)	T_AFTER(Y, X)
	T_OVERLAPS(X, Y)	T_OVERLAPPED_BY(Y, X)
	T_BEGINS(X, Y)	T_BEGINNED_BY(Y, X)
	T_ENDS(X, Y)	T_ENDED_BY(Y, X)
	T_DURING(X, Y)	T_COTAINS(Y, X)
	T_EQUALS(X, Y)	T_EQUALS(Y, X)

Table 4.4: Relations between time intervals, and functions returning 1.0 in each case, and 0.0 otherwise.

4.10.2.4. *Return values.* Constraint functions in κ_8 can also be used to modify parameters that are passed 'by reference' within the graph and in this way can incorporate new semantic knowledge into parse graphs. For example, in Figure 4.7, the 'return' value of 1 has been written to the indicated node by the OBJECTAT function. In this particular case, the OBJECTAT function takes (ordered) parameters 485 and 363 as the (x, y) co-ordinates of a deixis event on a screen. It then queries the WORLD database (via COGNITION) for the nearest object to those co-ordinates, writes the identification number of this object to the graph, and returns a score relative to the distance between $(x, y) = (485, 363)$ and the centroid of this object[2].

Clavius also supports multiple hypotheses to be returned in this way from a κ_8 constraint function. For instance, if the (x, y) co-ordinates of a deixis falls between n objects, then an OBJECTSAT function embedded in graph Ψ can force Clavius to replicate Ψ into graphs $\Psi_1\ \Psi_2\ \ldots\ \Psi_n$, each with a unique object id 'returned' OBJECTSAT, and each with a score reflecting that object's distance to the deixis.

4.11. Wordnet Ontology, κ_9

For more complex grammars, especially those that allow unconstrained conversational agents, lexical relationships may improve the coverage of the grammar. Instead of specifying a particular lexicography as being a required word in a grammar, the interaction engineer can specify a *synset*, or *synonymy set* of words.

[2]In practise, this is computed by a bivariate Gaussian centred at the indicated object.

For instance, by implementing a grammar rule Γ_i with a word w replaced by $\{w\}$ (encased in curly braces), then any word synonymous with w will be acceptable to unify with Γ_i.

WORDNET [**34**] is a free lexical database and reference system relating English words – nouns, verbs, adjectives and adverbs, in particular – into synonym sets by lexical concept. Moreover, WORDNET encodes the semantic 'distance' between words representing how divergent they are in meaning on a granular scale $1..D_i$ (for word w_i). κ_9 therefore is the average inverse distance between all words w_i in a parse Ψ, and the 'target' words w_i' in the grammar rules constructing it, scaled on the average scale size D_i'.

Since gestural lexica are generally fairly flat and sparse, these relations have little applicability outside of the primary mode – speech – and even then will only generally become useful as the grammar size increases to conversational systems, which has not yet been explored in Clavius. Due to its exclusion in later experiments, however – this metric remains only partially implemented.

4.12. Prosodic Information, κ_{10}

Prosodic information can be useful in predicting the presence of co-occurring events across modes referring to the same multimodal phrase (§2.2.2). The tracker discussed in §3.5.5 is used to discover sudden, sharp shifts in pitch via the F_0 contour and produces 'words' w_i that effectively simply notify the time of these shifts.

The $\kappa_1 0$ module promotes partial parses that have prosodic information, where this acoustic pitch shift occurs in the proximity of lexical events in at least two different modes. The two nearest words to this prosodic shift in time are used to compute $\kappa_1 0$ if they come from different modes, according to a Gaussian distance metric:

(4.14)

$$\kappa_{10}(\Psi) = \begin{cases} 0.0 & \text{if there does not exist any prosodic tokens } w_i \text{ in } \Psi, \\ 0.0 & \text{if a prosodic token } w_i \text{ occurs only in a unimodal parse,} \\ \frac{e^{\frac{-(t_1-t_p)^2}{2(\kappa_1 0^{(c)})^2}} + e^{\frac{-(t_2-t_p)^2}{2(\kappa_1 0^{(c)})^2}}}{\kappa_1 0^{(c)}\sqrt{8\pi}} & \text{for shift centred at time } t_p\text{, disparate-mode events centred at times } t_1 \text{ and } t_p. \end{cases}$$

.

This module depends on assumptions based on the literature of such shifts indicating correlation of multimodal events[**57**], but in a strictly unimodal utterance $\kappa_1 0$ will have no effect on the ordering of partial parses.

4.13. Consequences of the GS Algorithm

Certain consequences of Clavius, including the assumptions lifted (§4.1.1) and constraint methodologies applied, have unexpected solutions, as described in the following subsections.

4.13.1. Infinite recursion. In Clavius, infinite recursion manifests itself as expansions in the grammar where the GENERALIZER selects Ψ such that $\textsc{Cat}(\Psi) = A$, and a rule $\Gamma_A : A \rightarrow \alpha A \beta$, for some possibly null sequences of non terminals α and β. The new graph produced is in turn generalizable by the same rule, Γ_A – and this process recurses infinitely, producing the infinite graph $\Psi' : A \xrightarrow{*} \alpha \Psi \beta$.

This is potentially a serious drawback, since methods usually used to combat this sort of problem – such as transforming immediate left-recursive grammars into purely right-recursive ones [**69**], or into terminal-first Greibach normal form (GNF) [**44**] have no effect since Clavius makes no distinction between the order of its constituents [3]. In fact, unification grammars for speech recognition have had only mixed success in dealing with this sort of problem [**31**], and infinite recursion continues to be a serious problem for some of the most modern parsers – such as those using the JSpeech Grammar Format (JSGF) [**113**].

In Clavius this problem is kept under control to some extent by the use of the probabilistic grammar constraint κ_3 (§4.5). For any set of grammar rules $G = \{\Gamma_A\}$ for $A = 1..|G|$ such that $\forall A, 0 < P(\Gamma_A) < 1$, any large arithmetic product will tend to 0. In theory, one would expect the scorer would eventually 'throw away' any Ψ encapsulating a large number of grammar rules, say $|G| >$. While it is true that more densely packed parses result from κ_3 – it does not necessarily provide a hard constraint against infinite recursion. Namely, even if $\kappa_3(\Psi)$ goes to 0, the parse is not thrown out. The probabilistic grammar constraint, then, may be changed to return $-\infty$ for very low probabilities.

4.13.2. Complete Coverage. Any legitimate parsing algorithm must expand every possible grammatical construction, given any legal grammar and any legal set of input. This is obvious, but also vital so the proof of this capacity in GS Algorithm is relevant. That is,

THEOREM 4.13.1. *Given an empty search space, and an infinite timeout ($T = \infty$) on the search process (the search is never interrupted by* COGNITION*), the GS Algorithm will produce every possible partial hypothesis spanning input (tracker) graphs $I = \{\Psi_1, \Psi_2, \ldots \Psi_n\}$, given grammar $\Gamma = \{\Gamma_1, \Gamma_2, \ldots \Gamma_g\}$.*

PROOF. If there are undiscoverable parses in the GS Algorithm, given Γ and I, then there *must* be at least one 'lowest' undiscoverable parse, Ψ_x such that each of its constituent partial parses (call this set C) is either been discoverable by the GS Algorithm, or is in I. Each of the parses in C are subject to Generalization, since all graphs begin their 'lifespan' in its subspace. For each $\Psi_i \in C$, a generalization unifying Ψ_i to the RHS of Ψ_x's top-level grammar rule, Γ_x will be created since Ψ_i's top-level grammar rule Γ_i must be on the RHS of Γ_x. As these generalizations accumulate in $\Xi^{[S_{Inactive}]}$, a process of $|C|$ specifications will incrementally unify these generalizaions at the root. The resulting parse consists of Γ_x as the top-level rule, with each of C unified to the RHS. This is exactly the parse Ψ_x. Since Ψ_x is discoverable

[3]These approaches also tend to drastically expand the size of the grammar – hindering performance

by the GS Algorithm, there is no 'lowest' undiscoverable parse, hence no undiscoverable parse given Γ and I. $\square$

Naturally, a benefit of the GS Algorithm's best-first strategy is that it can quickly expand its ideal case, and not consider extraneous parses.

4.13.3. Unique Expansion. Duplicate nodes are search nodes σ_i and σ_j in the search space such that $\sigma_i \neq \sigma_j$, but $\textsc{GraphOf}(\sigma_i) = \Psi_i = \Psi_j = \textsc{GraphOf}(\sigma_j)$. The presense of duplicate nodes would result in the slowdown of the GS Algorithm due to redundant processing of Ψ_i and Ψ_j in the best case, and complete deadlock in the worst, if the number of duplicate nodes becomes prohibitive.

The condition that $\textsc{ActiveConstituents}(\Psi_i) \cap \textsc{ActiveConstituents}(\Psi_j) = \emptyset$ was put into place to avoid a vulnerability in the GS Algorithm. Without this condition for specification, a single graph could be created via duplicate paths, such as:

(i) (a) Given $\Psi_1 : (_{NP}\ (_{DT}\ )\ (_{NN}\ house))$ and $\Psi_2 : (_{NP}\ (_{DT}\ the)\ (_{NN}\ ))$.
  (b) Generalize Ψ_1, giving $\Psi_3 : (_{S}\ (_{VP}\ )\ (_{NP}\ (_{DT}\ )\ (_{NN}\ house)))$.
  (c) Generalize Ψ_2, giving $\Psi_4 : (_{S}\ (_{VP}\ )\ (_{NP}\ (_{DT}\ the)\ (_{NN}\ )))$.
  (d) Specify Ψ_3 and Ψ_4, giving $\Psi_5 : (_{S}\ (_{VP}\ )\ (_{NP}\ (_{DT}\ the)\ (_{NN}\ house)))$.

(ii) (a) Given $\Psi_1 : (_{NP}\ (_{DT}\ )\ (_{NN}\ house))$ and $\Psi_2 : (_{NP}\ (_{DT}\ the)\ (_{NN}\ ))$.
  (b) Specify Ψ_1 and Ψ_2, giving $\Psi_3 : (_{NP}\ (_{DT}\ the)\ (_{NN}\ house))$.
  (c) Generalize Ψ_3, giving $\Psi_4 : (_{S}\ (_{VP}\ )\ (_{NP}\ (_{DT}\ the)\ (_{NN}\ house)))$.

Both processes 1 and 2 can occur in a single run of the GS Algorithm, giving identical graphs Ψ_5 in the first, and Ψ_4 in the second. The condition $\textsc{ActiveConstituents}(\Psi_i) \cap \textsc{ActiveConstituents}(\Psi_j) = \emptyset$ prohibits step d in process 1 above from occurring, because graphs Ψ_3 and Ψ_4 have the same active constituent, NP.

This condition does not change the results of complete coverage (§4.13.2).

CHAPTER 5

Experiments and Analysis

A suite of experiments has been designed, implemented, and conducted in order to evaluate the Clavius framework, and to make predictions on future courses of action.

This chapter begins by describing the experimental process in terms of the data collection, training and testing software, and the methodology employed (§5.1). It then presents the empirical results according to various criteria (§5.2) and the effects of training before concluding with a discussion of the implied consequences to linguistics and HCI (§5.3).

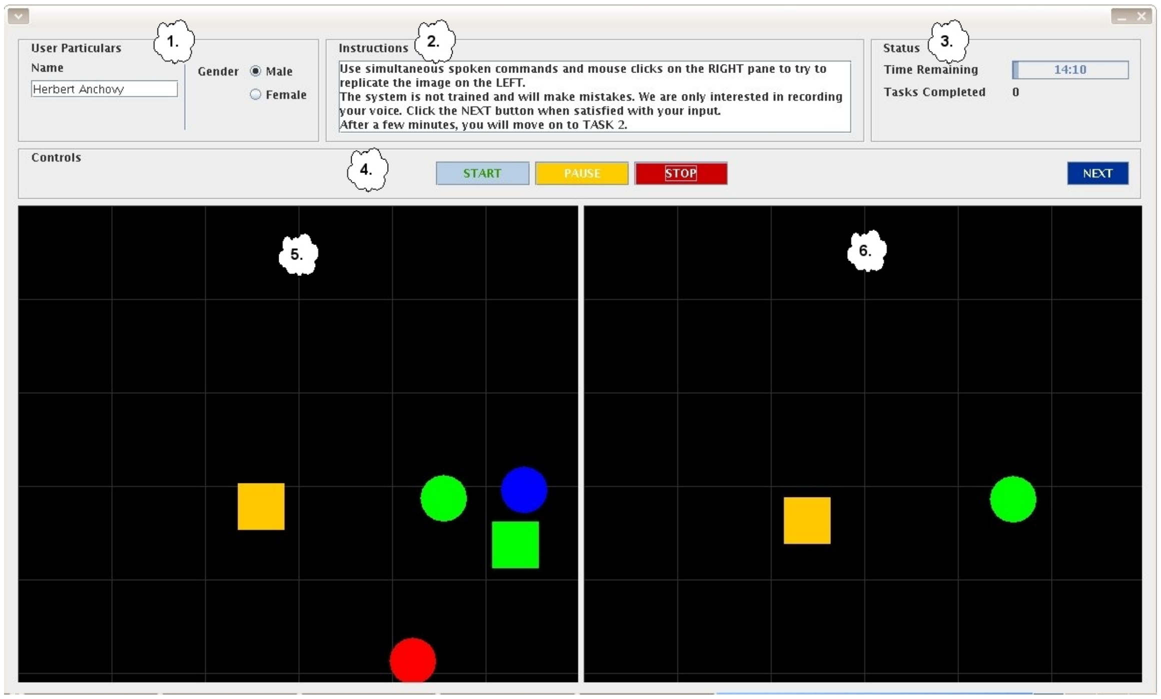
5.1. Experimental Setup

Over the course of approximately one week, multimodal data was collected through the use of a simple instantiated deployment of the Clavius architecture in order to develop an empirical and qualitative analysis of the system, and of multimodal language. The following subsections describe the collection process.

5.1.1. Task Description. A graphical software interface called REPLICATOR, shown in Figure 5.1, was developed and served as the portal through which multimodal data was collected. Once some minimal user information is recorded (1), the user is shown two panes: the *Target* pane (on the left (5)) and the *Current* pane (on the right (6)) where both panes are populated by a collection of randomly positioned coloured shapes. The user's *task* is to use spoken commands and mouse gestures in the *Current* pane in order to replicate the arrangement of shapes in the *Target* pane (see §3.1 for a similar interaction schema).

Table 5.1 summarizes the actions available to the user in REPLICATOR, with the associated effects. User utterances only affect the *Current* pane – the *Target* pane does not change during the performance of a task. The language is further described in §5.1.3.

Objects come in three shapes: *square*, *circle* and *triangle*, and six colours: *red*, *orange*, *yellow*, *green*, *blue*, and *violet*.



Use simultaneous spoken commands and mouse clicks on the RIGHT pane to try to replicate the image on the LEFT.
The system is not trained and will make mistakes. We are only interested in recording your voice. Click the NEXT button when satisfied with your input.
After a few minutes, you will move on to TASK 2.

Figure 5.1: Example state of REPLICATOR tool showing 1) user particulars, 2) instructions, 3) system status, 4) the control panel, 5) the target pane, and 6) the current pane.

Action	Effect	Parameters	Example phrase
Move	relocates one or more objects	OBJ, LOC	"*put* ↘ *this* ↘ *there*"
Colour	changes the colour of one or more objects	OBJ, COL	"*colour these* ↘ ↘ ↘ *squares red*"
Transform	changes the shape of one or more objects	OBJ, SHAPE	"*turn the circles into triangles*"
Delete	removes one or more objects	OBJ	"*delete all objects*"
Create	places a new object at a specified target	LOC, COL+SHAPE (optional)	"*put a green circle here* ↘"

Table 5.1: Action types available in REPLICATOR.

There is no software mechanism that determines when the configuration in both panes are satisfactorily identical and the task, therefore, complete. Rather, the user decides when to move on to the next *task* by clicking the NEXT button on the control panel (4), at which point both panes are filled with new random configurations of objects. This allows for more meaningful data to be collected – as the purpose of the tool is not to measure the users' competence in this application domain.

The REPLICATOR data collection tool was implemented in Java 1.5.

