NORTHWESTERN UNIVERSITY

HIERARCHICAL PARALLEL MARKOV MODELS FOR INTERACTIVE SOCIAL AGENTS

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

by

ROBERT ZUBEK

EVANSTON, ILLINOIS

June 2005

Abstract

Hierarchical Parallel Markov Models for Interactive Social Agents

Robert Zubek

In this report I present hierarchical parallel Markov models for the creation of interactive social agents for video games and entertainment.

The approach extends existing, popular character technologies for social, communicative interaction. First, adding the knowledge of temporal interaction structure enables natural language interaction on a time scale much longer than current chatterbot technologies. Second, adding support for hierarchical interaction decomposition, where an interaction is represented as a collection of smaller, simpler elements, simplifies the authoring of complex engagement. Third, adding support for the concurrent engagement of these elements enables engagement in interleaved, naturalistic communication.

The resulting decomposition supports redundancy of representation, graceful performance degradation through the simultaneous engagement of behaviors on different levels of abstraction, and the stochastic approximation mechanism increases robustness in the face of noise and ambiguity.

In this report, I present the details of hierarchical parallel Markov models, I examine two entertainment agents that use this technique, and explain the implementational details of how such an approach can be used in the development of future systems.

Rodzicom, Katarzynie i Henrykowi.

Przez nich wszystko się stało

Co się stało.

Acknowledgements

> *Albowiem nie szukamy tego co doskonałe, szukamy tego, co zostaje z nieustannego dążenia. / Pamiętając, ile znaczy traf szczęśliwy, zbieg słów i okoliczności, ranek z obłokami, który później wyda się nieuchronny.*
>
> For we do not seek perfection, we seek that, which remains from endless pursuit. / Remembering how important is the stroke of luck, the coincidence of words and events, the morning with puffy clouds, that will later seem inevitable.
>
> — Czesław Miłosz, *Do Alana Ginsberga.*

I have been lucky to be surrounded by great people. It is thanks to you that I first noticed what only later became significant, and first started working on what would eventually consume my passions. Who I am, I owe to you.

First, I would like to thank my research group, a spectacular set of people whose company I will miss greatly: Ian Horswill, my advisor, a source of great inspiration over the decade I've known him, and whose support for my unorthodox research I will always appreciate; Robin Hunicke, who taught me all I know about game design; Aaron Khoo, an astute partner in research as well as gaming; Magy Seif El-Nasr, whose passion

for interactive characters fueled my own; and Dac Le, Cuong Pham, and other students with whom I worked on robots and entertainment technologies.

Many thanks go the members of the computer science department, especially Ken Forbus, who provided endless support and advice not only in my research work, but also in my undergraduate education, and Andrew Ortony, whose work on emotion was a great motivation, and who got me started on interactive characters in the first place.

My friends inside and outside of the department made the entire experience immensely enjoyable, and I appreciate their advice, camaraderie, and indulging me in frequent shop talk about various items of geeky interest: Praveen Paritosh, whose thirst for knowledge was completely contagious; David Ayman Shamma, the rare exemplar of a scientist with artistic abilities, and an artist with scientific insight; Jason Skicewicz, who never sweats the small stuff; and Pinku Surana, who inspired me to see the world as full of solutions just waiting to be discovered. Special thanks go to Anna Szuber, for sharing her linguistic insight, which I will never be able to match; and former housemates Mark DePristo and Matt Hubenschmidt, who made grad school experience so enjoyable.

Finally the greatest thanks go to my parents, for their unending support, and to my partner Matt, for his inspiration and encouragement, for keeping me sane and, really, for putting up with me for so many years. You were right—I miss it all already.

Table of Contents

Table of Figures

Table of Tables

Chapter 1. Introduction.

Wierzcie mi: nic bardziej pożądanego,

a nic trudniejszego na ziemi

jak prawdziwa rozmowa.

Believe me: there is nothing more desired,

and nothing more difficult on earth

than true conversation.

— Adam Mickiewicz

1.1. Motivation and Overview

Game characters connect the player to the game world. They present stories, introduce plot elements, compete with players, amuse them, assist them, and work to immerse them in the fiction of the game. Like actors in theatre, they are presenters of the coherent reality of an imaginary world. Their appeal depends on the believability of their performance—on the successful presentation of themselves as living inhabitants of the game world, reacting sensibly and intelligently to what happens around them.

Imagine characters in future games—how might they behave? They should certainly be social, conversational, and fully interactive, they should be able to chat with the player about what goes on in the game world, and engage in interactions that players enjoy with each other. They should behave like believable citizens of the game world, playing the roles of player's snubbed enemies and lost loves, of frustrated friends and cautious customers, of enthusiastic street vendors and impatient strangers. The desire for such interactive characters, making the game world a living, breathing space, is at the core of this dissertation.

This report presents an approach to adding more knowledge to existing techniques for social interaction with the player. Games are filled with interaction—players chat with each other, talk about events in the game, exchange information about the world,

gossip about others, coordinate trade, and so on. Characters would do well to tap into these types of interaction.

However, the computer entertainment domain imposes difficult constraints on the techniques used to achieve such interaction. Techniques for interaction must support believable behavior within the game context—that is to say, they must make the character appear like a reasonable resident of the game world, supporting the illusion that is it more than just an automaton. They must also support easy behavior authoring—such that very specific behaviors and aesthetic effects can be developed reliably for a wide range of possible player activity, and on a shortened time scale characteristic of commercial product development. Finally, the techniques must be efficient—such that they can run on a fraction of processing cycles available on commodity hardware.

Finite-state techniques are, to this day, the most common class of approaches for interaction modeling. Game developers remain loyal to finite-state techniques, because they simplify authoring and encourage efficient implementation—their representation of behavior is straightforward, and therefore accessible and easy to implement.

In particular, popular techniques for social interaction and communication tend to fall into two general categories: finite-state “dialog trees”, which represent some of the structure of the interaction, but disallow free-form input, and “chatterbots”, which

support natural language text, but have very little knowledge of the long-term structure of the interaction.

Unfortunately, it can be quite difficult to represent efficiently the extended temporal structure of social interaction. First, player behavior can be noisy, ambiguous, full of error, made up of several threads of conversation being interleaved at the same time—and the finite-state system must be able to deal with the errors, the difficulties, and the concurrency as a fact of life. Second, the interaction can be quite complex, and the finite-state representation must support such complexity without devolving into an authoring nightmare.

This dissertation is about extending existing approaches to support these kinds of phenomena. First, adding the knowledge of temporal interaction structure, to enable natural language interaction on a time scale much longer than current chatterbot technologies. Second, adding support for hierarchical interaction decomposition, such that the interaction could be represented as a collection of smaller, simpler elements, combined to reproduce the complex engagement. Third, adding support for the concurrent engagement of these elements, to support the interleaved and unordered nature of naturalistic human communication.

Interaction structure is approached by recasting interaction as a stochastic process, and using stochastic models such as hidden Markov models (HMMs) to track the

development of this process. Stochastic models perform very robustly in the face of ambiguous and noisy inputs.

The overall interaction can be further decomposed into simpler elements, corresponding to individual sub-structures of the interaction, modeled as multiple separate HMMs. By running all the HMMs in parallel, the system is able to support a number of concurrent threads of conversation, whose productions are allowed to interleave in arbitrary ways.

Finally, in order to replicate the ordered performance of the complex interaction, the dependencies between the elements need to be made explicit. These dependencies can be implemented very efficiently, via a reduction to state coupling and then evidence estimation.

The resulting *hierarchical parallel* HMMs allow for robust engagement in composite, interleaved interactions. The advantages of the resulting technique are multiple. It supports decomposition of interaction into smaller elements, which simplifies authoring. It supports the simultaneous engagement of elements on different levels of abstraction, which enables redundancy, fallback, and graceful performance degradation. It is stochastic, allowing for greater robustness in the face of noise and ambiguity. Finally, the representation remains finite-state, retaining structural benefits so important in artifact production.

In the remainder of this chapter, I examine the constraints of game development that make finite-state techniques appealing. Subsequently, I describe the hierarchy and parallelism requirements in detail, and additional details of the hierarchical parallel approach.

1.2. Domain Constraints

Entertainment production presents a daunting set of constraints, imposed both by the way games are used by the players, and by the game development process itself. These constraints are slightly askew from what is commonly expected in general artificial intelligence, or even general computer science, as the end result is an entertainment title, whose value is in the aesthetic experience it produces.

The constraints can be seen as falling into three broad classes. The development process requires that interactive characters be easy to author and their behavior easy to debug. At the same time, interactive characters ought to be more than simple automata, but rather believable performers of their given roles—they must act believably, just like cartoon characters behave believably in spite of being recognizably unreal. Finally, the resulting techniques have to be computationally inexpensive, functioning on a fraction of an impoverished CPU.

1.2.1. AUTHORING

Game development places great emphasis on the aesthetics of the experience. However, unlike in other forms of entertainment, the game designer has astonishingly little control over the final performance: games allow the player to influence the experience, and the designer has to give the player enough freedom to be entertained, while at the same time preventing any behavior that could lead to unpleasant or undesirable results.

Developers require tight authorial control over the gameplay, and routinely have to go back and forth between desirable experience and an implementation that will produce it. Game characters have to be 'debugged' not merely in terms of their computational correctness, but primarily with respect to the effect their behavior has on the user. For example, after engaging the agent for a while, the developer might say: this agent is coming off too strongly; I need to make him appear more nonchalant, and more trustworthy. So the developer will need to open the hood, and add or debug some particular behaviors or turns of phrase that will convey the right sentiment, then test the behavior again, and continue debugging until the right effect is achieved. Techniques used in game development must support this: make it easy to produce a computational representation that will have desirable runtime behavior, and allowing one to predict the runtime behavior given the representation at hand.

We can analyze this problem of authoring more formally using the Mechanics/Dynamics/Aesthetics (MDA) model of game design (LeBlanc 2003, 2004, Hunicke, LeBlanc, and Zubek, 2004). We can consider game design and development as the simultaneous engagement of three different levels of abstraction. On the lowest level, a game is implemented as a set of *mechanics:* rules of the game and computer code that embodies them. When interacting with autonomous players, and with each other, the mechanics give rise to system *dynamics:* the runtime behavior of the pieces that make up the system. Finally, the dynamics engaged over time lead to particular *aesthetics* of the player experience: some interesting and enjoyable impressions resulting from interaction with the game's behavior.

The process of game design usually involves back-solving from top to bottom: starting from some notion of the desirable aesthetic experience, then designing a dynamic system that results in these aesthetics, and then creating a set of mechanics that will produce this dynamic behavior. Unsurprisingly, design decisions propagate across layers—as the aesthetic design evolves, it necessitates modifications of the behavior, which requires changes to the mechanics, which usually changes some other behaviors as a side effect, which reverberates back into the aesthetics, and so on. The game's designer must find a good solution to this highly constrained system of mutually dependent effects. However, the designer cannot shape the dynamics or aesthetics directly—only the mechanics are available for manipulation.

In development practice, the problem is often made tractable by constraining the system. Developers adopt constrained techniques that make the mechanic-dynamic link easy to design and debug, which lets them concentrate on resolving just the dynamic-aesthetic interactions.

Finite-state machines and menu trees are popular because of their relatively transparent mechanic-dynamic mapping, because their inner workings are intuitive and easy to map to runtime behavior. As Woodcock (1998b) puts it, finite-state systems are "generally predictable and hence easy to test", which makes them highly attractive.

1.2.2. Believability

The second requirement is believability: characters must behave like reasonable living creatures, acting attentively and intelligently given the fiction of the virtual world. The corollary is that characters do not have to be realistic, but their behavior must be consistent with the fiction.

This is a truism of game production, and the practice of design routinely refers to it. As one designer puts it:

> Another potential AI programming pitfall is creating an AI which, though it actually performs like a "real" person, ends up detracting from the gameplay as a result. In terms of the stories they tell and the settings they employ, games are

> often contrivances, strictly unreal situations that are specifically set up because they are interesting, not because they are authentic, and the AI must support this. [. . .] If the AI is really very sophisticated but, as a result, the game is unplayable or extremely frustrating, a player is not going to remark on how smart the AI is. (Rouse 2001: 171-2)

Concern with believability entered artificial intelligence via animation (Thomas and Johnston 1981, Bates, Loyall, and Reilly 1991), from the description of techniques used to convey emotion, intention, and reason in animated characters. It can be described as selective non-realism: to make a character appear believable, one must not try to make the behavior realistic, but rather exaggerate certain important aspects, while ignoring others, to let the viewer's imagination fill in the rest. This is related to the *uncanny valley* principle of anthropomorphic design (Mori 1982), which suggests that iconic or caricatured behavior is much more appealing than imperfect realism.

The approach gained influence in AI largely due to the work of the Oz group, which introduced the idea of believable agents as a basis for a number of systems (for an excellent overview see Mateas 1997), and similar concerns in the animate agents community. The term is usually used to describe a stance, rather than a precise set of requirements, although in one of the Oz publications, Loyall does attempt to present a number of desirable elements of believability, from very specific such as "concurrent pursuit of goals", to broad, such as "social relationships" (Loyall 1997: 27).

Unfortunately, believability is highly problematic in an interactive medium. Agent designers cannot mimic the approach animators or writers take to create believable characters; believability in film involves capturing some desirable elements of activity, and freezing it in the medium. Games, however, are fully interactive—the player is inherently unpredictable, and the agent must act appropriately *in spite* of the player's potentially problematic behavior. Agent designers are not merely scripting believable behavior—they must design entire systems whose dynamic behavior turns out believably, regardless of what the player does. The developer's task is to proceduralize the complex characteristics of believable performance into algorithms and knowledge representation, such that their runtime behavior is robustly believable.

1.2.3. Efficiency

Game agents must be reasonable and responsive to the world around them. At the same time, they are allowed very little CPU time into which to squeeze the necessary computation. This means the system must be very efficient at what it does.

The popular intuition among practitioners is that out of the overall CPU time available to the game, the AI subsystem, broadly construed, receives less than 25%, and most often only around 10%. By "AI broadly construed", I mean a mélange of behavior control and other components that, while important to character performance, are not usually considered as components of artificial intelligence as such: physics calculations,

animation control, world model consistency maintenance, and so on. Furthermore, the processor time allowance applies to the entire AI subsystem, and since a game will typically employ a number of characters, each character's individual behavior control can only expect a fraction of this allowance.

The combination of processing poverty, and demand for reactive believable behavior, leads to considerable difficulties. Gameplay enforces soft-real-time performance constraints on behavior, because untimely reaction to the dynamically changing world will damage the illusion of intelligence. But this reactive, dynamic intelligence must be achieved using the minimum number of processing cycles, demanding exceedingly inexpensive techniques.

Unfortunately, it is difficult to obtain hard data about the exact processing allowance. First, as we mentioned, the 'AI system' usually includes much more than character control, but the usage of the term varies between developers and studio cultures—for instance, some developers count animation as part of AI, some do not. Second, variance across genres is considerable—for example, AI takes a tiny slice of processor time in arcade-style shooters, but a lion's share in turn-based strategy games. Third, this information is based on informal surveys (Woodcock 1998, 2000, 2002, Kirby 2004), since reliable data are simply not available.