5.1.2. Trackers and Artificial Noise. In order to keep the data from being corrupted by potential problems in video tracking (§3.5.6), a probabilistic mouse-based approximator was used to imitate a more ideal video-based arm tracker. The mouse tracker of §3.5.8 was therefore modified to produce 'pepper'-noise in addition to its regular feedback. Specifically:

- upon a mouse click, graphs $\Psi_i : i = 1...n$ are produced for n selected uniform randomly on $1 \leq n \leq 4$. All Ψ_i are given the timestamp of the mouse click. $\Psi_1 \ \overrightarrow{\mathrm{x}}$ and $\Psi_1 \ \overrightarrow{\mathrm{y}}$ are instantiated according to the precise (x, y) co-ordinates of the mouse click and is hence the 'correct' hypothesis for the user's activity. For $\Psi_2 \ldots \Psi_n$, these variables are instantiated according to an un-correlated (bivariate) Gaussian distribution centred at the mouse click (x, y). Hypotheses $\Psi_2 \ldots \Psi_n$ simulate possible errant hypotheses of an imperfect tracker.
- at random intervals, item 1 is executed regardless of user activity, except the current (x, y) co-ordinates are used to instantiate the variables, and n is only selected randomly on $1 \leq n \leq 3$. This is done to simulate false positives in video tracking, and the value of n is arbitrary.

In this way 'errors' in precision (1) and robustness (2) in tracking could be simulated while ensuring that recall is 100% (all actual clicks are correctly encoded). Mouse clicks are represented by the grammar type DEIXIS, and a delimited area drawn with the mouse by the type AREA, as discussed in §3.5.8, and shown in Appendix B. The latter type of user action is *not* augmented with artificial noise.

5.1.3. Language and Configuration. The mouse and speech trackers were integrated directly into REPLICATOR, but communicated with Clavius on different threads. The speech tracker was a modified CMU Sphinx Engine with the Wall Street Journal-trained acoustic model (512 FFT points, 16K sample rate, frequency range 130-6800Hz, with 40 Mel filters).

The language in REPLICATOR involves five types of utterances, for each of the command types in Table 5.1. Each of these differ chiefly from each other in terms of the verb structure used. Important phrases in the grammar are the OBJECT_REFERENCE and LOCATION non-terminals that include unimodal and multimodal references to groups of objects and areas, respectively. Appendix B presents the important rules of the grammar used in these experiments.

Memory Management was deactivated for these experiments, and the minimum score necessary to add graphs to the search space was set very low (**0.05**).

5.1.4. Recording Mechanism and Data Transcription. All mouse tracker activity was recorded to specially formatted log files, specified by time, location, and status of the mouse. All speech activity was recorded to 44.1 kHz wave files by the Sphinx engine from a head-mounted microphone worn by the participants. During the data collection process, Sphinx was used to both record these files, and perform real-time recognition. Since it frequently made utterance-level segmentation errors, all wave files had to be concatenated and re-segmented "by ear" using WaveSurfer [**107**]. This process is akin to sentence-demarcation in the construction of text corpora and was done here so that utterances could be treated atomically in order to measure the sentence-level accuracy of the system.

A directory was created for each user, in which their wave files and log files were recorded, contributing to data set Δ . A data index/transcription file was then created such that each datum Δ_i is listed by the ID of the user, a reference to a *wav file ($\Delta_i \triangleright \Lambda$), indices of gesture events tied to the multimodal utterance and a transcription of the speech. Wavesurfer was also used to segment wave files into a series of time-demarcated words ($\Delta_i \triangleright \tau$), and to transcribe them into their correct interpretations.

5.1.5. Test Platform. All software was run on an AMD Sempron 3000+ laptop at 1.8 GHz with 768 MB RAM, 128 MB of which is shared with an ATI Radeon XPRESS 200M video card with 2D and 3D video acceleration. The operating system used was SUSE Linux 10.0 with the 2.6.13 kernel. The ALSA [1] driver was version 1.0.9b for microphone input, and the video driver was the standard Xorg 'radeon' driver that ships with the 6.8.2 version of X11. The indicated ALSA drivers required an abnormal configuration in order to accept input on the indicated platform.

5.1.6. Participants. Ten human subjects were recruited for the purposes of data collection. Of these, **3** were female, and **8** were between the ages of 21 and 28 (the remaining **2** were in their mid-50s). All users use computers daily as part of their work or studies and were therefore familiar with normal WIMP interfaces. Each user was paid $4CDN for 15 minutes of their time, although some volunteered more data.

Each user was provided with a preparatory form to read before commencing that described the physical setup of the environment. It was emphasized that that there was no "desirable" behaviour on their part in their interaction with the system, nor were example phrases suggested – in order to elicit more spontaneous phrase types.

5.1.7. Analysis Methodology. The primary source of empirical analysis derives from observing changes in accuracy rates as each scoring module κ_i is reparametrized individually, as well as through the joint weighting of all ω_i together. Recall that a particular set $\omega_i \rightarrow \Re$ mappings for each ω_i associated with a module is represented by the vector $\overrightarrow{\Omega}$.

Four scoring modules were not included in basic system analysis. Specifically, κ_6 (multimodal language model) and κ_9 (wordnet ontology) were omitted because to really take advantage of either of these modules would require much more complex vocabularies on the order of several thousand entries. κ_7 (dialogue modelling) was also omitted because a) the interaction schema is strictly unidirectional command-driven and b) there is no justification to assume that any command is a function of the previous n commands, since the tasks described in 5.1.1 are essentially uniform random. Finally, κ_{10} (prosody) is not part of these experiments, due to discrepancies in the autocorrelation method on which it depends.

[1]Advanced Linux Sound Architecture

The methodology used operates under the assumption that the effects of each scoring module are independent of all others used, therefore the parameters of each can be trained in isolation. The overall process is decomposed according to the three steps described in the following three subsections.

5.1.7.1. 1^{st} *joint reparametrization of weights.* Initially, parameters $\kappa_i^{(\rho)}$ are hand-tuned for each scoring module. Hand-tuned parameters in this context can be considered pseudo-random, since they were not derived from any theoretical or data-driven motivation. Five initial distributions of scoring weights $\overrightarrow{\Omega}_i$ are also determined in this way, as shown in Table 5.2. Although in all cases $\omega_2 = 0.0$, κ_2 (ancestry constraint) is still active, providing a 'hard constraint', as discussed previously. Measures of accuracy and performance (see §A.2.5) are recorded using each of these $\overrightarrow{\Omega}_i$.

Config	ω_1	$\omega_2^{(*)}$	ω_3	ω_4	ω_5	ω_8
$\overrightarrow{\Omega}_1$	0.4	0.0	0.3	0.1	0.1	0.1
$\overrightarrow{\Omega}_2$	0.2	0.0	0.1	0.3	0.2	0.2
$\overrightarrow{\Omega}_3$	0.1	0.0	0.3	0.3	0.15	0.15
$\overrightarrow{\Omega}_4$	0.2	0.0	0.2	0.2	0.1	0.3
$\overrightarrow{\Omega}_5$	0.1	0.0	0.1	0.1	0.3	0.4

Table 5.2: Five weight configurations for ω_i in the scoring function.

The first phase of training involves the joint reparametrization of $\overrightarrow{\Omega}_i$ for $i = 1..5$ using a simple single-layer feed-forward neural network as shown in Figure 5.2. The weights in this network are initialized to $\overrightarrow{\Omega}_i$ and are adjusted according to standard error backpropagation (§A.1.2). For each datum Δ_i in the training set, the wave file and gesture events are played to the speech and mouse trackers via 'control' scripts, and the resulting interpretation Ψ produced by Clavius is compared against the human transcription. If the interpretation is correct, the desired output of the training network is 1.0, otherwise it is 0.0, with inputs set to the values of $\kappa_i(\Psi)$.

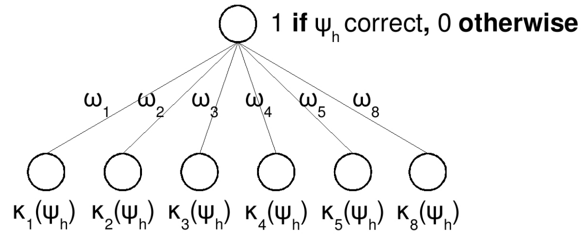


Figure 5.2: Two-layer neural network used in reparameterization of scoring weights.

Accuracy and performance are again recorded at the end of this process, given $\overrightarrow{\Omega}_i'$. In all cases, analysis was performed on a random selection of **230** test samples, with the remaining **2074** being used for training.

Training occurs in stages where each element of the training data set is presented to the network. Training is halted when 5 subsequent stages have not improved the training set accuracy by at least 1% absolutely. Overfitting was not taken into account.

5.1.7.2. *Individual training of each module, κ_i.* Once $\overrightarrow{\Omega}$ has been optimized for the pseudo-random module parameters $\kappa_i^{(\rho)}$, these are then individually tuned according to appropriate techniques, discussed in detail where appropriate in §5.2.2.

5.1.7.3. 2^{nd} *joint reparameterization of weights.* The second round of weight reparametrization essentially repeats §5.1.7.1, except individual module parameters have been adjusted according to §5.1.7.2, and the weights between them, $\overrightarrow{\Omega}$ remain unchanged since their optimization in §5.1.7.1.

5.2. Empirical Analysis

In total, the 2.5 hours of recorded speech and gesture data was partitioned into 2304 multimodal utterances. This averages to only 3.91 seconds per utterance, which is partially explained by the simplicity of the phrases used as discussed in §5.3.2. Table 5.3 shows a breakdown of *all* utterances (training and test sets) by the type of action. A small number of utterances conjoined action types, as discussed in §5.3.4. These proportions are more indicative of the general strategy used with REPLICATOR than generally applicable, as they indicate only that within the test application users tended to delete objects and create new ones, rather than transforming those that already were present in the 'current' canvas. This phenomena is partially discussed in §5.3.2.

action	total occurrences	test set occurrences
Move	493 (21.4%)	53 (23%)
Colour	458 (19.9%)	51 (22.2%)
Transform	82 (3.6%)	9 (3.9%)
Delete	674 (29.3%)	66 (28.7%)
Create	529 (23.0%)	53 (23.0%)
Move + Colour	45 (2.0%)	n/a
Create + Colour	23 (1.0%)	n/a

Table 5.3: Partition of utterances by action type, where conjunctions combined different action types 3% of the time (these were included separately in the test set).

Table 5.4 shows a breakdown of how multimodality was used, and in what proportion, according to grammar rule. Certain actions necessitate certain grammar types to be used (movement requires an OBJECT_REFERENCE, for instance), but many utterances made reference to several objects, for example. Most unimodal phrases were speech-only, although some were mouse-only (ex. "$\searrow$ $\searrow$" recognized as a **move**).

Clearly, references to location are almost invariably multimodal, although references to objects were unimodal roughly 17.9% of the time. This may indicate that users find vocalizing object properties easier than

grammar type	multimodal expression	occurrences (% within group/% total)
OBJECT_REFERENCE	unimodal (ex. "*the red square*")	343 (17.9% /11.3%)
	speech+click (ex. "↘*this square*")	1474 (76.8% /48.6%)
	speech+multiclicks (ex. "*these* ↘ ↘ ↘")	102 (5.3% /3.4%)
LOCATION	unimodal (ex. "↘")	18 (1.6% /0.6%)
	speech+click (ex. "*here* ↘")	1093 (98.4% /36.1%)

Table 5.4: Usage of multimodality, according to grammar rule.

specifying precise location. This is contrasted with other work that suggests that users interact multimodally with object references only 50% of the time [**102**]. The preponderance of multimodal rather than unimodal reference to location in the data is a confirmation that humans make frequent use of gestures to complement physical descriptions in space, as introduced in Chapter 1.

5.2.1. First Round Reparameterization. The initial accuracy of randomly initialized configurations of Clavius are compared against a 'baseline' performance of an unoptimized Sphinx system (using a unimodal grammar projection) in Figure 5.3. In all cases, accuracy rate is the proportion of sentences whose semantics are correctly interpreted. Although this is not a word-based measurement (word insertions and deletions are not counted, for example), it turns out that determiners are the only words whose presense or absense are irrelevant to the semantic interpretation of a sentence.

Ignoring function words such as determiners does not affect the semantic meaning of the utterances in these experiments.

Phrases that conjoined action types A and B for $A \neq B$ were counted twice.

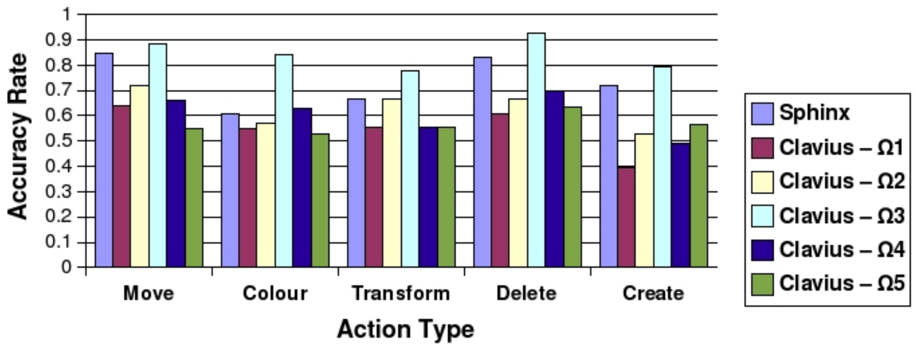


Figure 5.3: Initial accuracy rates for unimodal Sphinx versus random initial weight configurations, partitioned by action type.

The overall accuracy rate of unimodal Sphinx was **71%**, which was **22%** better absolutely than the worst initial Clavius configuration ($\overrightarrow{\Omega}_1$) and just **6%** worse absolutely than the best ($\overrightarrow{\Omega}_3$). The best recognized action type was **delete** with an average recognition rate of **74%**, and **create** the worst at **60%**, on average. These results are summarized in Table 5.5. Actions invoking colour and shape fared worst, with over **65%** of their errors being replacement errors [2] on the colour adjective.

On the training set during weight updates, $\overrightarrow{\Omega}_1$ converged to $\overrightarrow{\Omega}'_1$ in 12 iterations (+20%), $\overrightarrow{\Omega}_2$ to $\overrightarrow{\Omega}'_2$ in 10 iterations (+17%), $\overrightarrow{\Omega}_3$ to $\overrightarrow{\Omega}'_3$ in 9 iterations (+13%), $\overrightarrow{\Omega}_4$ to $\overrightarrow{\Omega}'_4$ in 11 iterations (+16%), and $\overrightarrow{\Omega}_5$ to $\overrightarrow{\Omega}'_5$ in 12 iterations (+19%), as shown in Figure 5.4. After reparametrization, each of the 5 new weight sets were presented with test data. On this set, $\overrightarrow{\Omega}'_1$ improved on $\overrightarrow{\Omega}_1$ by 7.8% RER (§A.2.5), $\overrightarrow{\Omega}'_2$ improved on $\overrightarrow{\Omega}_2$ by 11.4% RER, $\overrightarrow{\Omega}'_4$ improved on $\overrightarrow{\Omega}_4$ by 14.6% RER, and $\overrightarrow{\Omega}'_5$ improved on $\overrightarrow{\Omega}_5$ by 15.6% RER, but $\overrightarrow{\Omega}'_3$ performed worse than the initial estimate by -4.3% RER.

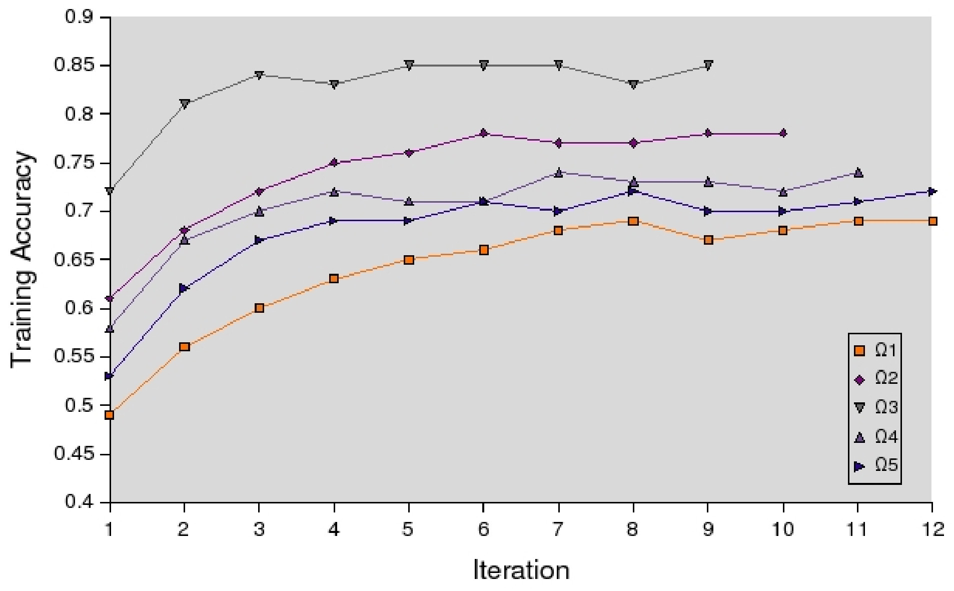


Figure 5.4: Motion of accuracy on training data, for each set of weights, over iterations of the training data set Δ.