Even give these difficulties, however, it remains interesting that the reported AI's share of the main processor remained stable over the years, even in spite of the popularization of additional processing hardware such as programmable video cards. Furthermore, as games develop, more and more components crowd under the AI umbrella—most recently, elements of physical simulation and animation control. This suggests that even though the raw number of cycles available to the AI will keep increasing, so will the demands placed on them.

1.2.4. Constraints Reconsidered

These constraints highlight an important problem in games-related research: the evaluation criteria for game technologies differ somewhat from those typical of artificial intelligence research, where authoring and believability concerns are uncommon, and efficiency is often of secondary interest.

However, in games, these enjoy great prominence. Authoring is important because designers need to create consistent and predictable behavior. Believability is important because it enables a wide range of aesthetic experience. And efficiency is important because games have to run on inexpensive commodity platforms and underpowered handheld devices.

1.3. Hierarchy and Parallelism

Two broad classes of techniques are popularly used in the implementation of interactive, social characters. Pattern-matching production systems, à la Eliza, allow for interaction via free-form text utterances, but retain practically no information about the larger structure of the interaction. Conversation trees, on the other hand, are similar to finite-state dialog systems; they represent structure in detail, but disallow free-form inputs. Their abilities to express complex behavior are limited.

As mentioned earlier, this work concentrates on extending these approaches, to allow for both natural language input, and the knowledge of temporal structure of interaction. This will be achieved by representing the interaction using stochastic finite-state models, extended with novel mechanisms for dealing with complexity via hierarchical and parallel decomposition.

1.3.1. Hierarchy and Parallelism in Interaction

Stochastic finite-state representations, such as hidden Markov models, are a popular method for tracking stochastic processes. A hidden Markov model (HMM) can be considered a state space, in which both the position and state transitions are probabilistic. Transitions are treated as probabilities of moving from state to state, and position is replaced with a *probability distribution* of the possible position likelihoods over the entire state space. I will examine these in detail in Chapter 3—it will suffice to

say for now that they retain the structural simplicity of standard finite-state models, but behave more robustly in the face of noise and input uncertainty.

Finite-state interaction models allow for easy, structural description of the activity itself. Their simplicity enables very efficient implementation, and simplifies authoring by supporting greater system transparency—the developer can predict much about the system's runtime behavior given the computational representation, as well as design a computational representation that will result in some particular behavior.

I would like to concentrate on two properties that are desirable in finite state models, but rarely expressed together in finite-state systems: the properties of hierarchical decomposability, and of independent parallel engagement.

Rich, believable communication requires engagement in different social protocols, and broad competence in human conversational moves. Implementing such a system using a single, flat dialog space would be an authoring nightmare: finite-state models with non-trivial numbers of states are difficult to design and debug. Rather, it is desirable to decompose the overall system into simpler elements, which could be combined hierarchically, in a tree or some other topology, such that their ordered performance reconstructs that of the original. This should be uncontroversial—decomposition is the standard approach for modeling complex phenomena.

At the same time, communication is not a simple, clean composite of the elements. Many different activities and threads of conversation can be engaged at the same time, interleaved or overlapped in the course of an interaction. Systems should be able to deal gracefully with a large variety of player moves and behaviors, if only to steer the interaction back to the topic at hand; they must recover from failure and error by attempting to deal with the situation as a human might. The system should be able to support numerous elements acting at the same time.

1.3.2. The Hierarchy Requirement

It is desirable to manage system complexity by introducing structural abstraction: decomposing interaction into separate modules, which can control each other. Consider a typical fantasy role-playing game. When building a shopkeeper agent, for example, we can decompose a *selling interaction* into threads of deciding on an item, evaluating the item, haggling about the price, and so on; the *evaluation* thread can consist of praising the workmanship, pointing out features, or defusing criticism, *praising* the workmanship can then finally bottom out in particular speech acts.

A hierarchical decomposition is beneficial in numerous ways. Such a component hierarchy, as illustrated above, allows for an economical representation and code reuse, since the individual modules can be written separately, as well as reused across

different agents. At the same time, each of the simple elements would also be efficient to compute.

A natural implementation of such decomposition would be as a linked hierarchy of elements that transfer control between each other: the root of the interaction relays control to a child, which calls another child, which finally accepts and produces speech acts, transferring control to another child, or one of its ancestors, and so on.

These constituent elements can be represented with separate stochastic models, used as building blocks of the overall interaction. This improves robustness and aids in code reuse: the interaction becomes a matter of sequencing, building up larger interactions out of smaller ones. Readers familiar with the problem of action planning may find this approach quite natural.

When such a decomposable interaction is used, the different levels must be able to influence each other's evolution; for example, evaluation requires the focus to be settled, then it can be engaged in several different ways, and only once evaluation is finished can haggling commence. The parent processes must be able to interact with the children, and the children with the parents. This requires that the different processes be causally interdependent; that the parent processes be able to influence child processes, and vice versa, enabling each other, or affecting their productions in some desirable manner.

1.3.3. The Parallelism Requirement

Social protocols can also be engaged concurrently; for example, the player may switch back and forth between the haggling subinteraction and the evaluation subinteraction, advancing them both independently on different time scales. Models that track them must be capable of changing state independently and asynchronously. The finite state representation must allow for numerous, parallel but interdependent protocols.

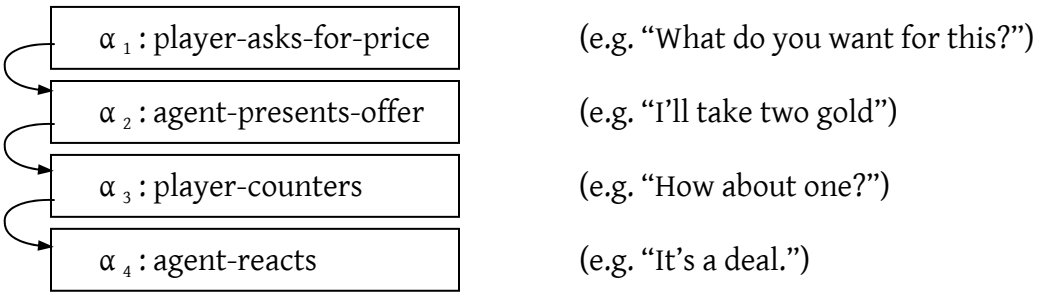
For a more concrete example, consider an interleaving of two independent models, for a character that sells items to the player. First, we may have a finite-state model of how to haggle about an item price. The model has some number of states arranged in some specific topology, perhaps one that resembles space **A** in Figure 1-1.

But the agent should also be business-savvy, and so it should try to praise the item before the player, and defuse any undesirable reactions. This is a completely different mini-interaction, and can be described using its own state space; perhaps it has a state space akin to **B** from the same figure.

In this simple example we assume that these particular models are linear: each production α_n only leads to α_{n+1}. Actual models do not have to be linear, of course.

Nota bene: the following state spaces are simple and linear to make the example easy to follow; the discussion is not limited to linear spaces, of course.

A: Haggling about the price:



B: Praising an item:

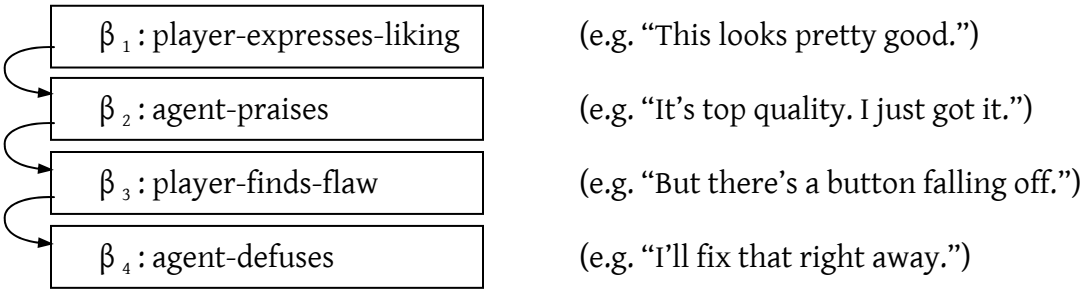


Figure 1-1. Sample spaces A and B and their individual productions.