These results show that, in general, training results in more accurate recognition rates on both the training and test sets, although training convergence appears slightly shallow and quick, and relative error reduction is lower than expected, given the low initial recognition rates.

5.2.2. Accuracy Maximization on κ_1, κ_3,κ_4 and κ_5. Of the trainable scoring modules used, temporal alignment (κ_1), input coverage (κ_4) and entropy (κ_5) are specified by parameters $\kappa_i^{(c)}$ controlling their

[2]See 'Precision' in §A.2.5.

behaviour. Each of these is updated sequentially and in isolation from the others according to an iterative steepest-ascent (hill-climbing) metric of the overall accuracy rate on the training data, as a function of the $\kappa_i^{(c)}$ in question.

Specifically, when updating $\kappa_i^{(c)}$, all values of $\kappa_j^{(c)}$ for $j \neq i$ are kept constant. The accuracy rate with $\kappa_i^{(c)}$ is compared against accuracy rates for $\kappa_i^{(c)} + \epsilon$ and $\kappa_i^{(c)} - \epsilon$ for some small ϵ. If the accuracy of either of these neighbours is higher than that of $\kappa_i^{(c)}$, then $\kappa_i^{(c)}$ is set to its best performing neighbour and the process is repeated. If $\kappa_i^{(c)}$ results in a better accuracy than its neighbours, then $\epsilon \leftarrow \frac{\epsilon}{2}$. This process is repeated as long as $\epsilon > 0.01$, which is a constant threshold across all modules.

Naturally, this process leads to local maxima, which may be prevalent in the accuracy rate functions under consideration. Final parametrizations were found to be $\kappa_1^{(c)} = 2.17$, $\kappa_4^{(c)} = 0.32$, and $\kappa_5^{(c)} = 0.8$. Note that $\kappa_4^{(c)} = 0.32$ appears to favour *discrimination* between somewhat longer sentences, falling between the two lower curves of Figure 4.4. The Gaussians resultant from training temporal alignment were also much wider than expected ($\kappa_1^{(c)}$ acts as a multiplier on standard deviation).

5.2.2.1. *Maximum Likelihood on Grammar Rules (κ_3).* The probabilistic grammar constraint (κ_3) is parametrized by the probabilities of each rule in a given grammar. Specifically, to estimate $P(\Gamma_{i,j})$ for grammar rule $\Gamma_{i,j} : N^{(j)} \leftarrow \zeta^{(i)}$, one counts the number of occurrences of $\Gamma_{i,j}$ in the training data, and divide by the number of occurrences of $N^{(j)}$. This can be accomplished in a single step and is numerically stable with regards to the data.

Within the context of Clavius, κ_3 promotes the processing of new input words and shallower parse trees. Effects of these reparametrizations on the overall accuracy are summarized in Figure 5.5.

5.2.3. Second Round Reparametrization. Overall accuracy increased again after a second round of parametrizations on individually trained modules, as expected. A summary of the accuracy rates for each $\overrightarrow{\Omega}_i$ set over the four stages of training is provided in Table 5.5, along with a comparison of the best and worst relative error reductions according to each task type as measured against performance of the baseline Sphinx system. Weighted average refers to the total performance of the indicated configuration and not the arithmetic mean of the 5 task scores.

Table 5.6 shows the final weight configurations for each ω_i in the scoring function after the second reparametrization. These weights did not change as drastically as one might expect and they certainly did not converge towards an apparent stable point. A possible explanation for this behaviour is put forward in §5.3.

The *change* in relative error reduction provides insight into the effects of learning on Clavius, where error reduction is always measured relative to a single baseline, the performance of Sphinx. Figure 5.5 shows this change, and emphasizes that the primary source of error reduction appears to come from reparametrizing weights $\overrightarrow{\Omega}_i$, rather than from individual training of modules. In fact, error levels for the five sets *increased*

System		Move	Colour	Transform	Delete	Create	Weighted Average
Sphinx		0.85	0.61	0.67	0.83	0.72	0.76
Initial	$\overrightarrow{\Omega}_1$	0.64	0.55	0.56	0.61	0.40	0.56
	$\overrightarrow{\Omega}_2$	0.72	0.57	0.67	0.67	0.53	0.63
	$\overrightarrow{\Omega}_3$	0.89	0.84	0.78	0.92	0.79	0.87
	$\overrightarrow{\Omega}_4$	0.72	0.67	0.56	0.74	0.55	0.67
	$\overrightarrow{\Omega}_5$	0.58	0.59	0.56	0.68	0.62	0.63
	Best RER	0.25	0.60	0.33	0.55	0.27	n/a
	Worst RER	-1.75	-0.15	-0.33	-1.36	-1.13	n/a
First Reparametrization	$\overrightarrow{\Omega}'_1$	0.70	0.61	0.56	0.68	0.42	0.61
	$\overrightarrow{\Omega}'_2$	0.79	0.65	0.67	0.71	0.57	0.69
	$\overrightarrow{\Omega}'_3$	0.89	0.82	0.78	0.80	0.89	0.85
	$\overrightarrow{\Omega}'_4$	0.77	0.71	0.67	0.82	0.62	0.74
	$\overrightarrow{\Omega}'_5$	0.64	0.69	0.67	0.74	0.70	0.70
	Best RER	0.25	0.55	0.33	-0.09	0.6	n/a
	Worst RER	-1.38	0.00	-0.33	-0.91	-1.07	n/a
Tuned κ	$\overrightarrow{\Omega}'_1$	0.68	0.61	0.67	0.68	0.43	0.61
	$\overrightarrow{\Omega}'_2$	0.77	0.63	0.67	0.71	0.53	0.67
	$\overrightarrow{\Omega}'_3$	0.89	0.84	0.78	0.80	0.81	0.84
	$\overrightarrow{\Omega}'_4$	0.75	0.71	0.67	0.77	0.64	0.73
	$\overrightarrow{\Omega}'_5$	0.68	0.67	0.67	0.73	0.66	0.69
	Best RER	0.25	0.6	0.33	-0.18	0.33	n/a
	Worst RER	-1.13	0.0	0.0	-0.91	-1.0	n/a
Second Reparametrization	$\overrightarrow{\Omega}''_1$	0.70	0.65	0.67	0.70	0.47	0.64
	$\overrightarrow{\Omega}''_2$	0.79	0.69	0.67	0.76	0.60	0.72
	$\overrightarrow{\Omega}''_3$	0.87	0.84	0.78	0.83	0.89	0.86
	$\overrightarrow{\Omega}''_4$	0.79	0.75	0.67	0.82	0.62	0.75
	$\overrightarrow{\Omega}''_5$	0.66	0.71	0.67	0.74	0.68	0.70
	Best RER	0.13	0.6	0.33	0.00	0.60	n/a
	Worst RER	-1.25	0.1	0.00	-0.82	-0.87	n/a

Table 5.5: Accuracy rates for Sphinx and Clavius across the four stages of training of the latter, according to various parametrizations, partitioned according to task.

during individual training, although this may also be an issue of local maxima on non-smooth accuracy rate functions. This might imply that scoring modules are more co-dependent than initially theorized.

5.3. Investigative Analysis

The degree to which the joint reparametrization of scoring weights appears to be preferable to individual training of scoring modules (§5.2.3) as a means of improving performance is surprising, and highlights a possibly problematic phenomenon in Clavius, namely, correlation phenomena between scoring modules κ_i. These modules are not necessarily independent, due to common conditioning variables in the parses they co-evaluate.

Config	ω_1	$\omega_2^{(*)}$	ω_3	ω_4	ω_5	ω_8
$\overrightarrow{\Omega}_1''$	0.31	0.0	0.27	0.16	0.20	0.06
$\overrightarrow{\Omega}_2''$	0.19	0.0	0.16	0.27	0.17	0.21
$\overrightarrow{\Omega}_3''$	0.10	0.0	0.30	0.30	0.18	0.12
$\overrightarrow{\Omega}_4''$	0.21	0.0	0.17	0.22	0.14	0.26
$\overrightarrow{\Omega}_5''$	0.16	0.0	0.11	0.12	0.26	0.35

Table 5.6: Five *final* weight configurations for ω_i after reparametrizations.

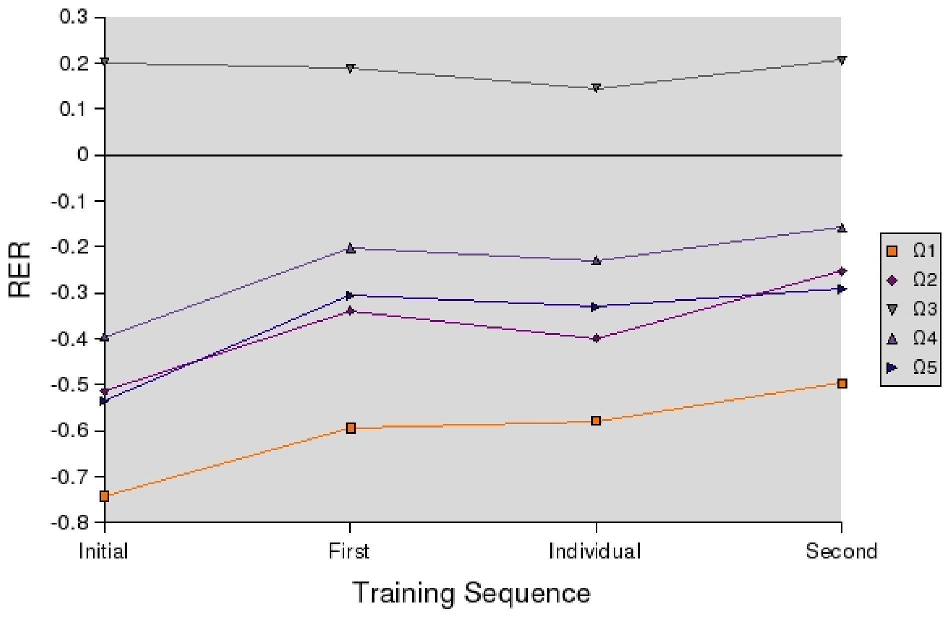


Figure 5.5: Change in Relative Error Reduction for the configurations

For example, the probabilistic grammar criterion (κ_3) prefers parses that minimize the number of grammar rules applied, since each additional rule multiplicatively decreases the score. Contrariwise, the coverage criterion (κ_4) prefers parses that maximize the number of input words used. This particular dichotomy is the result of partial qualification, but is indicative of larger issues related to the computation of scores. Likewise, the informativeness criterion (κ_5) is theoretically opposed to temporal proximity (κ_1) since it promotes the combination of disparate rather than shared information. This is partially played out in the data, as Figure 5.6 demonstrates an apparent correlation between accuracy and $\frac{(\max(\omega_3,\omega_4)-\min(\omega_3,\omega_4))+((\max(\omega_1,\omega_5)-\min(\omega_1,\omega_5))}{2}$ which is the average difference between weights of these theoretically 'competing' modules, although this behaviour might also be the result of other factors.

The sections that follow discuss other aspects uncovered during data analysis.

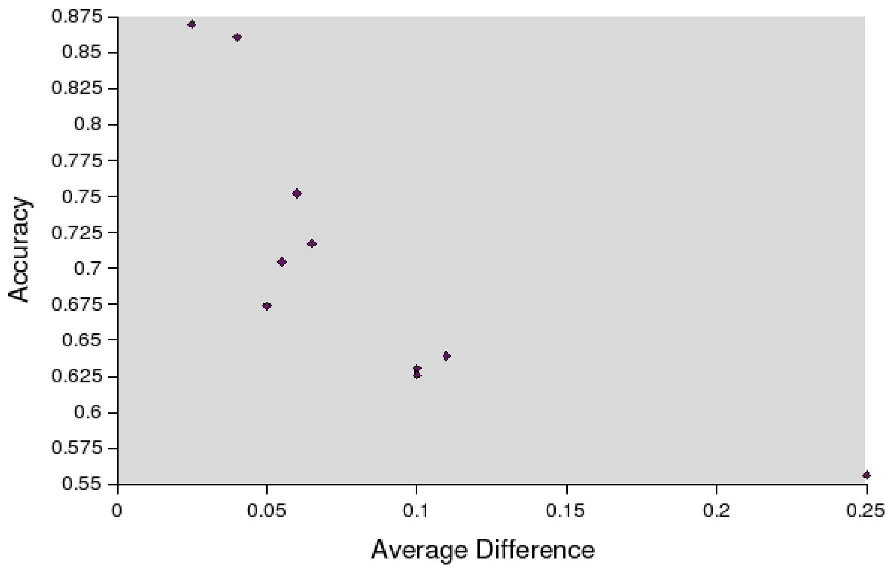


Figure 5.6: Relationship between accuracy and the average difference between weights of 'competing' modules (ω_3, ω_4), and (ω_1, ω_5).

5.3.1. Work Expenditure. To test whether the best-first approach compensates for Clavius' looser constraints (§4.1.1), the bottom-up chart parser of Algorithm 4 (§2.2.3) was constructed and the average number of edges (partial parses) it produces on sentences of varying length in the whole data set Δ was measured. This value approximates the expected work expenditure in GEMINI [**32**], SmartKom [**116**] and Johnston [**49**].

Measures on the chart parser were compared against two measures of Clavius. First, the average number of edges Clavius produces (across all parametrizations of §5.2.3) before the parse it eventually returns is found is recorded for each sentence length. This represents a near-optimal bound on performance of Clavius, as it would ideally make its decision as soon as possible. However, more parses are added to the search space before Clavius returns its decision, so this value is also recorded as a practical indicator of average work expenditure.

Figure 5.7 summarizes these results. In particular, although Clavius generally finds its accepted parse relatively quickly ('Clavius - found'), the cognition module will delay its acceptance ('Clavius - accepted'). The length of this overhead grows with sentence length, from near 0 for sentences of length 2 to over twice the number of edges needed to find the parse.

The main factor in the proportional increase of overhead with sentence length appears to be more empty constituent edges appearing as a result of necessarily more complex grammar constructions. In order to reduce the amount of work *actually* performed by Clavius, either tighter restrictions on the allowance of

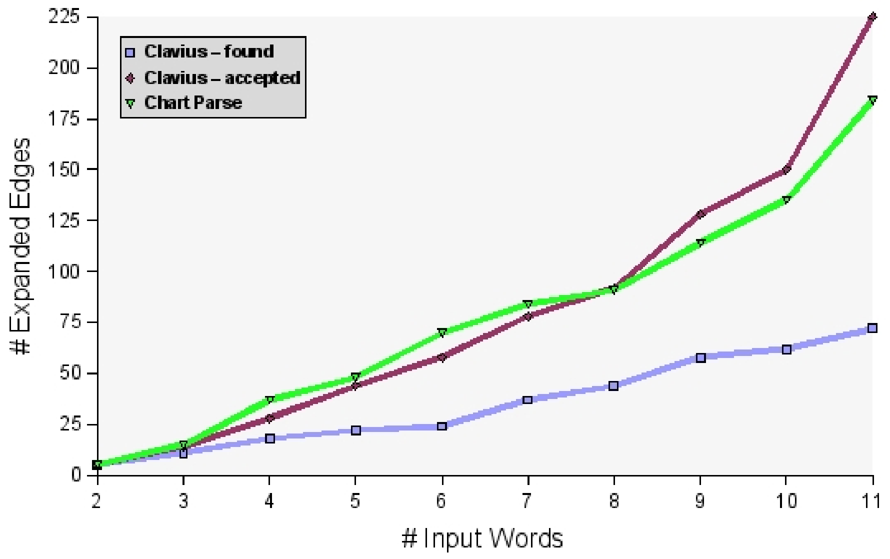


Figure 5.7: Average number of edges expanded, given sentence length.

null constituents need to be enforced, or the cognition module needs to be modified to make its decisions more quickly. Although in practise Clavius does more work than an all-paths bottom-up multidimensional chart parser for longer sentences, it appears comparably efficient for sentences up to length 8, and a more appropriate sentence-acceptance mechanism in the cognition module may improve performance.