If one were to join these two into a simple hierarchical system, only one of the spaces would be active at any time; such a hierarchical system could concatenate or nest the two sub-interactions. But human interaction are not so orderly—both protocols could easily be engaged at the same time, interleaved arbitrarily, producing interactions that perhaps look like the following:

$\alpha_1 \alpha_2 \beta_1 \alpha_3 \beta_2 \beta_3 \alpha_4 \beta_4$, or

$\alpha_1 \beta_1 \alpha_2 \alpha_3 \alpha_4 \beta_2 \beta_3 \beta_4$, or

$\alpha_1 \alpha_2 \beta_1 \beta_2 \beta_3 \alpha_3 \beta_4 \alpha_4$, etc.

However, in addition to correct interleavings, a system must *not* produce invalid exchanges that violate the orderings of the protocols. The following examples present invalid exchanges, with out-of-order productions ~~crossed out~~:

$* \alpha_1 \beta_1 \sout{\alpha_1} \beta_2 \alpha_2 \beta_3 \beta_4 \ldots$, or

$* \alpha_1 \beta_1 \alpha_2 \alpha_3 \beta_2 \sout{\beta_1} \beta_4 \ldots$, or

$* \alpha_1 \sout{\beta_4} \alpha_2 \beta_1 \beta_2 \beta_3 \sout{\alpha_2} \ldots$, etc.

When considered in terms of movement through the state spaces, not only does each component interaction have to maintain its ordering, but also each of them can be ‘put on hold’ at any time and resumed an arbitrary number of steps later.

Therefore the two chains should be treated as really separate, active in parallel, possibly engaged or disengaged at any time by either participant. In other words, they must be allowed to be temporally independent, advancing separately and at different rates from each other.

This parallelism helps accomplish broad behavioral competence, since believable activity will routinely involve a large number of different activities on overlapping time scales (Bates, Loyall, and Reilly 1991). In real communication, these activities can easily be engaged asynchronously, at the same time; they can influence each other, but they can also happen concurrently and independently. If these are represented with independent state spaces running in parallel, they can all track the situation at the same time and respond independently of each other.

Parallelism also improves failure recovery. The agent must not fail completely during a conversation, admitting "I don't understand what you're saying". It is unacceptable to just give up, because it is jarring (Mateas and Stern 2004). Rather, the agent must mimic human modes of error recovery to keep the interaction going—try to get the missing information, force an interpretation, talk around the subject, and so on. It must fail gracefully at the edge of competence.

Parallel models allow for such redundant representation: the same protocol can be represented on multiple levels concurrently, from the most specific to the most general

level. For example, a shopkeeper could track a question on three levels: with a specialized exchange such as *answering-questions-about-item-price,* embellished with exaggeration and flattery, as well as the more straightforward *answering-questions* exchange, and the most basic *turn-taking* model. This improves error recovery—when the more specific interaction fails, the less specific representations will still have some knowledge of what goes on, and may be able to steer the conversation in the right direction.

These elements are not necessary or sufficient conditions for believability—they are guidelines, bits of "selective non-realism" that, all else being equal, help mimic the very human ways in which people deal with communication. However, we must also remember that believability is painfully context-sensitive. Believable characters do not have to be realistic—rather, they have to behave appropriately given the context of their roles in the fictional world. But this context is, in turn, influenced by the game design and user expectations; character believability can be completely destroyed or restored by manipulating only those two elements. Therefore, believability cannot be considered inherent in the behavior itself. It is sensitive to the design of the overall system.

1.3.4. The Hierarchical-Parallel Problem

This is the difficulty: models of interaction ought to allow for both hierarchical causal dependence, and parallel temporal independence, within the same system. I refer to this as the *hierarchical-parallel problem*.

The hierarchical-parallel problem makes finite-state modeling difficult. Hierarchy is easy to represent using standard pushdown techniques known from theory of computation; but a pushdown machine does not support interleaved, temporally independent engagement. Parallelism, on the other hand, is easily achieved by activating all models separately, at the same time; but this makes no allowance for causal interdependence.

1.4. Hierarchical Parallel Markov Models

I would like to propose an approach that ameliorates such problems. The solution is to extend finite-state models, and in particular HMMs, to support both hierarchy and parallelism within the same system.

The approach can be outlined as follows. The interaction itself can be decomposed into simpler elements, represented via simple, separate HMMs. Stochastic models in particular improve robustness in face of noisy and uncertain inputs.

Parallelism can be accomplished easily, by making all the HMMs concurrent. Running all the models at the same time, all the time, makes it possible to track several interleaving or simultaneous activities.

Hierarchy requires an addition of causal dependencies between the HMMs, such that the parent models could influence the children, and vice versa. This can be reduced to very efficient operations.

In section 3.3, we notice that two particular types of causal dependencies are of interest in this context. First, state dependency, which happens when a single state in one model influences another single state from a different model. Second, model dependency, which happens when a single state in one model influences a different model in its entirety. These dependencies are both cases of state-to-state coupling.

This coupling can be reduced to evidence estimation. We turn the state of the controlling HMM "inside-out", into a type of evidence that could be used by other HMMs. Finally, this bottoms out in a simple and efficient mechanism described in section 3.2.

The result will be a system composed of enhanced HMMs that run in parallel, at the same time, while also implementing a hierarchy of control that is required for the performance of complex activity. The hierarchical dependencies are implemented

using a simple coupling mechanism, which does not require the alteration of existing, efficient algorithms for belief estimation.

The benefit of such an approach is that each space remains inexpensive to compute, and the cost of the overall system is merely the sum of the individual costs. The approach also keeps the individual models separate and parallel, improving ease of authoring, and broad engagement in conversational behaviors.

The drawback is that the approach is not a general replacement for all sorts of hierarchical dependencies; however, it appears to be sufficient for the kinds of hierarchies required in player interaction.

Two systems will be described later, which demonstrate interaction with the player in the context of a game, and illustrate the benefits of this approach with regard to modeling and performance efficiency. They are built of a number of hierarchical parallel HMMs, many of which are shared across the two implementations.

1.5. Contributions

To reiterate, the following are the contributions of the work:

1. Hierarchical parallel Markov models, through which a complex interaction could be represented as a collection of smaller, simpler elements, running in parallel, and combined to reproduce the complex engagement, and

2. Coupling mechanism through evidence separation, which enables hierarchical dependencies to be implemented simply and efficiently, without requiring any changes of existing belief update algorithms.

These are the contributions I present explicitly; I also hope you will find useful my discussion of knowledge representation and believable behavior development for computer games.

1.6. Dissertation Outline

The dissertation is laid out as follows.

Chapter 2 situates my approach in the space of contemporary conversation and interaction models. I examine existing approaches in the gaming domain, the state of the art in finite-state dialogs and other artificial intelligence systems, and consider findings from linguistics and related disciplines, the different approaches they propose, their benefits, and limitations. Finally, I use this work to advance a structural stance: that simple interaction can be successfully modeled using stochastic finite-state mechanisms, and that a large number of interesting in-game interactions belong to this simple category.

Chapter 3 introduces the details of my approach: coupling for hidden Markov models, which implements interdependence between concurrent representations. The

algorithmic details of Markov modeling are presented, followed by the formal description of the extensions, and examination of how coupling leads to hierarchical parallel protocol engagement. This chapter concludes with the examination of the performance improvement over an equivalent flat mechanism.

Developing on the model description, Chapter 4 is dedicated to results. I present a general description of the system, I describe two game characters developed using the engine, and discuss their behavioral effects, such as redundancy and smooth performance degradation.

Following that, in Chapter 5, I discuss how such a system could be re-implemented. This describes not only the existing engine's architecture, but also details of implementation of the different modules of the system, and opportunities for optimization. This section will be of greatest interest to those interested in building their own variations on such a system.

Finally, Chapter 6 re-examines the contributions of the works, and evaluates the system in the larger context of entertainment production. This discussion concludes with notes on the future directions for this class of mechanisms.

Chapter 2. Perspectives on Interaction.

People know what they do; they frequently know why they do what they do; but what they don't know is what what they do does.

— Michel Foucault

In this chapter I survey the different perspectives on interaction and communication, beginning with existing approaches from entertainment, followed by contemporary techniques for human-computer dialog, both finite-state and planning-based. Finally, perspectives from sociology and linguistics are introduced, which will pave the way towards a coupled structural approach.

2.1. Simple Techniques

In entertainment production, interaction tends to be simple, because of the strong demands of easy authoring, believable runtime performance, and efficient resource usage. Techniques popularly used in games and entertainment products tend to fall into several categories.

2.1.1. Command Interfaces

Perhaps the best-known approach to language-based interaction is the command interface used in adventure games and interactive fiction. These games are usually text-based, and the player issues commands *directly to the game,* which also acts as the player's proxy in the virtual world. For example, the player might say: *"pick up the first gem",* or *"attack the goblin with the poisoned dagger",* at which point the computer will simulate the performance of the tasks, and inform the player of the new state of the world.

Technically speaking, the command interface performs standard syntactic parsing, and then runs some appropriate actions based on the parse tree. The author of one modern text adventure engine describes this situation as follows:

> The built-in parser handles most of the "syntactic analysis" part of parsing: dividing a command into individual words (or "tokens"), determining the sentence boundaries of command lines with multiple commands, figuring out which words specify the verb and prepositions, and identifying noun phrases. The built-in portion also defines the execution process by calling a series of methods in the objects involved in a command to carry out the action of the command. (Roberts 2002, ch. 4)

Most of the game involves action performance in the world. Consequently, commands are issued in imperative voice (second person singular), sometimes chained as subordinate clauses of one sentence. This complicates parsing, since off-the-shelf parsers are not trained to deal with this unusual convention; product developers therefore routinely resort to building their own parsers and command interpreters.

Interacting with a non-player character (NPC) in a game also has to be expressed in the form of action commands. Actions are either canned speech acts (usually *ask* or *tell*, e.g., *"ask the goblin about the gem"*), or commands that get parsed recursively from the point of view of the character (e.g., *"say to the goblin, give me the gem"*).

While NPC code in such games can potentially be arbitrarily complex, the imperative command interface severely restricts what kinds of conversation can take place. In practice it is difficult to implement anything other than giving the NPC orders to perform specific actions.

This approach results in a language interaction that is highly unnatural—the grammar of the command language is usually tightly constrained and unambiguous in order to simplify processing, and the computer tends to be inhumanly pedantic and unforgiving. The experience of the system is very reminiscent of interactions with an operating system shell, where the novice user has to be tutored in the proper construction of commands and dealing with errors.

2.1.2. Nonstructural Techniques

Another popular approach to natural language interaction is exemplified by pattern matching systems commonly known as *chatterbots*.

In its simplest shape, a chatterbot is a reactive pattern matching system, where each pattern matches the surface features of the player's sentence, and suggests the production of some particular responses. The first well-known engine of this type was *Eliza* by Weizenbaum (1966), featuring an immensely clever pattern library that imitated a Rogerian psychoanalyst.

I call this approach nonstructural, because it retains no representation of the temporal structure of the interaction as it develops over time. More complex implementations sometimes retain a few specific bits of information, which can be used to mimic very simple structure—for example, ALICE (Wallace 2000) can set memory registers, to be used in subsequent matching. However, this is hardly enough support for the easy modeling of long-term engagement.

A different approach, although similar in spirit, is to use word-based n-gram models, such as in the system by Hutchens and Barnes (2002), based on *MegaHAL* (Hutchens 1998). The system uses an n-gram of word sequences: when an utterance comes in, a set of custom heuristics picks a salient starting word, and the n-gram generates a reply utterance.

The major difference is that the n-gram models can easily be acquired automatically, by observing human players conversing in networked games, chat rooms, discussion boards, and so on. With regard to behavior, however, the system resembles *Eliza* in that it contains no model of the larger structure of the interaction.

In general, nonstructural approaches prove popular in certain settings, and a large number of chatterbots have been developed for a variety of purposes. Some of the popular contemporary implementations include *Verbots,* used for fielding customer

support questions (based on the Julia engine by Mauldin, 1994), and the open-source ALICE engine, used mainly for amusement (Wallace 2000).

However, nonstructural techniques exhibit considerable problems maintaining meaningful long-term coherence in communication. Coherence is important locally, among the utterances, as well as globally, among the different segments of discourse (Grosz et al. 1995). Structure-impoverished models lose coherence exactly because they represent very little about the discourse—which results in failure to participate in interpersonal behavior protocols, and failure to produce a focused interaction.

When these systems do succeed, it is *in spite* of the lack of structure—because the designers set up the interaction to trick the user into an expectation of meaningfulness. As sociologists have noticed, when people expect communication to be meaningful, they will routinely *force* a meaningful interpretation upon the communication (Garfinkel 1967, ch. 3). The messages do not even have to be produced by an understanding, meaningful agent—they can be even produced randomly—but it is crucial that the person not be aware of this, otherwise the expectation of meaningfulness will disappear. This is akin to the artistic notion of suspension of disbelief—the viewer will play along with the fiction of what is being shown, but any inconsistencies will cause the entire artifice to fall apart.

Chatterbots' success is contingent on whether they avoid failing this expectation of meaningfulness. In practice, failure occurs almost immediately, due to the lack of information about the history of the interaction. And while failure can be delayed in certain narrow contexts—such as in first-person shooters, where conversation tends to be degenerate (Zubek and Khoo 2002)—those are rare and not representative of typical game interactions.

This failure mode of chatterbots is directly related to the *nonstructural conversation stance* they embody—the assumption that it is not necessary to retain much information about the long-term structure of the conversation. In chatterbots or n-gram systems, knowledge representation typically includes little more than stimulus-response pairs, perhaps supplemented with a small number of specialized variables, and no effort is made at retaining structured knowledge about the larger interaction or its topics.

The justification for this stance is uncertain. Most authors are careful not to claim that their chatterbots replicate realistic communication, and Wallace only goes as far as to propose well-known word distribution regularities (Zipf 1935) as a possible grounding:

> Our experiments with ALICE indicate that the number of choices for the "first word" is . . . only about two thousand. Specifically, 1800 words covers 95% of all the first words input to ALICE. The number of choices for the second word is

> only about two. To be sure, there are some first words ("I" and "You" for example) that have many possible second words, but the overall average is just under two words. The average branching factor decreases with each successive word. (Wallace 2004)

Unfortunately, in choosing the Zipf analysis, the author already assumed that structural information is unimportant; a chatterbot can have pre-scripted responses for the vast majority of common phrases, but this does not demonstrate that structure is unimportant in naturalistic conversation.