This measurement is an important indicator of Clavius' performance relative to state-of-the-art techniques, specifically those mentioned above. In particular, it demonstrates that the best-first process narrows the search space and finds its solution relatively quickly, expanding only about 39% of the number of edges the chart parse model creates for the longest observed sentences.

5.3.2. Questions of Grammatical Usability. The majority of sentences spontaneously uttered by most users were simplistic (see §5.3.4), as was expected. In fact, the grammar of Appendix B was built based on this assumption and covers 2248 (97.6%) of the utterances collected. It is not presently clear whether sentences were simple because of the imperative command structure all users assumed, or because of multimodality itself. Use of pronouns, definite NPs and tense are typical aspects of language speakers employ to simplify expressions through context, but only pronominal references to objects were seen during this study. Unimodal and conversational multimodal data would help categorize these effects.

Experiment participants were able to express themselves in more complex, compound phrases (ex. "*move this* $\searrow$ *triangle here* $\searrow$ *and make it a blue square*") but typically decided to split their commands into two separate phrases. Participants also often opted to delete existing objects and create new ones rather than use the move command coupled with the transform command (ex. "*delete this* $\searrow$ *triangle*" and a "*put* *a blue sphere here* $\searrow$"), which may explain the relative proportions of commands, in Table 5.3. Similarly, although participants were able to use the AREA gesture to select multiple objects simultaneously (§3.5.8), only 1 user discovered it, using it only twice.

Finally, there were 877 (38.1%) instances where dysfluencies (other than artificial noise) occurred. Of these, 814 (92.8%) were *not* part of the sentences accepted by Clavius. Of those 63 that were accepted, 48 constituted simple insertion errors, and the remaining 15 were incorrectly combined with adjacent words by the Sphinx tracker, resulting in substitution errors. The ability to ignore dysfluencies in Clavius is a notable boon to seamless interaction, and appears to be a result of the partial qualifications discussed in §4.1.1.3, which is especially relevant in noisy environments.

5.3.3. Co-occurrence statistics. From the literature, one expects a close correlation in time between the occurrence of deictic gestures with the pronominals or nouns with which they co-refer semantically (κ_1 is based on this assumption). In fact, literature suggests that the relative order of spoken and gestured expressions are dependent on modality, namely that deictic gestures precede associated keywords in pen-based systems [**81**], while overlapping or *following* them 97% of the time in large-screened environments with arm gestures [**101**].

Figure 5.8 compares onset time between speech and gesture between Holzapfel et al. [**43**] and Clavius using the domain (bins covering periods of time of 0.24 seconds) and range (percentage of occurrences[3]) of the former. Holzapfel et al. suggest that arm gestures often occur anywhere between 0.5 seconds before speech to 0.7 seconds after it, with an apparent preference for precedence. This is notably contrasted with REPLICATOR/Clavius, where gestural mouse activity *follows* the beginning of a spoken co-referring word (ex. "this", etc.) 90.9% of the time in a near-Gaussian distribution centred between 0.72 and 0.96 seconds after the onset of speech.

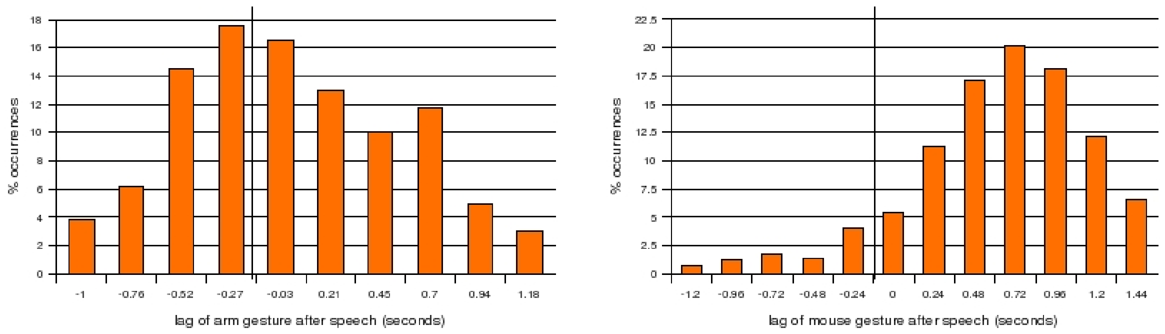


Figure 5.8: Comparison of onset times of arm gesture (from Holzapfel et al.[**43**], left) and of mouse gesture (right) following associated speech keywords.

[3]This metric is inferred from [**43**], as they do not label their y-axis properly.

These results almost indicate the reverse of what one expects, since the literature suggests that speech is generally followed by arm gestures, and preceded by hand-held device input.

In Clavius, over 74% of mouse events in multimodal commands occurred after the *end* of the co-referring speech part, indicating an apparent sequential approach of users to multimodality. This may be partially explained by the familiarity of all users to standard WIMP interfaces that generally ‘expect’ sequential modes of interaction. However, for ‘2-word’ phrases, such as “*red* ↘” or “*delete* ↘”, gestures on average occurred *during* the spoken word, or preceded its onset 91% of the time. Therefore, it appears that temporal proximity is dependent *both* on device *and* on phrase type.

5.3.4. Sentence structure. In §2.3.3, references in the literature were discussed that suggest that an important dichotomy between multimodal and unimodal spoken language is the use of locative constituents, especially in prepositional phrases. Namely, locatives are often moved to a sentence-initial position in pen/voice input – changing S-V-O-LOC word order to a LOC-S-V-O one.

Unfortunately, the data collected for Clavius did not really include any subject arguments of verbs, since all sentences were effectively verb-initial imperative command driven sentences. This sentence type also effectively prohibits locatives from occurring before the verb. However, one can still measure whether the participants altered their expressive grammar with regards to the position of locatives relative to the object reference, between unimodal and multimodal cases.

All **move** and **create** commands used locatives, with 2 occurring with deletion. Table 5.7 summarizes the behaviour of locatives within the context of REPLICATOR. Note the interesting presence of phrase type O-LOC, where the absent leading verb can be inferred from its two arguments. In addition, the vast majority of delete commands, and 64 create commands, did not use any locatives whatsoever.

	Action	**Unimodal Locative**		**Bimodal Locative**	
		Example	Number	Example	Number
V-LOC-O	**create**	“*put ↘ square*”	5	“*put here ↘ a square*”	1
V-O-LOC	**move**	“*put this ↘ ↘*”	1	“*put this ↘ there ↘*”	516
	create	“*put a red square ↘*”	8	“*put a red cube here ↘* ”	474
	delete	N/A	0	“*delete all objects in ↘ this area ↙*”	1
V-LOC	**delete**	“*delete ↘ ↙* ”	1	N/A	0
O-LOC	**move**	“↘ ↘”	3	“*this ↘ there ↘*”	18

Table 5.7: Use of locatives in various sentence structures, according to the arity of the locative.

5.3.4.1. *Conjunctions and Fragments.* Conjunctive sentences of the form “*S and S*” occurred 120 times, 68 of which combined disparate action types (ex. “*move this ↘ here ↘ and colour it green*”). Of these, 40 were interactions on a single object that employed anaphoric pronouns (“*it*”), suggesting preference for cognition of objects rather than actions. In all but three cases these involved one conjunction, but in three cases it involved two conjunctions of the form $(_S\ S\ and\ (_S\ S\ and\ S))$. The relative rarity of conjunctive phrases is to be expected, and emphasises that participants favour simplistic, short phrases in this environment.

Sentence fragments occur frequently in natural speech, for instance a verb phrase lacking a direct object. Since Clavius supports ‘partially complete’ representations at the sentence level, these types of sentences *should* be decipherable, or at least the application could counter appropriately with a request to parametrize the verb (ex. “I’m sorry, which object should I delete?”). Selectional constraints might even be used to approximate the meaning of an unknown word. In this regard, more data would be required to draw any deeper conclusions, as well as a standard analysis method for classifying locative phrases across multimodal and unimodal contexts.

CHAPTER 6

Concluding Discussion and Remarks

This thesis describes a software architecture built to interpret multiple modes of input generically, as described in Chapter 3. The mechanism used in this interpretation is a new multi-threaded and bidirectional parsing algorithm on graph-based unification grammars that follows a best-first approach according to a combination of scoring metrics, described in Chapter 4.

Results from experiments reveal several features of Clavius, discussed further in §6.1 and its subsections, that indicate both a satisfiable general level of applicability, but also several important limiting practical factors. For instance, while certain configurations of Clavius achieve better accuracy than a baseline system, others consistently fare worse, even after training. The complexity of the system leads to several apparent 'hidden' variables (see §5.3) related to dependencies beyond the layers of the XML grammar and scoring modules.

The design of grammars for Clavius is also non-trivial as unexpected consequences can easily result from casual grammars. For example, rules Γ_i and Γ_j having a high degree of RHS overlap can easily create a glut in the search space if care is not taken to necessitate the presence of distinguishing (usually head) words. For example, the determiner *'the'* might be present in both `object_reference` and `location` rules, and if in some utterance it is part of one, parses assuming it is part of the other should be limited. Otherwise, the search space can very quickly be dominated by irrelevant parses, from which Clavius might pull an incorrect sentence decision.

The painstaking construction of grammars in this fashion may also indicate that generalizing the process arbitrarily to new applications is also non-trivial. At this point there remains no single or best generalization to the problem, however multi-component architectures, like Clavius , have recently been gaining in popularity.

The GS Algorithm has performed reasonably admirably, given the novelty of the parsing algorithm. The overview of its key features in §6.1 is followed by a summary of interesting avenues open to exploration in §6.2.

6.1. Features of the GS Algorithm

Multiple distinguishing behavioural features characterize the GS Algorithm. These are consequences of decisions described in §4.1.1 and are summarized below.

6.1.1. Asynchronous collaboration. The bidirectional nature of the GS Algorithm differs from those of Satta [**97**] and Rocio [**91**], for example, except that its asynchronous and distributed nature reduces several processing bottlenecks. Namely, allowing these processes to be performed simultaneously has resulted in quick discovery of accepted parses, for instance, and avoids thread starvation. The modular nature of its scoring functions allows Clavius to incorporate quite disparate approaches and knowledge sources in the field, from acoustic prosody to statistical linguistic information.

A few scoring modules κ_i have different behaviours depending on whether they were invoked by the generalizer or the specifier. This has not yet been detrimental to the process, but since the resultant graphs are always put into the generalizer's search space first, correct internal ordering would be preferable in the long-term.

6.1.2. Unordered constituents. Ignoring ordering of constituents γ_i in the RHS of rule Γ_i is perhaps the most controversial decision in Clavius, since fundamental and often essential syntactic constraints are inherent in the traditional arrangement. While these syntactic constraints can easily be replaced by the use of constraint functions, this might appear to be an unnecessary step.

This aspect of the GS Algorithm still serves as an integral part of the system that effectively handles permutation of multimodal word order and the admittance of unexpected phrase types. Early unquantified observation during development suggests that another resultant phenomenon is that parses incorporating lead words, such as head nouns, command verbs and pointing gestures in particular, are emphasized and form sentence-level parses early, and are later 'filled in' with function words.

6.1.3. Unspecified edges. The process of generalization produces parses that instantiate top-level grammar rules where only one constituent is non-null. This, in combination with a best-first strategy based on several scoring criteria and unordered constituents approximates a dynamic programming approach to multimodal interpretation. This is confirmed in Chapter 5 by the speed with which Clavius discovers its accepted sentences, relative to current approaches in chart parsing.

6.1.4. Constraint Functions. The implementation of κ_8 in Clavius provides a unique graph-theoretic implementation of constraint functions in the λ calculus. To be effective, when values returned from one function parametrize others, it is necessary to evaluate the master term recursively until no fully parametrized functions remain unevaluated.

Secondly, it is important that κ_8 evaluates the *entire disjunction* of all constraint function terms present in a given parse. If this were not the case, constraints that were applied during evaluation would always be lost. This can be accomplished by applying the multiplication operator on each of the constituent and top-most terms.

6.2. Possible Future Explorations

There are several avenues open to Clavius for future exploratory endeavours that extend beyond the goals of this thesis. These are broadly classified into computational, linguistic and interactive subjects.

Computational Aspects. Currently, once a partial parse Ψ is given a score that remains fixed throughout the life cycle of that graph, until it is removed from the search space. To truly take advantage of the real-time recognition capability of Clavius, allowing scores to change dynamically as new information is presented to the system would, in particular, push out semantically incongruous parses by lowering their scores to 0, or thereabouts. Doing so, however, would either incur a substantial computational cost or necessitate specialized semantically indexed data structures whose behaviours are left to future consideration.

Furthermore, as described in §3.9, Clavius transparently lends itself to complex multi-processor architectures, but this aspect has not yet been especially explored. A different computational model of the score (§4.2) might also address issues of codependency observed during experimentation. One possible such candidate is to use a hidden layer in the joint reparametrization of $\overrightarrow{\Omega}_i$ (§5.1.7.1), which is the standard approach to allaying non-linearly separable classes in neural networks. A more measured approach to individual module parametrization (§5.2.2) would also be advised.

Linguistic and Interactivity Aspects. Certain linguistic questions lie beyond the scope of this thesis, primarily those related to context. All language in Clavius is constricted to simple verb-initial imperative command phrases, whereas the GS Algorithm is designed to inhabit the same class as general text parsers. In order to evaluate its potential in this space, more complex linguistic phenomena need to be analyzed. In particular, long-distance dependencies and proper anaphora resolution co-referring entities and concepts are open areas. Currently, concatenating multiple sentences together can be used to force unification on two semantic entities deterministically, but more applicable approaches exist to this problem. New grammars need to expand to cover more general phrases, such as those popularized in the Wall Street Journal corpus, and be applied to more general application contexts such as conversational agents.

Other areas of linguistic interest include the use of the subsumption architecture between scoring modules κ_i, such that κ_i can selectively apply $\Re_{[0..1]}$ multipliers on some set $\{\kappa_j \| j \neq i\}$, effectively de-activating certain modules under particular conditions. If grammatical selectional constraints, for example, override language modelling, the former can nullify the effect of the latter with a 0 multiplier. This has a direct effect on the interpretation of the language. Elements of compositional and generative semantics also play a role, for instance in cases where discourse entities cannot be immediately instantiated in the world.

Finally, further investigation into alternate methods of sentence acceptance in the augmentation of the COGNITION module would take advantage of Clavius' speed in this regard, and may more effectively utilize inherent benefits of cross-modal information in increasing current accuracy rates.

APPENDIX A

Essential Background

Certain aspects of this thesis assume a certain level of understanding of fundamental mathematical and linguistics concepts. Some of these are assembled here for the elucidation of the curious reader.

A.1. Mathematic Essentials

In this section we briefly review two key areas of graphical models and probability theory as they relate to fundamental concepts in CLAVIUS.

A.1.1. Graphical Models. Graphs pervade much of computer science and engineering and are used extensively in CLAVIUS by virtue of their use in representing *all* user activity. The structure of the type of rooted graphs used in CLAVIUS were provided in §2.1. Most of the usage of these graphs involve traversal down from their roots to extract particular information, although certain classic graph algorithms, such as Dijkstra's shortest-path algorithm [**26**] play roles in its implementation. Some relevant aspects are discussed in the following subsections.

A.1.1.1. *Size Bounds on* CLAVIUS *DAGs.*

THEOREM A.1.1. *For directed, and acyclic graph Ψ_a with a set of nodes N, $\frac{|N|(|N|-1)}{2} \geq |\Psi_a| \geq |N| - 1$*

PROOF. Proof of this comes in two steps:

- $|\Psi_a| \geq |N| - 1$ proved by induction. If $|N| = 1$ (Ψ_a is atomic), then it has $0 \geq |N| - 1$ edges. Then, we assume that we have some subgraph Ψ'_a with $|N| - 1 \geq 0$ nodes such that $|\Psi'_a| \geq |N| - 2$. If we add a new node to Ψ'_a, forming Ψ_a, then that node must be connected to at least one other node by an arc, thus $|\Psi_a| \geq |\Psi'_a| + 1 \geq |N| - 2 + 1 = |N| - 1$.
- $\frac{|N|(|N|-1)}{2} \geq |\Psi_a|$ by a mechanism similar to that used in the Pidgeonhole Principle [**92**]. Namely, given a set of $|N|$ nodes, we attempt to add as many arcs as possible. Initially, with 0 arcs, we

can pick any node and draw it's maximum number its outgoing arcs - namely directed arcs to the $|N| - 1$ other nodes. Assuming that after drawing the maximum allowable number of outgoing arcs for the first n nodes, then the maximum number of arcs we can add for the $(n-1)^{th}$ node x is $|N| - n - 1$ - because the first n nodes all already point to x - and we cannot allow cycles or reflexive loops (no arc from x to x is allowed). In this way, we get a maximum $\max|\Psi_a| = \sum_{n=1}^{|N-1|} n = \frac{|N|(|N|-1)}{2}$ [**99**].