In the practice of implementing conversational characters, pattern-matching techniques are rarely enough—without the knowledge of what goes on in the conversation, their performance is fragmented and difficult to engage over the long term. Furthermore, nonstructurality introduces a formidable technical problem: with no concept of the ongoing context, the agent has to re-acquire all the information it needs from each incoming utterance, which is only viable in limited situations. Such approaches are therefore suited best to exchanges that can be modeled as series of mostly disjointed utterance pairs.

2.1.3. Menus and Conversation Trees

Many game designers have noticed that when the activity space is limited and the options easy to enumerate, interaction could be easily implemented using standard graphical user interface (GUI) approaches.

The Sims (Maxis 2000) may be the best-known contemporary example of a character interaction GUI. The user, controlling a character in the game, orders action performance by clicking on objects; at this point an action menu appears, sensitive to the subject and object of the interaction, and the user selects which action to perform. Dyadic interaction and communication work in exactly the same manner—the player clicks on the character that will be the target, then a contextual menu presents the list of available interpersonal actions.

A special case of the GUI is a *conversation tree*, popular in role-playing games such as *Neverwinter Nights* (Bioware 2002), and commonly used to implement dialog in games. As the name suggests, it is a tree-based structure of text blurbs, corresponding to all the possible NPC utterances and all the possible player responses. The agent starts the conversation by reading out the text of the root node, and the player is presented with some response blurbs; the player chooses one of the responses, which leads to another text node, potentially another set of choices, and so on.

The GUI interface has the unique property of making the player's action space explicit. This revelation carries both positive and negative consequences.

On one hand, action explication makes it easy for the player to see precisely what they can do to advance the game; it removes the need for "verb hunting", and abstracts the interaction away into the level of speech acts, pre-written utterances, or other desirable action abstractions. In the context of conversation trees, it also allows for very precise control over the dialog text.

At the same time, explication makes it easy for the player to reconstruct the topology of the world state space. It encourages the player to treat interaction as graph traversal, and to reduce the activity into path advancement or backtracking. The net effect is the transformation of an interaction into a formal puzzle.

The relative importance of these effects depends on game type and designer's intentions. However, in my experience with conversational contexts, action explication is rarely desirable. When the activity is conversation, explication turns it into mere graph search; the resulting "puzzle effect" robs the interaction of the aesthetics of engaging linguistic performance.

2.2. Finite-State Dialog

The dream of human-computer interaction is as old as the field of AI, but the problems it presents are legion. Even a "weak AI" approach, concentrating on the implementation of dialog, is immensely difficult. As Scott and Kamp introduce it:

> A central problem which the development of dialogue systems encounters is one that it has inherited directly from contemporary linguistics, where one is still struggling to achieve a genuine integration of semantics and pragmatics. A satisfactory analysis of dialogue requires in general both semantic representation, i.e. representation of the content of what the different participants are saying, and pragmatic information—what kinds of speech acts they are performing [. . .] and, more generally, what is the purpose behind their various utterances or even behind their entering upon the dialogue in the first place. (Scott and Kamp 1995: 230)

The quotation above only touches upon the issue of language understanding; a full-fledged dialog system also requires good language generation.

Commercially used dialog systems can be categorized into two broad categories: finite-state approaches, and plan-based approaches. We now examine them both, and then turn our attention to other aspects of AI that will come to bear on this work.

2.2.1. Finite-State Approach

Human dialogue exhibits numerous structural regularities, for example, adjacency pairs of questions and answers or requests and replies (Schlegoff and Sacks 1973). These regularities can be expressed structurally, as a grammar, and implemented efficiently using a state machine.

Dialog participation becomes explicit state space traversal, collapsing both the evolving situation and the conversation into a single finite-state model. McTear succinctly describes the application of this approach, as it pertains to question answering:

> In a finite-state model the dialogue structure is represented in the form of a state transition network in which the nodes represent the system's questions and the transitions between the nodes determine all the possible paths through the network, thus specifying all legal dialogues. (McTear 1998)

The nodes are not limited to questions, of course, but they can represent valid communication actions in general—and the similarity to conversation trees (see Section 2.1.3) is evident, although these systems tend to allow for natural language input.

Even though "[m]ost commercially available dialogue systems use some form of finite-state dialogue modelling" (McTear 1998, *sic*), finite-state models are routinely criticized

as failing to scale. First is the problem of knowledge representation: the explication of each possible dialog as a path through the state space leads to explosively large spaces. Second, the problem of action estimation: the human's utterance must be mapped to the correct graph edge, otherwise the system's interaction state will be incorrect. The first of these classes of problems can be tackled by language abstractions and partitioning the space into hierarchical subspaces; the second by the introduction of stochastic models.

2.2.2. Language Abstraction.

The space of possible user utterances is infinite, but large subsets of them routinely share similar meaning; therefore, they should be collapsed into a smaller number of abstract 'conversational actions'. This is routinely done in practice, using speech acts (Austin 1962, Searle 1965) as the preferred level of abstraction: some set of speech acts is identified, some detection conditions are postulated against which user's inputs could be matched, and those are translated into graph edges. As Traum (2000) points out, the taxonomies of such communicative acts are still being worked out. In finite-state systems, communicative act identification is often based on surface features, such as general speech act verbs, like *ask*, *want,* or *implore* (Wierzbicka 1987), more specific implicature conventions, such as *"can you (do-such-and-such)"* (Morgan 1978), or even on very specific idioms such as *"what's up"*.

Most speech acts are predicated on some situation variables, and this can be problematic given the propositional nature of finite state models. It is possible to implement some variable binding in an essentially propositional system (e.g., Agre and Chapman 1987, Maes 1990, Horswill 1998), but the lack of a standard solution remains a significant problem.

With regard to agent behavior modeling in particular, finite-state dialogs stand out in several ways. First, spaces are easy to inspect and predict. Given information about the current state of a dialog, one can often tell how the space will change given some new input; and vice versa, one can often tell what kinds of inputs were detected given some change in the model. This is crucial in the process of authoring (and thus debugging) desirable behavior.

Second, the explication of the temporal structure—how a situation is supposed to progress from one time slice to the next—allows for a great deal of control over the activity. The temporal structure is highly meaningful, and individual interaction elements acquire meaning from their position in the activity; a finite state approach allows detailed control over this progression.

2.2.3. Stochastic Dialogs.

Stochastic finite-state techniques, such as hidden Markov models, have been very successfully applied to the lower-level problems of parsing and spoken language

understanding, as they perform robustly in the face of noise and error (Jelinek 1997, Roche and Schabes 1997).

The same approach has recently been extended to dialog modeling (Levin and Pieraccini 1997, Young 1999, Roy, Pineau, and Thrun 2000, Singh et al. 2002). Typically, dialog is represented as a Markov model with an action policy, otherwise known as partially-observable Markov decision process (POMDP). The representation often works as follows:

> The domain is represented as the partially observable state of the user, where the observations are speech utterances from the user. The POMDP representation inverts the traditional notion of state in dialogue management, treating the state as unknown, but inferable from the sequences of observations from the user. (Roy, Pineau, and Thrun 2000).

An additional attraction of stochastic approaches is that one can attempt to acquire the policies automatically: "dialogue policy can be designed using the formalisms of Markov decision processes (MDPs) and reinforcement learning (RL)" (Singh et al. 2002: 106). Automatic dialog strategy acquisition is a topic of active research (Levin, et al. 1999, Pietquin and Dutoit 2003).

However, we will not concern ourselves with automatic model or policy acquisition here. As Pietquin and Renals (2002) say: "Though the field of spoken dialogue systems

has developed quickly in the last decade, rapid design of dialogue strategies remains uneasy. . . [The] quality of the strategy learned by the system depends on the definition of the optimization criterion and on the accuracy of the environment model." Unfortunately, as Finney et al. (2002) admit, it is difficult to define the criteria appropriately for domains with complex representations, such as those that employ situational variables. Automatic model acquisition for a system whose semantics include more than a single space—such as the coupled approach presented here—unfortunately lies outside of the scope of this dissertation.

2.2.4. Stochastic Model Decomposition and Coupling.

The problem of state explosion led to the exploration of numerous composite representations for stochastic models. While not applied to dialogs as such, these approaches are relevant and potentially very useful in finite-state dialog modeling.

Complex space decomposition is usually approached in two general ways. Meuleau et al. (1998) describe it well:

> An MDP is either specified in terms of a set of "pseudo-independent" subprocesses or automatically decomposed into such subprocesses. These subMDPs are then solved and the solutions to these subMDPs are merged, or used to construct an approximate global solution. These techniques can be divided into two broad classes: those in which the state space of the MDP is

> divided into regions to form subMDPs, so that the MDP is the union (in a loose sense) of the subMDPs; and those in which the subMDPs are treated as concurrent processes, with their (loosely) cross-product forming the global MDP. (Meuleau et al. 1998: 165).

Furthermore, once decomposed, the models can be coupled in a number of ways. Authors who work with HMMs tend to combine the conditional probabilities of the constituent spaces, and change the estimation algorithms to support that. Authors who work with POMDPs often combine the action policies instead, by making the reward functions in the different MDPs dependent on each other. The latter makes it easier to acquire to policy automatically, since the individual models remain small and completely separate.

Space decomposition allows for hierarchical reconstruction. Hierarchical structure is a novel enhancement of Markov models. The motivation comes from the observed space explosion: flat models induce an undesirable multiplication of redundant substructures. Hierarchical approaches seek to abstract away redundancies; as Pineau, Roy, and Thrun (2001) put it, “in many tasks the action hierarchy gives rise to state and observation abstractions, which can drastically reduce the computational complexity”.

The basic approach is the Hierarchical HMM, or HHMM. Under this approach, each hidden state can represent an entire stochastic process:

> HHMMs generalize the standard HMMs by making each of the hidden states an "autonomous" probabilistic model of its own, that is, each state is an HHMM as well. [. . .] An HHMM generates sequences by a recursive activation of one of the substates of a state. This substate might also be composed of substates and would thus activate one if its substates, etc. (Fine, Singer, and Tishby 1998).

This extended Markov model includes a new class of "vertical transitions", corresponding to the transition of activation from a parent node down to the child model it represents; each model also contains a unique end state, with an "up transition" from the end state, back to the parent. These are independent of the standard "horizontal transitions" between nodes in the same model, and the forward algorithm is extended to account for them.

The hierarchy forms a strict tree, in which the children are not shared. If we consider the set of observations as an alphabet, and a sequence of events as a string in the alphabet, we can see that each child recognizes a strict substring of the string recognized by its parent. As the authors observe, "if state q had generated the string $o_i \ldots o_j$, then its parent state generated the string $o_k \ldots o_l$, such that $k \leq i$ and $j \leq l$" (Fine, Singer, and Tishby 1998, section 3.2). This scheme of vertical transition does not allow for parallel co-activation of different children; it only recognizes interactions that result in properly nested sequences of observations.

While HHMMs specify that each child has a unique exit node, Theocharous and Mahadevan extend this model to allow for "multiple entry points into abstract states which we call entry-states, and multiple exit-states" (Theocharous and Mahadevan, 2002). This allows for easier estimation of likely sequences, since such a non-linear child model can be used in a number of ways. The nesting property persists, however.

Factorial hidden Markov models (FHMMs) decompose a single model into a collection of completely independent models that combine only in output production (Ghahramani and Jordan 1995). The collection of components is much more efficient than a composite model, but its underlying assumption, that the processes are independent, holds only in limited circumstances.

Several other approaches allow for a degree of interdependence, and have been used for certain kinds of problems. Linked hidden Markov models are akin to FHMMs extended to include joint probabilities for co-occurring states, as a way to model interdependence within a time slice (Saul and Jordan 1995). Hidden Markov decision trees extend this further, by using the joint probabilities to implement hierarchical top-down dependencies, with master processes controlling slave processes (Jordan et al. 1997). However, the joint probabilities complicate belief computation.

Temporal dependence extensions have been made in an elegant coupled hidden Markov model approach (Brand 1997), in which separate models are enhanced with

conditional probabilities across space and time boundaries, and algorithms over the model are enhanced to support that. However, the cross-dependencies can easily become numerous.

Finally, it is worth noting that HMMs can be treated as a special case of dynamic Bayesian networks (DBNs), which can efficiently replace them in all manner of situations (c.f. Murphy 2002, ch. 2). Both single and separable HMMs are easily and efficiently modeled using DBNs. However, networks of interdependent HMMs are still problematic, as the resulting DBNs require numerous 'coordination nodes' to represent the dependencies (ibid, section 2.3.9).

Having discussed methods for model-based coupling, we should quickly consider action-based coupling. This general approach is considerably different, in that the component spaces remain independent, usually active in parallel, but the action performance is coupled.

In their robot dialog system, Pineau, Roy and Thrun (2001) use a hierarchical POMDP—a tree of separate HMMs augmented with action policies, but where all elements are coactive in parallel, and policies are local to each constituent HMM. To perform an action, at each iteration the system re-evaluates the policy of the root model, which may select the evaluation of a child model, recursing down in a manner very similar to teleo-reactive trees (Nilsson 1994) until a primitive action is produced.

This "differs from many hierarchical MDP algorithms where an agent 'remains' in a subtask until a so-called terminal state is reached" (Pineau, Roy, and Thrun 2001, section 3.4), because each iteration re-evaluates the most active sub-process, which can change at any time. Thus coupling is accomplished on the level of action performance, such that in every time slice, the parent chooses whether to hand over control to a child, the child to another child, and so on, until an action is finally produced.

In a different approach, Meuleau et al. (1998) present a system in which action policies are coupled through resource consumption—expected action performance reward is made conditional on the availability and depletion of some set of resources, such as a global resource shared across processes, and resource allocation to one process influences the expected reward across the other processes. They found their approach to scale very well, such that the individual policies can be computed reasonably quickly even for a large number of constituent processes.

2.3. Plan-based Dialog

The other major approach casts dialog participation as a special case of action planning. Conversation can be viewed as the performance of actions (Searle 1965), such as requesting, commanding, informing, and so on, which are part of the speakers' plans to bring about some kind of an effect on the participants and the situation. Under this plan-based theory of communication, "the listener's job is to uncover and respond

appropriately to the underlying plans, rather than just to the utterance" (Cohen 1995: 237).

2.3.1. Dialog as Activity

Historically, the approach was an answer to the earlier work on semantic language understanding, concentrating on the mapping from syntactic structures to logical form and semantic meaning. Under one popular model (Montague 1973), semantics were obtained via the application of very simple rules, transforming language elements into lambda calculus. The rules were grounded in syntactic constituents, so the logical truth value of a sentence could be evaluated by simply transforming the parse tree. An extension of such an approach can be seen in certain frame-based language systems (Seneff 1992), which collapse the syntactic parsing and semantic recognition into one process.

However, recent work in conversation has been dominated by the treatment of dialog as *activity*. As it had been observed, language understanding requires general reasoning about action performance, which requires mechanisms different from theorem proving (Allen 1993). Unfortunately, bare semantics of what is being said do not necessarily correspond to what the speaker means to communicate given the task at hand (Morgan 1978, Cohen and Perrault 1979), leading to erroneous response.