□ □

A.1.2. Optimization. Experiments in CLAVIUS included the use of simple single-layer feedforward neural networks without explanation of how they are trained (§5.1.7.1), due to the somewhat tangential nature of their details.

This network was trained using a simple linear step and binary decision rule as output, o. Specifically, given weights $\omega_j \in \Re_{[0..1]}$ and activations $a_j \in \Re_{[0..1]}$ (the latter are determined from the appropriate κ_j functions), then

$$\begin{cases} 0 & \text{if} \sum_j \omega_j a_j < \theta, \\ 1 & \text{otherwise.} \end{cases} \tag{A.1}$$

The parameter $\theta = 0.8$ for the given experiments. For a set of data Δ, if instance Δ_i is recognized correctly then ω_j is left unchanged for all j. If, however, the proposed sentence recognized by CLAVIUS does not provide the correct lexography, then the update rule $\omega_j \leftarrow \omega_j + \alpha a_j$ adjusts the network weights, given some learning rate parameter α [1]. More introductory information can be found in [**95**].

A.1.3. Derivation of Kullback Leibler. The symmetric Kullback-Leibler divergence is utilized in one of CLAVIUS' scoring modules (κ_1, §4.3). Its partial derivation is provided here. Remembering that

(i) $\frac{1}{\sigma\sqrt{2\pi}} \int e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx = 1$
(ii) if X is a continuous random variable, and $f(x)$ is the value of its probability density at x, then the expected value of $g(x)$ is $E[g(X)] = \int g(x)f(x)dx$ [**35**]
(iii) $\sigma^2 = E[X^2] - E[X] = E\left[X - E[X]^2\right] = E\left[(X-\mu)^2\right]$

substituting the Gaussians, gives

[1]There is no derivative of an activation function, since the threshold function is non-differentiable

$$
\begin{aligned}
KL\left(N_1 \| N_2\right) &= \frac{1}{\sigma_1\sqrt{2\pi}} \int e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} \log \frac{\sigma_2}{\sigma_1} e^{\left(\frac{(x-\mu_2)^2}{2\sigma_2^2} - \frac{(x-\mu_1)^2}{2\sigma_1^2}\right)} dx \\
&= \frac{1}{\sigma_1\sqrt{2\pi}} \int e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} \left[\log \frac{\sigma_2}{\sigma_1} + \frac{\left(x-\mu_2\right)^2}{2\sigma_2^2} - \frac{\left(x-\mu_1\right)^2}{2\sigma_1^2}\right] dx \\
&= \log \frac{\sigma_2}{\sigma_1} + \frac{1}{\sigma_1\sqrt{2\pi}} \int e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} \left[\frac{\left(x-\mu_2\right)^2}{2\sigma_2^2} - \frac{\left(x-\mu_1\right)^2}{2\sigma_1^2}\right] dx \\
&= \log \frac{\sigma_2}{\sigma_1} - \int \frac{\left(x-\mu_1\right)^2}{2\sigma_1^2} N_1\left(x, \mu_1, \sigma_1\right) dx + \frac{1}{\sigma_1\sqrt{2\pi}} \int e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} \frac{\left(x-\mu_2\right)^2}{2\sigma_2^2} dx \\
&= \log \frac{\sigma_2}{\sigma_1} - E\left[\frac{\left(x-\mu_1\right)^2}{2\sigma_1^2}\right] + \frac{1}{\sigma_1\sqrt{2\pi}} \int e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} \frac{\left(x-\mu_2\right)^2}{2\sigma_2^2} dx \\
&= \log \frac{\sigma_2}{\sigma_1} - \frac{E\left[\left(x-\mu_1\right)^2\right]}{2\sigma_1^2} + \frac{1}{\sigma_1\sqrt{2\pi}} \int e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} \frac{\left(x-\mu_2\right)^2}{2\sigma_2^2} dx \\
&= \log \frac{\sigma_2}{\sigma_1} - \frac{\sigma_1^2}{2\sigma_1^2} + \frac{1}{\sigma_1\sqrt{2\pi}} \int e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} \frac{\left(x-\mu_2\right)^2}{2\sigma_2^2} dx
\end{aligned}
$$

In order to break down our last indefinite integral, a redundant μ_1 is introduced into the second term such that

$$
\begin{aligned}
\left(x-\mu_2\right)^2 &= \left(x-\mu_1+\mu_1-\mu_2\right)^2 \\
&= \left[\left(x-\mu_1\right)+\left(\mu_1-\mu_2\right)\right]^2 \\
&= \left(x-\mu_1\right)^2 + 2\left(x-\mu_1\right)\left(\mu_1-\mu_2\right) + \left(\mu_1-\mu_2\right)^2
\end{aligned}
$$

so

$$
\begin{aligned}
\frac{1}{\sigma_1\sqrt{2\pi}} \int e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} \frac{\left(x-\mu_2\right)^2}{2\sigma_2^2} dx &= \frac{1}{\sigma_1\sqrt{2\pi}} \int \frac{(x-\mu_1)^2}{2\sigma_2^2} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} dx + \frac{2}{\sigma_1\sqrt{2\pi}} \int \frac{(x-\mu_1)(\mu_1-\mu_2}{2\sigma_2^2} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} dx \\
&\quad + \frac{1}{\sigma_1\sqrt{2\pi}} \int \frac{(\mu_1-\mu_2)^2}{2\sigma_2^2} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} dx \\
&= \frac{E\left[(x-\mu_1)^2\right]}{2\sigma_2^2} + 2(\mu_1-\mu_2)\frac{E\left[(x-\mu_1)\right]}{2\sigma_2^2} + (\mu_1-\mu_2)^2 \frac{E\left[1\right]}{2\sigma_2^2} \\
&= \frac{\sigma_1^2}{2\sigma_2^2} + 0 + \frac{(\mu_1-\mu_2)^2}{2\sigma_2^2}
\end{aligned}
$$

Substituting this result back into our derivation of the KL distance gives

$$
\begin{aligned}
KL\left(N_1 \| N_2\right) &= \log\frac{\sigma_2}{\sigma_1} - \frac{\sigma_1^2}{2\sigma_1^2} + \frac{1}{\sigma_1\sqrt{2\pi}}\int e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}}\frac{\left(x-\mu_2\right)^2}{2\sigma_2^2}dx \\
&= \log\frac{\sigma_2}{\sigma_1} - \frac{1}{2} + \frac{\sigma_1^2}{2\sigma_2^2} + \frac{(\mu_1-\mu_2)^2}{2\sigma_2^2}
\end{aligned} \tag{A.2}
$$

From this point on the derivation follows that in Johnson [**47**].

A.2. Linguistic Essentials

Certain linguistic concepts are required for discussion of this work. The following subsections cover some basic topics of computational linguistics within the specific context of multimodal language.

A.2.1. Lexography and Morphology. Words can be broadly classified into parts-of-speech (POS) or lexographies, and differ from one another via numerous morphologies on common roots. For example, adding the trailing *-s* to the word *noun* is a morphology from the singular noun class to the plural noun class. Table A.1 provides a subset of the part-of-speech *tags* in the Penn Treebank [**104**], one of many such classification schemes. Eric Brill's transformational tagger is a popular tool to automatically associate POS tags with words in a corpus [**52**], with an expected accuracy (proportion of potentially polysemous words uniquely tagged with the correct POS) of 95%. This, however, would lead to an average of 1 error per 20-word sentence, having potentially negative consequences in parsing, in general. These complications, among others, are the reason why CLAVIUS does not perform this step explicitly. Note that Table A.1 also includes POS tags for POINT and AREA 'words'. In general, CLAVIUStreats lexical entries from all modalities the same, so deictic gestures are treated with the same mechanisms as written or spoken words.

tag	type	example	tag	type	example
CC	coordinating conjunction	*this* **and** *that*	PP$	possessive pronoun	**my** *lodge*
CD	cardinal number	**18**	RB	adverb	*the threads work* **concurrently**
DT	determiner	**the** *confectionery*	RBR	adverb, comparative	*I have* **more** *credibility*
IN	preposition or subordinating conjunction	**in** *the vestibule*	RBS	adverb, superlative	*the* **most** *chicken feed*
JJ	adjective	*the* **happy** *camper*	UH	interjection or dysfluency	*fool me once, shame on* **uh...** *you*
JJR	adjective, comparative	*the* **happier** *camper*	VB	verb, base form	*waiting for it to* **happen**
JJS	adjective, superlative	*the* **happiest** *camper*	VBD	verb, past tense	*it* **happened**
NN	noun, singular or mass	*that* **monkey**	VBG	verb, gerund or present participle	*it is* **happening**
NNS	noun, plural	*these* **boxes**	POINT	point deixis	$\searrow_{(x,y)}$
PP	personal pronoun	**I** *am Torgo*	AREA	area deixis	$\searrow_{(x,y,height,width)}\swarrow$

Table A.1: Selection of common parts of speech and their tags, from a subset of the Penn Treebank, plus additional tags POINT and AREA for deictic gestures.

A.2.2. Phrase Structure. **Context-free grammar rules** are expressions of the type A $\leftarrow$ B where B is any sequence of *terminals* or *non-terminals* - usually ordered. For example, the rule Γ_i : NP $\leftarrow DTJJNN$ claims that a noun phrase (NP) can be composed of a determiner, adjective, and singular noun in that order. The noun phrase is a non-terminal of the English grammar, and can be composed in a number of different ways.

A **partial parse** is a specific application of particular grammar rules to part or all of input terminals. For example, if a system is given text "*read the fine manual*", then the application of the rule Γ_i (above) to the subset "*the fine manual*" constitutes a partial parse of the sentence. This is typically expressed using bracketed terminology prefixed with non terminals, as in

$$(_{NP}(_{DT}the)(_{JJ}fine)(_{NN}manual))$$

Some semantic information can be inferred from such syntactic structure, but some info cannot so easily be extracted. For instance, the phrase "*delete* $\searrow$ <u>*this*</u> *box*" cannot in practice be understood with 100% certainty, if the deixis $\searrow$ falls between two objects. Information from the world is often required to complete interpretation.

A.2.3. Ambiguity and Uncertainty. Ambiguity and uncertainty can exist at all levels of the recognition process. Errors in tracking, or noise on the transmission line can distort the signal from tracked input such as speech or video. Segmentation between word boundaries is also often stochastic, and multiple POS tags can exist for a particular word ("*book*" can be a NN or a VB, for instance).

Structural ambiguity occurs when a single sequence of words can be covered by more than one possible combination of grammar rules, as exemplified in Figure A.1. Ambiguities related to attachment and coordination are frequent, and choosing between multiple possible syntactic interpretations is the subject of ongoing research.

A.2.4. Language Models. Statistical language models are often used in ASR to estimate word strings from acoustic data by means of maximum likelihood *a posteriori* decision rules to determine the optimal word sequence W_C for a given speech signal X:

$$W_C = \arg\max_W P(W|X) = \arg\max_W \frac{P(W)P(X|W)}{P(X)} \tag{A.3}$$

where $P(X|W)$ is the *optimal* acoustic model and $P(W)$ is the *optimal* language model [**61**]. Since W is an ordered sequence of words w_i ($W = \{w_1\ w_2\ ...w_n\}$), the word sequence probability, from the multiplication rule, is:

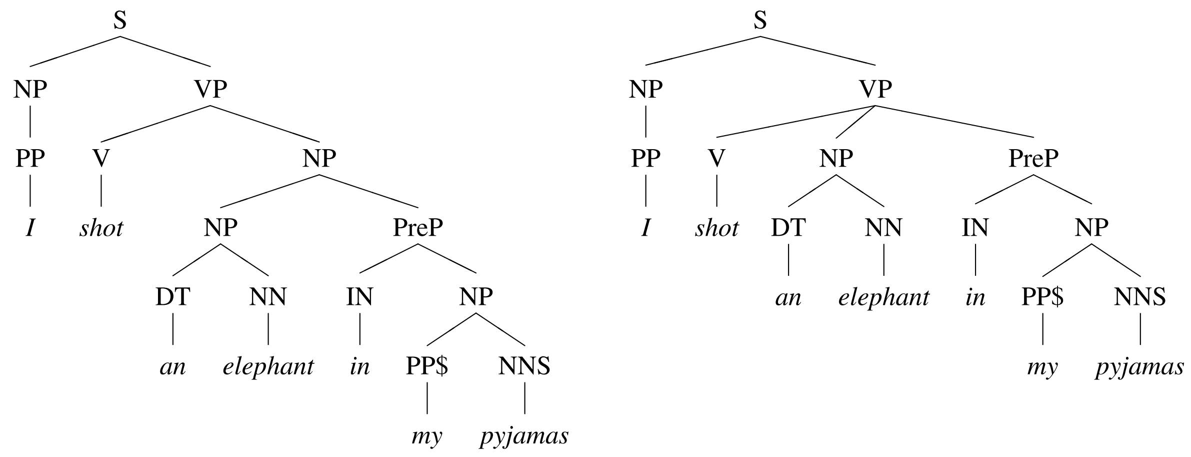


Figure A.1: Two interpretations of the phrase "*I shot an elephant in my pyjamas*", from Groucho Marx. The left parse suggests that the shot elephant was in the interlocutor's pyjamas, whereas the right parse indicates that the shooter was wearing their pyjamas at the time of the shooting.

$$\begin{aligned} P(W) &= P(w_1\ w_2\ ...w_n) \\ &= P(w_1)P(w_2|w_1)P(w_3|w_1\ w_2)\cdots P(w_n|w_1\ w_2\ldots w_{n-1}) \end{aligned} \tag{A.4}$$

where $P(w_i|w_1\ldots w_{i-1})$ is the probability that word w_i follows the ordered words $w_1\ldots w_{i-1}$. However, for sentences of any reasonable length, the dimensionality of this probability space becomes unwieldy, even for moderate-sized vocabularies - so a typical solution is to approximate conditional probabilities with only a single conditioning variable as in $P(w_x|w_y)$, giving

$$P(W) = P(w_1|<s>)P(w_2|w_1)P(w_3|w_2)P(w_4|w_3)\cdots P(w_n|w_{n-1}) \tag{A.5}$$

where $<s>$ is the *start symbol* indicating that $P(w_1|<s>)$ is the probability that w_1 begins a sentence. Employing conditional probabilities in this way is called the *bigram* model, and naturally incorporates ordering in language. Language models are sometimes preferred to parsing models in deciding between competing parses in terms of their 'naturalness' [**67**], although the absense of explicit structural representation keeps them from being a drop-in replacement for PCFGs.

A.2.5. Measures of Accuracy. Accuracy in general NLP is not limited to simple ratios of correct vs. incorrect interpretations. For instance,

$$\textbf{Recall} = \frac{\text{\# of correct hypotheses given by a system}}{\text{total \# of possible interpretations in a text}}$$

indicates how much of a text a given interpreter can 'cover', if it is allowed to not provide interpretations of sentences. Precision is more closely related to the standard definition of accuracy, and in the context of this thesis is synonymous with it:

$$\textbf{Precision} = \frac{\text{\# of correct hypotheses given by a system}}{\text{total \# of hypotheses given by a system}}$$

Errors in system hypotheses are often individually scrutinized subjectively, or classified into **insertion**, **deletion** or **replacement** types if they differ from a desired interpretation by the adding of extra words, removal of expected words, or misinterpretation of words, respectively.

Relative Error Reduction (RER) is used to compare improvements gained in precision non-commutatively by moving from system A to system B, where systems A and B may simply be reparameterizations of each other. Specifically,

$$RER(A, B) = \frac{\text{error rate in A} - \text{error rate in B}}{\text{error rate in A}}$$

APPENDIX B

Example Grammar

The following XML grammar exemplifies a subset of the one used in the experiments described in Chapter 5.

Grammar rules are elements one level below the <CLAV_GRAMMAR> tag. The name of one of these elements is the name of LHS constituent, and element names below the <RHS> tag are the RHS constituents of the rule. Otherwise, in general element names are used as feature names Φ in the graphs implementing the respective rules.

Reentrant nodes are specified by the '\$' symbol. For example, in clav_UTT $\leftarrow$ colour_command below, the content of utterance is linked to the content of the colour command by the '\$1' variable (characters following the '\$' define variable names for graph nodes). The '^' symbol is used to indicate return values from constraint functions, and also refer to reentrant nodes.

```
<CLAV_grammar>

<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!--                                   CLAV_UTT                                      -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- this defines the rule: clav_UTT <-- clav_UTT 'and' clav_UTT -->
<clav_UTT>
   <p_gamma>0.1</p_gamma>
   <time > </time>
   <score>  </score>
   <content> ^3  </content>
   <rhs>
       <clav_UTT> $1<content> </content> </clav_UTT>
       <and></and>
       <clav_UTT> $2<content> </content> </clav_UTT>
   </rhs>
   <params> </params>
   <constraints>
     <![CDATA[  add_Commands( $1, $2, ^3) ]]>
   </constraints>
</clav_UTT>

<clav_UTT>
   <p_gamma>0.2</p_gamma>
   <time > </time>
   <score>  </score>
   <content> $1  </content>
   <rhs>
```

```
      <colour_command> $1<content> </content>
       $cons <constraints></constraints>
      </colour_command>
   </rhs>
   <params> </params>
   <constraints> </constraints>
</clav_UTT>

<clav_UTT>
   <p_gamma>0.2</p_gamma>
   <time > </time>
   <score>  </score>
   <content> $1  </content>
   <rhs>
      <delete_command> $1<content> </content>
         $cons <constraints></constraints>
      </delete_command>
   </rhs>
   <params> </params>
   <constraints></constraints>
</clav_UTT>

<clav_UTT>
   <p_gamma>0.2</p_gamma>
   <time > </time>
   <score>  </score>
   <content> $1  </content>
   <rhs>
      <create_command> $1<content> </content>
         $cons <constraints></constraints>
      </create_command>
   </rhs>
   <params> </params>
   <constraints></constraints>
</clav_UTT>

<clav_UTT>
   <p_gamma>0.2</p_gamma>
   <time > </time>
   <score>  </score>
   <content> $1  </content>
   <rhs>
      <transform_command> $1<content> </content>
         $cons <constraints></constraints>
      </transform_command>
   </rhs>
   <params> </params>
   <constraints></constraints>
</clav_UTT>

<clav_UTT>
   <p_gamma>0.2</p_gamma>
   <time > </time>
   <score>  </score>
   <content> $1  </content>
   <rhs>
      <move_command> $1<content> </content>
          $cons <constraints></constraints>
       </move_command>
   </rhs>
   <params> </params>
   <constraints></constraints>
</clav_UTT>

<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!--                                       COLOUR_COMMAND                                        -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- colours (an) object(s) the specified colour examples: -->
<!-- "colour this (click) red" -->
<!-- "blue (click)" -->
<colour_command>
   <p_gamma>0.5</p_gamma>
   <time >  </time>
   <score>  </score>
   <content>
     <action> colour </action>
```

```
      <colour> $c </colour>
      <objects> $o1 </objects>
   </content>
   <rhs>
     <colour>
         <content> $c <colour> </colour> </content>
     </colour>
     <object_reference>
         <content> $o1 <objects> </objects> </content>
     </object_reference>
   </rhs>
   <params> </params>
   <constraints></constraints>
</colour_command>

<colour_command>
   <p_gamma>0.5</p_gamma>
   <time >  </time>
   <score>  </score>
   <content>
     <action> colour </action>
     <colour> $c </colour>
     <objects> $o1 </objects>
   </content>
   <rhs>
     <VB>
         $t_vb <time> </time>
               <word> colour </word>
     </VB>
     <object_reference>
         $t_objref <time> </time>
         <content> $o1 <objects> </objects> </content>
     </object_reference>
     <colour>
         $t_col <time> </time>
         <content> $c <colour> </colour> </content>
     </colour>
   </rhs>
   <params>
       $td  <td> 5 </td>
   </params>
   <constraints>
   <![CDATA[
        t_follows($t_col,$t_vb,$td) * t_follows($t_objref,$t_vb,$td)  ]]>
   </constraints>
</colour_command>

<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!--                                           DELETE_COMMAND                                                       -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- deletes (an) object(s) examples: -->
<!-- "delete these (click) (click) (click)" -->
<!-- "remove all blue cubes" -->
<delete_command>
   <p_gamma>1.0</p_gamma>
   <time >  </time>
   <score>  </score>
   <content>
     <action> delete </action>
     <objects> $o1 </objects>
   </content>
   <rhs>
     <delete>
           $t_vb <time> </time>
     </delete>
     <object_reference>
         $t_objref <time></time>
         <content> $o1 <objects> </objects> </content>
     </object_reference>
   </rhs>
   <params> </params>
   <constraints>
    <![CDATA[
             depends_on($t_objref, $t_vb) ]]>
   </constraints>
</delete_command>
```

```
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!--                                          CREATE_COMMAND                                           -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- creates (an) object(s) in the specified location  examples: -->
<create_command>
   <p_gamma>0.0</p_gamma>
   <time >  </time>
   <score>  </score>
   <content>
     <action> create </action>
     <number> $n </number>
     <colour> $c </colour>
     <shape>  $s </shape>
     <x>      $x </x>
     <y>      $y </y>
     <height> $h </height>
     <width>  $w </width>
   </content>
   <rhs>
     <create> </create>
     <CD>
        $t_num <time> </time>
        $n <number> </number>
     </CD>
     <colour>
         $t_col <time> </time>
         <content> $c <colour> </colour> </content>
     </colour>
     <obj_type>
         $t_shape <time> </time>
         <content> $s <shape> </shape> <num> p </num> </content>
     </obj_type>
     <location>
         <content>
      $x <x>      </x>
      $y <y>      </y>
      $h <height> </height>
      $w <width>  </width>
         </content>
     </location>
   </rhs>
   <params>
       $td  <td> 5 </td>
   </params>
   <constraints>
   <![CDATA[
        t_follows($t_shape,$t_col,$td) * t_follows($t_shape, $t_num, $td) * t_follows($t_col,$t_num,$td)  ]]>
   </constraints>
</create_command>

<create_command>
   <p_gamma>0.5</p_gamma>
   <time >  </time>
   <score>  </score>
   <content>
     <action> create </action>
     <number> 1 </number>
     <colour> $c </colour>
     <shape>  $s </shape>
     <x>      $x </x>
     <y>      $y </y>
     <height> $h </height>
     <width>  $w </width>
   </content>
   <rhs>
     <create> $t_create <time> </time> </create>
     <a> $t_a <time> </time> </a>
     <colour>
         $t_col <time> </time>
         <content> $c <colour> </colour> </content>
     </colour>
     <obj_type>
         $t_shape <time> </time>
         <content> $s <shape> </shape> <num> s </num></content>
     </obj_type>
```

```
      <location>
          <content>
       $x <x>      </x>
       $y <y>      </y>
       $h <height> </height>
       $w <width>  </width>
          </content>
      </location>
   </rhs>
   <params>
       $td  <td> 5 </td>
   </params>
   <constraints>
   <![CDATA[
        t_follows($t_shape,$t_col,$td) * t_follows($t_shape, $t_a, $td) * t_follows($t_shape,$t_create,$td)
       *t_follows($t_col, $t_a, $td) * t_follows($t_col,$t_create,$td)  * t_follows($t_a,$t_create,$td)      ]]>
   </constraints>
</create_command>

<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!--                                    TRANSFORM_COMMAND                                                  -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<transform_command>
   <p_gamma>0.5</p_gamma>
   <time >  </time>
   <score>  </score>
   <content>
      <action> transform </action>
      <objects> $o1 </objects>
      <colour> $colour </colour>
      <shape> $s </shape>
   </content>
   <rhs>
       <colour> $t_jj <time> </time>
                    <content> $colour <colour> </colour>
                    </content>
     </colour>
     <obj_type>
         $t_objtype <time> </time>
         <content> $s <shape> </shape> <num> s </num></content>
     </obj_type>
     <object_reference>
         <time> </time>
         <content> $o1 <objects> </objects> </content>
     </object_reference>
   </rhs>
   <params>   $td  <td> 5 </td> </params>
   <constraints>  <![CDATA[ t_follows($t_objtype,$t_jj,$td)*depends_on($t_jj, $t_objtype) ]]></constraints>
</transform_command>

<!-- turn OBJECT into COLOUR OBJECT_TYPE -->
<transform_command>
   <p_gamma>0.5</p_gamma>
   <time >  </time>
   <score>  </score>
   <content>
      <objects> $o1 </objects>
      <colour> $colour </colour>
      <shape> $s </shape>
   </content>
   <rhs>
     <transform> $t_transform<time></time> </transform>
     <object_reference>
         $t_objref <time> </time>
                   <content> $o1 <objects> </objects> </content>
     </object_reference>
     <IN> $t_in <time></time> <word> into </word> </IN>
      <!-- the colour of an object -->
     <colour> $t_jj <time> </time>
                    <content> $colour <colour> </colour>
                    </content>
     </colour>
     <obj_type>
         $t_objtype <time> </time>
         <content> $s <shape> </shape> <num> s </num></content>
     </obj_type>
   </rhs>
   <params>
       $td  <td> 5 </td>
   </params>
   <constraints>
   <![CDATA[
        t_follows($t_objtype,$t_jj,$td) * t_follows($t_jj, $t_in, $td) * t_follows($t_in,$t_objref,$td)
       *t_follows($t_objref, $t_transform, $td) *depends_on($t_jj, $t_objtype) ]]>
   </constraints>
</transform_command>

<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!--                                     MOVE_COMMAND                                           -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<move_command>
   <p_gamma>0.5</p_gamma>
   <time >  </time>
   <score>  </score>
   <content>
      <action> move </action>
      <objects> $o1 </objects>
      <x> $x </x>
      <y> $y</y>
      <height> $h</height>
      <width> $w </width>
   </content>
   <rhs>
     <move> $tmov <time></time> </move>
     <object_reference>
         $tref <time> </time>
         <content> $o1 <objects> </objects> </content>
     </object_reference>
     <location>
         $tloc <time> </time>
 <content> $x <x> </x>
           $y <y>  </y>
                   $h <height>  </height>
                   $w <width>  </width>
         </content>
     </location>
   </rhs>
   <params>      $dxy <dxy> 100 </dxy>
                 $td  <td>  5 </td>
   </params>
   <constraints>
    <![CDATA[ t_follows($tloc,$tref,$td) * t_follows($tref, $tmov, $td) ]]>
</constraints>
</move_command>

<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!--                                  OBJECT_REFERENCE                                         -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- this non terminal refers to one or more objects in the scene examples: -->
<!-- {that|this} (click) {{red|blue...} cube|...} : a single object, at the specified location -->
<!-- these (area)                                 : all objects within the specified area -->
<!-- these (click)+                               : each object clicked -->
<!-- these cubes (area)                           : all cubes within the specified area -->
<!-- these cubes (click)+                         : each cube clicked -->
<!-- all {green|blue...} objects|cubes|...        : all objects with the specified attributes-->
<!-- all cubes to the left of this sphere (click) : all cubes whose x coordinate is less than the indicated sphere -->
<!-- all red spheres (from|in) this area (area)   : all red spheres within the specified area -->

<!-- single object -->
<object_reference>
   <p_gamma>0.25</p_gamma>
    <time >  </time>
    <score > </score>
    <content>
       $o1 <objects> ^o1</objects>
    </content>
    <rhs>
         <!-- a single determiner -->
         <DT>
            $t_dt   <time> </time>
                    <num> s </num>
         </DT>
 <!-- a click -->
         <DEIXIS>
            $tclick <time > </time>
            <score > </score>
    $x      <x> </x>
    $y      <y> </y>
            <left> down </left>
            <middle> </middle>
            <right> </right>
 </DEIXIS>
 <!-- the colour of an object -->
         <colour> $t_jj <time> </time>
                        <content> $colour <colour> </colour>
                        </content>
         </colour>
 <!-- the type of object -->
 <obj_type>
            $t_nn <time> </time>
            <content> $obj_type <shape> </shape> <num> s </num></content>
         </obj_type>
    </rhs>
    <params>
       $dxy <dxy> 100 </dxy>
       $td  <td>  5 </td>
    </params>
    <constraints>
       <![CDATA[
             objectsAt ( $x, $y, $dxy, $dxy, ^o1 ) * objectsType($o1, $obj_type ) *objectsColour( $o1, $colour ) * t_foll
    </constraints>
</object_reference>

<object_reference>
   <p_gamma>0.25</p_gamma>
    <time >  </time>
    <score > </score>
    <content>
       $o1 <objects> ^o1</objects>
    </content>
    <rhs>
         <!-- a single determiner -->
         <DT>
            $t_dt   <time> </time>
                    <num> s </num>
         </DT>
 <!-- a click -->
         <DEIXIS>
            $tclick <time > </time>
            <score > </score>
    $x      <x> </x>
    $y      <y> </y>
            <left> down </left>
            <middle> </middle>
            <right> </right>
 </DEIXIS>
    </rhs>
    <params>
       $dxy <dxy> 100 </dxy>
       $td  <td>  5 </td>
    </params>
    <constraints>
       <![CDATA[
             objectsAt ( $x, $y, $dxy, $dxy, ^o1 )  ]]>
    </constraints>
</object_reference>

<object_reference>
   <p_gamma>0.25</p_gamma>
    <time >  </time>
    <score > </score>
    <content>
       $o1 <objects> ^o1</objects>
    </content>
    <rhs>
         <DEIXIS>
            $tclick <time > </time>
            <score > </score>
    $x      <x> </x>
    $y      <y> </y>
            <left> down </left>
            <middle> </middle>
            <right> </right>
 </DEIXIS>
    </rhs>
    <params>
       $dxy <dxy> 100 </dxy>
       $td  <td>  5 </td>
    </params>
    <constraints>
       <![CDATA[
             objectsAt ( $x, $y, $dxy, $dxy, ^o1 )  ]]>
    </constraints>
</object_reference>

<!-- many objects - selection -->
<!-- area -->
<object_reference>
   <p_gamma>0.25</p_gamma>
    <time >  </time>
    <score > </score>
    <content>
        $o1 <objects> ^o1</objects>
    </content>
    <rhs>
         <!-- a plural determiner -->
         <DT>
            $t_dt   <time> </time>
                    <num> p </num>
         </DT>
 <!-- an area -->
         <AREA>
            $t_area <time > </time>
            <score > </score>
    $x      <x> </x>
    $y      <y> </y>
            $width  <width> </width>
    $height <height> </height>
 </AREA>
 <!-- the colour of an object -->
         <colour> $t_jj <time> </time>
                        <content> $colour <colour> </colour>
                        </content>
         </colour>
 <!-- the type of object -->
 <obj_type> $t_nn <time> </time>
            <content> $obj_type <shape> </shape> <num> p </num></content>
         </obj_type>
    </rhs>
    <params>
       $dxy <dxy> 100 </dxy>
       $td  <td>  5 </td>
    </params>
    <constraints>
       <![CDATA[
             objectsIn ( $x, $y, $width, $height, ^o1 ) *  objectsType($o1, $obj_type ) *objectsColour( $o1, $colour ) *
    </constraints>
</object_reference>

<!-- multiclicks -->
<object_reference>
   <p_gamma>0.25</p_gamma>
    <time >  </time>
    <score > </score>
    <content>
         <objects> $o1</objects>
    </content>
    <rhs>
         <!-- a plural determiner -->
         <DT>
            $t_dt   <time> </time>
                    <num> p </num>
         </DT>
 <!-- multiple clicks -->
```

```
        <multi_clicks>
           $t_clicks <time > </time>
                $o1  <objects> </objects>
 </multi_clicks>
 <!-- the colour of an object -->
        <colour> $t_jj <time> </time>
                       <content> $colour <colour> </colour>
                       </content>
        </colour>
 <!-- the type of object -->
 <obj_type> $t_nn <time> </time>
           <content> $obj_type <shape> </shape> <num> p </num></content>
        </obj_type>
    </rhs>
    <params>
       $dxy <dxy> 100 </dxy>
       $td  <td>  5 </td>
    </params>
    <constraints>
       <![CDATA[
             objectsType($o1, $obj_type ) *objectsColour( $o1, $colour ) * t_follows($t_jj,$t_dt,$td) * t_follows($t_nn,
    </constraints>
</object_reference>

<!-- many objects - referential -->
<object_reference>
   <p_gamma>0.25</p_gamma>
    <time >  </time>
    <score > </score>
    <content>
         <objects> ^o1</objects>
    </content>
    <rhs>
         <!-- a plural determiner -->
         <DT>
            $t_dt   <time>  </time>
                    <word> all </word>
         </DT>
 <!-- the colour of an object -->
        <colour> $t_jj <time> </time>
                       <content> $colour <colour> </colour>
                       </content>
        </colour>
 <!-- the type of object -->
 <obj_type> $t_nn <time> </time>
           <content> $obj_type <shape> </shape> <num> p </num></content>
        </obj_type>
    </rhs>
    <params>
       $dxy <dxy> 100 </dxy>
       $td  <td>  5 </td>
    </params>
    <constraints>
       <![CDATA[
             objectsMatching( $colour, $obj_type, ^o1 ) * t_follows($t_jj,$t_dt,$td) * t_follows($t_nn, $t_jj, $td) *depe
    </constraints>
</object_reference>

<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!--                                        LOCATION                                                            -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- PP phrases and things like "here"  ex: -->
<!-- (there|here) (click) -->
<!-- to the right of this box -->
<!-- in this area (area) -->
<location>
   <p_gamma>0.5</p_gamma>
   <content>
          <x> $x </x>
          <y> $y </y>
          <height> 0 </height>
          <width> 0 </width>
   </content>
   <rhs>
<DEIXIS>
    $x      <x> </x>
    $y      <y> </y>
            <left> down </left>
            <middle> </middle>
            <right> </right>
 </DEIXIS>
   </rhs>
   <params> </params>
   <constraints></constraints>
</location>

<location>
   <p_gamma>0.5</p_gamma>
   <content>
          <x> $x </x>
          <y> $y </y>
          <height> $height </height>
          <width> $width </width>
   </content>
   <rhs>
      <AREA>
            $t_area <time > </time>
            <score > </score>
    $x      <x> </x>
    $y      <y> </y>
            $width  <width> </width>
    $height <height> </height>
 </AREA>
   </rhs>
   <params> </params>
   <constraints></constraints>
</location>

<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!--                                           MULTI_CLICKS                                          -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<multi_clicks>
 <p_gamma>1.0</p_gamma>
   $t_clicks <time > </time>
             <objects> ˆo1 </objects>
   <rhs>
<DEIXIS>
    $x      <x> </x>
    $y      <y> </y>
            <left> down </left>
            <middle> </middle>
            <right> </right>
 </DEIXIS>
         <multi_clicks>
            $o2 <objects>  </objects>
         </multi_clicks>
   </rhs>
   <params>
       $dxy <dxy> 100 </dxy>
       $td  <td>  5 </td>
   </params>
   <constraints>
         <![CDATA[   objectsAt ( $x, $y, $dxy, $dxy, ˆo3 )*concat( $o2, ˆo3,ˆo1 )]]>
   </constraints>
</multi_clicks>

<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!--                                    SEMANTIC WORD GROUPS                                         -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!-- AND -->
<and> <p_gamma>1.0</p_gamma> <rhs><CC> <word> and </word></CC></rhs></and>

<!-- CREATE (synonyms) -->
<create>    <p_gamma>0.25</p_gamma> <rhs><VB> <word> create </word> </VB></rhs></create>
<create>    <p_gamma>0.25</p_gamma> <rhs><VB> <word> make </word> </VB></rhs></create>
<create>    <p_gamma>0.25</p_gamma> <rhs><VB> <word> draw </word> </VB></rhs></create>
<create>    <p_gamma>0.25</p_gamma> <rhs><VB> <word> put </word> </VB></rhs></create>

<!-- TRANSFORM (synonyms) -->
<transform>     <p_gamma>0.4</p_gamma> <rhs><VB> <word> transform </word> </VB></rhs></transform>
<transform>     <p_gamma>0.3</p_gamma> <rhs><VB> <word> turn </word> </VB></rhs></transform>
<transform>     <p_gamma>0.3</p_gamma> <rhs><VB> <word> make </word> </VB></rhs></transform>
```

```
<!-- MOVE (synonyms) -->
<move>     <p_gamma>0.5</p_gamma>  <rhs><VB> <word> move </word> </VB></rhs></move>
<move>     <p_gamma>0.5</p_gamma>  <rhs><VB> <word> put </word> </VB></rhs></move>

<!-- DELETE (synonyms) -->
<delete> <p_gamma>0.4</p_gamma> <rhs><VB> <word> delete </word> </VB></rhs></delete>
<delete> <p_gamma>0.3</p_gamma> <rhs><VB> <word> remove </word> </VB></rhs></delete>
<delete> <p_gamma>0.3</p_gamma> <rhs><VB> <word> erase </word> </VB></rhs></delete>

<!-- a (article) -->
<a>    <p_gamma>0.5</p_gamma><rhs><DT> <word> a </word></DT></rhs></a>
<a>    <p_gamma>0.5</p_gamma><rhs><DT> <word> an</word></DT></rhs></a>

<!-- here_there -->
<here_there>    <p_gamma>0.5</p_gamma><rhs><RB> <word> here </word></RB></rhs></here_there>
<here_there>    <p_gamma>0.5</p_gamma><rhs><RB> <word> there </word></RB></rhs></here_there>

<!-- COLOURS -->
<colour>   <p_gamma>0.2</p_gamma>
  <content><colour>red</colour></content><rhs> <JJ> <word>red</word></JJ> </rhs>
</colour>
<colour>   <p_gamma>0.1</p_gamma>
  <content><colour>orange</colour></content><rhs> <JJ> <word>orange</word></JJ> </rhs>
</colour>
<colour>   <p_gamma>0.1</p_gamma>
  <content><colour>yellow</colour></content><rhs> <JJ> <word>yellow</word></JJ> </rhs>
</colour>
<colour>   <p_gamma>0.2</p_gamma>
  <content><colour>green</colour></content><rhs> <JJ> <word>green</word></JJ> </rhs>
</colour>
<colour>   <p_gamma>0.2</p_gamma>
  <content><colour>blue</colour></content><rhs> <JJ> <word>blue</word></JJ> </rhs>
</colour>
<colour>   <p_gamma>0.1</p_gamma>
  <content><colour>violet</colour></content><rhs> <JJ> <word>violet</word></JJ> </rhs>
</colour>
<colour>   <p_gamma>0.1</p_gamma>
  <content><colour>violet</colour></content><rhs> <JJ> <word>purple</word></JJ> </rhs>
</colour>

<!-- SHAPES -->
<obj_type> <time> </time>    <p_gamma>0.1</p_gamma>
           <content> <shape> cube </shape> <num> s </num> </content>
           <rhs> <NN> <word> cube</word></NN> </rhs>
</obj_type>
<obj_type> <time> </time>     <p_gamma>0.1</p_gamma>
           <content> <shape> cube </shape> <num> s </num> </content>
           <rhs> <NN> <word> box</word></NN> </rhs>
</obj_type>
<obj_type> <time> </time>    <p_gamma>0.1</p_gamma>
           <content> <shape> sphere </shape> <num> s </num> </content>
           <rhs> <NN> <word> sphere</word></NN> </rhs>
</obj_type>
<obj_type> <time> </time>    <p_gamma>0.1</p_gamma>
           <content> <shape> sphere </shape> <num> s </num> </content>
           <rhs> <NN> <word> ball</word></NN> </rhs>
</obj_type>
<obj_type> <time>  </time>    <p_gamma>0.1</p_gamma>
           <content> <shape> pyramid </shape> <num> s </num> </content>
           <rhs> <NN> <word> pyramid</word></NN> </rhs>
</obj_type>
<obj_type> <time> </time>    <p_gamma>0.1</p_gamma>
           <content> <shape> cube </shape> <num> p </num> </content>
           <rhs> <NNS> <word> cubes</word></NNS> </rhs>
</obj_type>
<obj_type> <time> </time>    <p_gamma>0.1</p_gamma>
           <content> <shape> cube </shape> <num> p </num> </content>
           <rhs> <NNS> <word> boxes</word></NNS> </rhs>
</obj_type>
<obj_type> <time> </time>    <p_gamma>0.1</p_gamma>
           <content> <shape> sphere </shape> <num> p </num> </content>
           <rhs> <NNS> <word> spheres</word></NNS> </rhs>
</obj_type>
<obj_type> <time> </time>    <p_gamma>0.1</p_gamma>
           <content> <shape> sphere </shape> <num> p </num> </content>
           <rhs> <NNS> <word> balls</word></NNS> </rhs>
</obj_type>
<obj_type> <time>  </time>    <p_gamma>0.1</p_gamma>
           <content> <shape> pyramid </shape> <num> p </num> </content>
           <rhs> <NNS> <word> pyramids</word></NNS> </rhs>
</obj_type>

</CLAV_grammar>
```

REFERENCES

[1] A. Ageno and H. Rodriguez, *Extending bidirectional chart parsing with a stochastic model.*

[2] Hassan Ait-kaci and Roger Nasr, *Integrating logic and functional programming*, Lisp and Symbolic Computation **2** (1989), no. 1, 51–89.

[3] Jan Alexandersson and Tilman Becker, *Overlay as the basic operation for discourse processing in a multimodal dialogue system*, Proceedings of the 2nd IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems (Seattle, Washington), August 2001.

[4] Kikuo Asai, Noritaka Osawa, Yuji Y. Sugimoto, and Yoshiaki Tanaka, *Viewpoint motion control by body position in immersive projection display*, SAC '02: Proceedings of the 2002 ACM symposium on Applied computing (New York, NY, USA), ACM Press, 2002, pp. 1074–1079.

[5] Speech at Carnegie Mellon University group, *Jsgfgrammar, http://cmusphinx.sourceforge.net/sphinx4/javadoc/edu/cmu/sphinx/jsapi/JSGFGrammar.html*, www-url, 2004.

[6] ______, *Dragon dictation products, http://www.speech.cs.cmu.edu/comp.speech/section6/ recognition/dragon.dictation.html*, www-url, March 2006.

[7] Micha Baum, Gregor Erbach, Markus Kommenda, Georg Niklfeld, and Estela Puig-Waldmller, *Speech and multimodal dialogue systems for telephony applications based on a speech database of austrian german*, OGAI Journal, 2001.

[8] Christian Beno&#238;t and Bertrand Le Goff, *Audio-visual speech synthesis from french text: eight years of models, designs and evaluation at the icp*, Speech Commun. **26** (1998), no. 1-2, 117–129.

[9] Adam L. Berger, Stephen Della Pietra, and Vincent J. Della Pietra, *A maximum entropy approach to natural language processing*, Computational Linguistics **22** (1996), no. 1, 39–71.

[10] Sabine Bergler, Rene Witte, Michelle Khalife, Zuo Li, and Frank Rudzicz, *Using knowledge-poor coreference resolution for text summarization*, proceedings of HLT-NAACL 2003 Text Summarization Workshop (DUC 03) (Edmonton AB, Canada), 2003.

[11] Niels Ole Bernsen and Laila Dybkjær, *Evaluation of spoken multimodal conversation*, ICMI '04: Proceedings of the 6th international conference on Multimodal interfaces (New York, NY, USA), ACM Press, 2004, pp. 38–45.

[12] Guy E. Blelloch and Margaret Reid-Miller, *Fast set operations using treaps*, SPAA '98: Proceedings of the tenth annual ACM symposium on Parallel algorithms and architectures (New York, NY, USA), ACM Press, 1998, pp. 16–26.

[13] Paul Boersma, *Accurate short-term analysis of the fundamental frequency and the harmonics-to-noise ratio of a sampled sound*, Proceedings of the Institute of Phonetic Sciences of the University of Amsterdam 17, 1993, pp. 97–110.

[14] Richard A. Bolt, *"put-that-there": Voice and gesture at the graphics interface*, SIGGRAPH 80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques (New York, NY, USA), ACM Press, 1980, pp. 262–270.

[15] Yves Boussemart, François Rioux, Frank Rudzicz, Mike Wozniewski, and Jeremy Cooperstock, *A framework for 3d visualisation and manipulation in an immersive space using an untethered bimanual gestural interface*, Proceedings of the ACM Symposium on Virtual Reality Software and Technology, 2004.

[16] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal, *Pattern-oriented software architecture: a system of patterns*, John Wiley & Sons, Inc., New York, NY, USA, 1996.

[17] William Byrne, P. Beyerlein, J.M. Huerta, Sanjiv Khudanpur, B. Marthi, J. Morgan, N. Peterek, Joseph Picone, D. Vergyri, and W. Wang, *Towards language independent acoustic modeling*, ASRU, 1999.

[18] Noelle Carbonell, *Multimodal interfaces a generic design approach*, pp. 209–223, Springer, 2005.

[19] J. Cassell and S. Prevost, *Distribution of semantic features across speech & gesture by humans and machines*, 1996.

[20] Adam Cheyer and Luc Julia, *Multimodal maps: An agent-based approach*, Multimodal Human-Computer Communication, Systems, Techniques, and Experiments (London, UK), Springer-Verlag, 1998, pp. 111–121.

[21] P. R. Cohen, M. Dalrymple, D. B. Moran, F. C. Pereira, and J. W. Sullivan, *Synergistic use of direct manipulation and natural language*, SIGCHI Bull. **20**, no. SI, 227–233.

[22] Philip R. Cohen, Michael Johnston, David McGee, Sharon Oviatt, Jay Pittman, Ira Smith, Liang Chen, and Josh Clow, *Quickset: multimodal interaction for distributed applications*, MULTIMEDIA '97: Proceedings of the fifth ACM international conference on Multimedia (New York, NY, USA), ACM Press, 1997, pp. 31–40.

[23] Françoise D. Néel and Wolfgang M. Minker, *Computational models of speech pattern processing*, NATO Asi Series. Series F, Computer and Systems Sciences, vol. 169, ed. Keith Ponting, pp. 404–430, Springer, 1999.

[24] Alain Colmerauer, *Les grammaires de métamorphose GIA*, Internal publication, Groupe Intelligence artificielle, Faculté des Sciences de Luminy, Université Aix-Marseille II, France, Nov 1975. English version, Metamorphosis grammars. In L. Bolc, (Ed.), *Natural Language Communication with Computers, Lecture Notes in Computer Science 63*, Springer Verlag, Berlin, 1978, pp. 133–189, 1975.

[25] W3C World Wide Web Consortium, *Multimodal interaction activity, http://www.w3.org/2002/mmi/*, www-url, December 2005.

[26] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to algorithms*, MIT Press/McGraw-Hill, 1990.

[27] A. Corradini and P. Cohen, *Multimodal speech-gesture interface for hands-free painting on virtual paper using partial recurrent neural networks for gesture recognition*, 2002.

[28] ———, *On the relationships among speech, gestures, and object manipulation in virtual environments: Initial evidence*, Proc. of Int'l CLASS Work. on Natural, Intell. and Effective Interaction in Multimodal Dialogue Systems, 2002.

[29] Rodrigo de Luis-Garc&#237;a, Carlos Alberola-L&#243;pez, Otman Aghzout, and Juan Ruiz-Alzola, *Biometric identification systems*, Signal Process. **83** (2003), no. 12, 2539–2557.

[30] Harsha Srivatsa (IBM developerWorks), *Multimodal applications:another step in computer/human interaction, http://www-128.ibm.com/developerworks/wireless/library/wi-multimodal/*, world wide web, June 2002.

[31] John Dowding, Beth ann Hockey, Jean Mark Gawron, and Christopher Culy, *Practical issues in compiling typed unification grammars for speech recognition*, Meeting of the ACL, 2001, pp. 164–171.

[32] John Dowding, Jean Mark Gawron, Douglas E. Appelt, John Bear, Lynn Cherny, Robert Moore, and Douglas B. Moran, *GEMINI: A natural language system for spoken-language understanding*, Meeting of the Association for Computational Linguistics, 1993, pp. 54–61.

[33] Jacob Eisenstein and C. Mario Christoudias, *A salience-based approach to gesture-speech alignment*, in proceedings of HLT-NAACL 2004 (Boston, MA), 2004, pp. 25–32.

[34] Christiane Fellbaum (ed.), *WordNet: An electronic lexical database*, MIT Press, 1998.

[35] John E. Freund, *Mathematical statistics*, 5 ed., Prentice Hall, 1992.

[36] Cassandra Gilliam, *Design of advanced multimodal interfaces for mobile, collaborative and informal learning, http://fas.sfu.ca/events/event.2005-04-07.3459738687*, world wide web, April 2005.

[37] Herv Glotin, Dimitra Vergyri, Chalapathy Neti, Gerasimos Potamianos, and Juergen Luettin, *Weighting schemes for audio-visual fusion in speech recognition*, Int. Conf. Acoust. Speech Signal Process., 2001.

[38] A. G. Hauptmann, *Speech and gestures for graphic image manipulation*, Proc. ACM CHI'89 Conference on Human Factors in Computing Systems, 1989, pp. 241–245.

[39] Gerd Herzog, Alassane Ndiaye, Stefan Merten, Heinz Kirchmann, Tilman Becker, and Peter Poller, *Large-scale software integration for spoken language and multimodal dialog systems*, Nat. Lang. Eng. **10** (2004), no. 3-4, 283–305.

[40] Wolfgang Hess, *Pitch determination of speech signals: Algorithms and devices*, Springer, Berlin, Germany, 1983.

[41] H.-G. Hirsch and D. Pearce, *The AURORA experimental framework for the performance evaluation of speech recognition under noisy conditions*, Proceedings of the ISCA ITRW ASR, 2000.

[42] Graeme Hirst, *Discourse-oriented anaphora resolution in natural language understanding: a review*, Comput. Linguist. **7** (1981), no. 2, 85–98.

[43] Hartwig Holzapfel, Kai Nickel, and Rainer Stiefelhagen, *Implementation and evaluation of a constraint-based multimodal fusion system for speech and 3d pointing gestures*, ICMI '04: Proceedings of the 6th international conference on Multimodal interfaces (New York, NY, USA), ACM Press, 2004, pp. 175–182.

[44] JE Hopcroft and JD Ullman, *Introduction to automata theory, languages, and computation*, Addison-Wesley, Reading, Massachussetts, 1979.

[45] Xuedong Huang, Alex Acero, and Hsiao-Wuen Hon, *Spoken language processing: A guide to theory, algorithm, and system development*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001, Foreword By-Raj Reddy.

[46] Juan M. Huerta, *Speech recognition in mobile environments*, Ph.D. thesis, Carnegie Mellon University. Department of Electrical and Computer Engineering, 2000.

[47] Don Johnson and Sinan Sinanovic, *Symmetrizing the kullbackleibler distance*.

[48] Mark Johnson, *Learning and parsing stochastic unification-based grammars.*, COLT, 2003, pp. 671–683.

[49] Michael Johnston, *Unification-based multimodal parsing*, Proceedings of the 36th annual meeting on Association for Computational Linguistics (Morristown, NJ, USA), Association for Computational Linguistics, 1998, pp. 624–630.

[50] Michael Johnston and Srinivas Bangalore, *Finite-state multimodal parsing and understanding*, Proceedings of the 18th conference on Computational linguistics (Morristown, NJ, USA), Association for Computational Linguistics, 2000, pp. 369–375.

[51] ______, *Finite-state methods for multimodal parsing and integration*, ESSLLI Workshop on Finite-state Methods, ESSLLI, 2001.

[52] Daniel Jurafsky and James H. Martin, *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[53] Martin Kay, *Functional grammar*, Proceedings of the 5^{th} Annual Meeting of the Berkeley Linguistics Society (C. Chiarello et al., ed.), 1979, pp. 142–158.

[54] P. Kay and C. Fillmore, *Grammatical constructions and linguistic generalizations: the what's x doing y*, 1999.

[55] Adam Kendon, *Gesture and speech: Two aspects of the process of utterance*, Nonverbal Communication and Language, pp. 207–227, 1980.

[56] Sanshzar Kettebekov and Rajeev Sharma, *Toward natural gesture/speech control of a large display*, EHCI '01: Proceedings of the 8th IFIP International Conference on Engineering for Human-Computer Interaction (London, UK), Springer-Verlag, 2001, pp. 221–234.

[57] Sanshzar Kettebekov, Mohammed Yeasin, Nils Krahnstoever, and Rajeev Sharma, *Prosody based co-analysis of deictic gestures and speech in weather narration broadcast*, Workshop on Multimodal Resources and Multimodal System Evaluation. (LREC 2002), 2002, pp. 57–62.

[58] Donald E. Knuth, *Sorting and searching*, The Art of Computer Programming, vol. 3, Addison-Wesley, 1973.

[59] S. Kopp, P. Tepper, K. Ferriman, and J. Cassell, *Trading spaces: How humans and humanoids use speech and gesture to give directions*, Spatial Cognition and Computation **(in press)** (2006).

[60] Robert M. Krauss, *Why do we gesture when we speak?*, Current Directions in Psychological Science **7** (1998), 54–59.

[61] Hong-Kwang Jeff Kuo, Eric Fosler-Lussier, Hui Jiang, and Chin-Hui Lee, *Discriminative training of language models for speech recognition*, Proc. of International Conference on Acoustics, Speech and Signal Processing (Orlando, Florida), 2002.

[62] Paul Lamere, Philip Kwok, Evandro Gouvea, Bhiksha Raj, Rita Singh, William Walker, M. Warmuth, and Peter Wolf, *The cmu sphinx-4 speech recognition system*, IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP 2003) (Hong Kong), Apr 2003.

[63] Paul Lamere, Philip Kwok, William Walker, Evandro Gouvea, Rita Singh, Bhiksha Raj, and Peter Wolf, *Design of the cmu sphinx-4 decoder*, Proceedings of the 8th European Conf. on Speech Communication and Technology (EUROSPEECH 2003), 2003.

[64] Susan J. Lederman, Roberta L. Klatzky, T. Morgan, and C. Hamilton, *Integrating multimodal information about surface texture via a probe: Relative contributions of haptic and touch-produced sound sources.*, Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2002, pp. 97–104.

[65] Kai-Fu Lee, *Automatic speech recognition: the development of the sphinx system*, Kluwer Academic, Boston, MA, 1989.

[66] Juergen Luettin, Neil A. Thacker, and Steve W. Beet, *Active shape models for visual speech feature extraction*, Speechreading by Humans and Machines (D. G. Storck and M. E. Hennecke (editors), eds.), NATO ASI Series, Series F: Computer and Systems Sciences, vol. 150, Springer Verlag, Berlin, 1996, pp. 383–390.

[67] Christopher D. Manning and Hinrich Schü tze, *Foundations of statistical natural language processing*, MIT Press, Cambridge, MA, USA, 2002.

[68] David McNeill, *Hand and mind: What gestures reveal about thought*, University of Chicago Press and CSLI Publications, Chicago, Illinois, 1992.

[69] Robert C. Moore, *Removing left recursion from context-free grammars*, Proceedings of 1st Annual Conference of the North American Chapter of the ACL, 2000.

[70] Ezequiel Morsella and Robert M. Krauss, *The role of gestures in spatial working memory and speech*, American Journal of Psychology **117** (2004), 411–424.

[71] Petr Motlcek, Luks Burget, and Jan Cernock, *Visual features for multimodal speech recognition*, Proceedings of Radioelektronika 2005, Fakulta elektrotechniky a komunikacnch technologi VUT, 2005.

[72] P. Muller, *Modular specification and verification of object-oriented programs*, Ph.D. thesis, FernUniversitat Hagen, 2001.

[73] J. G. Neal and S. C. Shapiro, *Intelligent multimedia interface technology*, Intelligent User Interfaces (J. W. Sullivan and S. W. Tyler, eds.), Frontier Series, ACM Press, New York, 1991.

[74] C. Neti, G.Iyengar, G. Potamianos, A. Senior, and B. Maison, *Perceptual interfaces for information interaction: Joint processing of audio and visual information for human-computer interaction*, Proc. of the International Conference on Spoken Language Processing (ICSLP'2000) (Beijing, China), 2000, pp. 11–14.

[75] Nils J. Nilsson, *Artificial intelligence: a new synthesis*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[76] Douglas O'Shaughnessy, *Modelling fundamental frequency, and its relationship to syntax, semantics, and phonetics*, Ph.D. thesis, Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science, 1976.

[77] ______, *Speech communications - human and machine*, IEEE Press, New York, NY, USA, 2000.

[78] Sharon Oviatt, *Multimodal interfaces*, pp. 286–304, Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA, 2003.

[79] Sharon Oviatt, Antonella DeAngeli, and Karen Kuhn, *Integration and synchronization of input modes during multimodal human-computer interaction*, CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems (New York, NY, USA), ACM Press, 1997, pp. 415–422.

[80] Sharon Oviatt and Karen Kuhn, *Referential features and linguistic indirection*, Multimodal Language, 1998.

[81] Sharon Oviatt and Robert vanGent, *Error resolution during multimodal human-computer interaction*, Proceedings of the International Conference on Spoken Language Processing, 1996.

[82] V. I. Pavlovic, G. A. Berry, and T. S. Huang, *Integration of audio/visual information for use in human-computer intelligent interaction*, ICIP '97: Proceedings of the 1997 International Conference on Image Processing (ICIP '97) 3-Volume Set-Volume 1 (Washington, DC, USA), IEEE Computer Society, 1997, p. 121.

[83] Vladimir Pavlovic, Rajeev Sharma, and Thomas S. Huang, *Visual interpretation of hand gestures for human-computer interaction: A review*, IEEE Transactions on Pattern Analysis and Machine Intelligence **19** (1997), no. 7, 677–695.

[84] Vladimir I. Pavlovic and Thomas S. Huang, *Multimodal prediction and classification on audio-visual features*, Workshop on Representations for Multi-modal Human Computer Interaction (Menlo Park, CA), 1998, pp. 55–59.

[85] P. Peer and F. Solina, *An automatic human face detection method*, Proceedings of Computer Vision Winter Workshop (Rastenfeld, Austria) (N. Brandle, ed.), 1999, pp. 122–130.

[86] Carl Pollard and Ivan A. Sag, *Head-driven phrase structure grammar*, University of Chicago Press and CSLI Publications, Chicago, Illinois, 1994.

[87] The NTP Project, *Ntp: The network time protocol, http://www.ntp.org/*, www-url, February 2005.

[88] Derek Proudian and Carl Pollard, *Parsing head-driven phrase structure grammar*, Proceedings of the 23rd annual meeting on Association for Computational Linguistics (Morristown, NJ, USA), Association for Computational Linguistics, 1985, pp. 167–171.

[89] Adwait Ratnaparkhi, *Maximum entropy models for natural language ambiguity resolution*, Ph.D. thesis, University of Pennsylvania, 1998.

[90] Thomas Richardson, *A polynomial-time algorithm for deciding markov equivalence of directed cyclic graphical models*, Proceedings of the 12th Annual Conference on Uncertainty in Artificial Intelligence (UAI-96) (San Francisco, CA), Morgan Kaufmann Publishers, 1996, pp. 462–469.

[91] Vitor Rocio and José Gabriel Lopes, *Partial parsing, deduction and tabling.*, TAPD, 1998, pp. 52–61.

[92] Kenneth H. Rosen, *Discrete mathematics and its applications (2nd ed.)*, McGraw-Hill, Inc., New York, NY, USA, 1991.

[93] P. Rubin, E. Vatikiotis-Bateson, and C. Benoit, *Audio-visual speech processing [special issue]*, Speech Communication **26** (1998).

[94] Frank Rudzicz, *Put a grammar here: Bi-directional parsing in multimodal interaction*, Proceedings of the CHI 2006 Extended Abstracts, 2006.

[95] Stuart J. Russell and Peter Norvig, *Artificial intelligence: A modern approach*, Pearson Education, 2003.

[96] Eric D. Sandness, *Discriminative training of acoustic models in a segment-based speech recognizer*, Master's thesis, Massachussetts Institute of Technology. Department of EECS, 2000.

[97] Giorgio Satta and Oliviero Stock, *Bidirectional context-free grammar parsing for natural language processing*, Artif. Intell. **69** (1994), no. 1-2, 123–164.

[98] Raimund Seidel and Cecilia R. Aragon, *Randomized search trees*, Algorithmica **16** (1996), no. 4/5, 464–497.

[99] Clifford A. Shaffer, *A practical introduction to data structures and algorithm analysis*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.

[100] R. Sharma, M. Yeasin, N. Krahnstoever, I. Rauschert, G. Cai, A. MacEachren, K. Sengupta, and I. Brewer, *Speech-gesture driven multimodal interfaces for crisis management*, Proceedings of IEEE special issue on Multimodal Human-Computer Interface, 2003.

[101] Rajeev Sharma, Jiongyu Cai, Srivat Chakravarthy, Indrajit Poddar, and Yogesh Sethi, *Exploiting speech/gesture co-occurrence for improving continuous gesture recognition in weather narration*, FG '00: Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition 2000 (Washington, DC, USA), IEEE Computer Society, 2000, p. 422.

[102] G. Skantze, *Coordination of referring expressions in multimodal human-computer dialogue*, 2002.

[103] Kay Stanney, Shatha Samman, Leah Reeves, Kelly Hale, Wendi Buff, Clint Bowers, Brian Goldiez, Denise Nicholson, and Stephanie Lackey, *Functional optical brain imaging using near-infrared during cognitive tasks*, International Journal of Human-Computer Interaction **17** (2004).

[104] IMS Stuttgart, *The penn treebank tag set, http://www.ims.uni-stuttgart.de/projekte/corpusworkbench/cqp-htmldemo/penntreebankts.html*, www-url, 1998.

[105] Satoshi Tamura, Koji Iwano, and Sadaoki Furui, *A robust multi-modal speech recognition method using optical-flow analysis*, IDS-2002, 2002.

[106] ______, *Multi-modal speech recognition using optical-flow analysis for lip images*, J. VLSI Signal Process. Syst. **36** (2004), no. 2-3, 117–124.

[107] Music TMH, Speech and Hearing, *Wavesurfer, http://http://www.speech.kth.se/wavesurfer/*, www-url, February 2006.

[108] Masaru Tomita, *An efficient context-free parsing algorithm for natural languages and its applications*, Ph.D. thesis, 1985.

[109] M. Turk and M. Kolsch, *Perceptual interfaces*, 2004.

[110] Carnegie Mellon University, *Sphinx-4 a speech recognizer written entirely in the javatm programming language (http://cmusphinx.sourceforge.net/sphinx4/)*, www-url, September 2005.

[111] Dimitra Vergyri, Stavros Tsakalidis, and William Byrne, *Minimum risk acoustic clustering for multilingual acoustic model combination*, Proc. ICSLP, vol. 3, 2000, pp. 873–876.

[112] Minh Tue Vo, *A framework and toolkit for the construction of multimodal learning interfaces*, Ph.D. thesis, 1998, Chair-Alex Waibel.

[113] W3C.

[114] Ipke Wachsmuth, *Communicative rhythm in gesture and speech*, Lecture Notes in Computer Science **1739** (1999), 277–N/A.

[115] Wolfgang Wahlster, *Planning multimodal discourse*, Proceedings of the 31st annual meeting on Association for Computational Linguistics (Morristown, NJ, USA), Association for Computational Linguistics, 1993, pp. 95–96.

[116] ———, *Smartkom: Symmetric multimodality in an adaptive and reusable dialogue shell*, Proceedings of the Human Computer Interaction Status Conference 2003, 2003, pp. 47–62.

[117] Kenneth Wauchope, *Two multimodal interfaces to military simulations*, Proceedings of the fifth conference on Applied natural language processing (San Francisco, CA, USA), Morgan Kaufmann Publishers Inc., 1997, pp. 21–22.

[118] Kent Wittenburg, *F-patr: functional constraints for unification-based grammars*, Proceedings of the 31st annual meeting on Association for Computational Linguistics (Morristown, NJ, USA), Association for Computational Linguistics, 1993, pp. 216–223.

[119] Wai Yu and Stephen A. Brewster, *Comparing two haptic interfaces for multimodal graph rendering.*, Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2002, pp. 3–9.

[120] Shlomo Zilberstein, *Using anytime algorithms in intelligent systems*, AI Magazine **17** (1996), no. 3, 73–83.

[121] G. K. Zipf, *Human behavior and the principle of least-effort*, Addison-Wesley, Cambridge, MA, 1949.

Document Log:

Manuscript Version 0

Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-LaTeX — 28 September 2006

FRANK RUDZICZ

CENTRE FOR INTELLIGENT MACHINES, MCGILL UNIVERSITY, 3480 UNIVERSITY ST., MONTRÉAL (QUÉBEC) H3A 2A7, CANADA, *Tel.* : (514) 398-8200

E-mail address: `frudzi@cim.mcgill.ca`