The observations manifested themselves most clearly with implicatures. To disambiguate implicatures and underconstrained speech acts, one has to know something about what is being done in the situation. The early work concentrated on the addition of speech act theory (Austin 1962, Searle 1965) to the existing work on action planning (Fikes and Nilsson 1971), for the purpose of the generation and interpretation of speech acts in conversation (Cohen and Perrault 1979, Allen and Perrault 1980, Appelt 1985, Litman and Allen 1990). In this manner, figuring out what to say was approached using explicit reasoning about action performance.

This is further extended in several directions. Some activity is noticeably difficult to understand as coming from isolated plans, and requires some notion of mutual collaboration and understanding, which led to the view of joint activity as the basis of understanding (Cohen and Levesque 1990, Grosz and Sidner 1990). Furthermore, actions and utterances themselves have meaningful surface forms, which calls for the re-examination of how structural and surface patterns can be used in the task of understanding (Grosz 1977, Alshawi 1987) and generation (McKeown 1985).

The major benefits of plan-based dialogs are that they present an elegant method for disambiguating implicatures and speech acts (via plan estimation); they embody a clear approach to constructing action performance policies (via plan construction), and they fit existing cognitive frameworks, by offering “a generalization in which dialogue can

be treated as a special case of other rational noncommunicative behavior." (Cohen 1995)

Interactive agents implemented using a plan-based approach, such as Steve (Rickel et al. 2000, 2002), or Paco (Rickel, Lesh, et al. 2001), usually exist in the context of collaborative, tutoring activity. Teaching and assistance tasks are notoriously difficult, since the artificial agent must not only succeed at its part of the task, but also help the human—and in order to help, the agent must be able to tell what the human is doing and in what particular ways they fail. This is commonly implemented using a model based on joint intentions and shared plans.

In his critique, Cohen cites several theoretical and implementational problems (ibid: 238-9). First, the problem of act recognition: the correspondence of speech acts to atomic actions in a plan is somewhat problematic (Cohen and Levesque 1990). Next, the problems of scope: not only is plan recognition and planning computationally intractable in the worst case, but the situation may require numerous simultaneous plans of different scope (Litman and Allen 1990). Finally, the theoretical foundations are yet to be worked out: it remains unclear what the basic theoretical constructs should be, and how to evaluate their applicability to observable phenomena.

2.3.2. Mixed Approaches

Many approaches treat dialog as a problem of mixed representation, linking both finite-state and plan-based approaches.

In entertainment-oriented research, dialog is usually joined with the task at hand. Mateas and Stern's *Façade* (2004) uses simple pattern matching to trigger multiple speech act possibilities, which are then matched against the dramatic situation "beats" arranged by the reactive planner. Utterance production in *Façade*, however, is the assembly of complete pre-recorded blurbs. Loyall's *Woggles* (1997), on the other hand, produce language from word-level fragments, using goal-based behaviors; this is similar to their general action control system, except the outputs are lexical as well as behavioral.

The addition of embodiment in the virtual world takes this even further, since embodied language production benefits from linking with action production, including gestures, postures, facial displays, and eye movement (Cassell 2000, Cassell, Bickmore, et al. 2000, Rickel and Johnson 2000). For example, the Bosnia Mission Rehearsal Exercise system (Rickel et al. 2002) uses embodiment in a game-like world to accomplish its tutoring tasks—where the agents use their artificial bodies to augment the dialog, by guiding the conversation and attention, expressing emotion, or demonstrate understanding.

Another type of a mixed approach is presented by frame-based dialog systems (Seneff and Polifroni 2000, Lamel et al. 2000, Bonneau-Maynard and Rosset 2003). As the name suggests, they represent the situation using frames, whose slots need to be filled, in order for the activity to proceed.

Such systems recover some semantics of the user's utterance, typically through inexpensive parsing, fit this information against the situational frame, and perform a modicum of dialog management to specify what to do with this information. For example, in popular ticket reservation tasks, the details of when and where are easily represented as frame slots—so when an utterance was received and matched against the situation frames, the dialog manager then produces a new utterance based on which information remains missing. The result is an approach related to structural situation modeling, though the name may not immediately suggest it.

As mixed approaches suggest, the distinction between finite-state models and plan-based models makes for a convenient taxonomy, but is not grounded in an inherent dichotomy. Future systems will likely successfully employ a mixture of both approaches. The situated activity model suggests that, for reasonable situations in the limited worlds of computer entertainment, a situated, structural description of activity can be quite sufficient. As the complexity of game worlds increases, however, it would be quite beneficial to merge the finite-state and planning-based approaches, to create

systems which can not only efficiently engage in known interaction structures, but also deal intelligently with novel and unpredictable situations.

2.4. Perspectives from the Social Sciences

This work is also influenced by two perspectives from the social sciences: theories of situated action, and of structural modeling of cognition.

2.4.1. Situated Action

Sometimes known as the local management model, the stance comes out of ethomethodology, the sociological study of the creation and maintenance of shared meaning (Garfinkel 1967, Heritage 1984). It has been applied with success in cognitive anthropology as well as artificial intelligence (Rosenschein and Kaelbling 1986, Agre and Chapman 1987, Suchman 1987, Sibun 1991, Agre and Horswill 1992).

The approach emphasizes the use of local resources to produce globally coherent behavior:

> The local management model sees adaptation as a continuous process of reacting to the concrete contingencies of sequentially organized, situated action. In this view, abstract act-types and global organizational structures (like plans) may be useful as post-hoc, summary descriptions of adaptive behavior,

> but adaptation itself is made possible by processes not easily described in the everyday language of plans and goals [. . .] . (Lambert 1992, p. 3)

Agre describes an application of this approach to artificial systems:

> It is sometimes necessary to engage in symbolic reasoning about the future, of course, and to make representations of action to help guide future activities. But we would like to suggest that these more complex forms of reasoning about action are delimited and controlled to a substantial extent by the structures in the world that support simpler forms of moment-to-moment action choice. (Agre, 1995, p. 44, citing Agre and Horswill, 1992.)

With regard to linguistic communication in particular, the stance is expressed similarly: that simple situation-sensitive activity is at least as important as abstract reasoning. From Herrmann:

> Mental representations of situations, goals adopted, and means chosen to attain them—all depend on the speaker's acquired knowledge, i.e., experience. This includes both his declarative knowledge of the world (*knowledge that*) and his procedural knowledge (*knowledge how*). The mental representations of situations and the setting of goals require the activation of both kinds of knowledge [. . .] (Herrmann 1983, p. 6)

Procedural knowledge entails the existence of verbal behaviors applied to particular situations. It seems unlikely that these behaviors are recreated from scratch for each new situation; indeed, in many domains different people produce communication that is very similar in structure and content. This suggests a possible avenue of analysis, by decomposition into constituent situational idioms.

Linde and Labov (1975) suggest that a "conversational idiom" approach works for well-practiced situations that do not require novel or strategic performances. However, recent analysis suggests that even complex communication exhibits idiom-based composition. O'Keefe and Lambert (1995) present a series of experiments in which the interlocutors are placed in a delicate situation: breaking a date, rejecting admission to an honor society, or negotiating drug compliance with a patient. Even in such situations, significant regularities emerged in the particular idioms employed, and their mutual relations. In the first case, 72 basic types of conversational messages were found; in the second, 21 types, grouped into eight consistent themes; in the third, 61 types, in eleven themes, and only three of them "influencing perceptions of effectiveness in meeting task and interpersonal goals." (O'Keefe and Lambert 1995, p. 64)

Furthermore, O'Keefe and Lambert found an interesting disconnect between the semantics and the effects of these messages; as they noticed, messages with similar points and effects could have very different contents, and messages with very different

effects could have similar content. Also, the antecedents and effects were associated with the specific message elements, not the entirety of the message.

Given the strong evidence of structured idiomatic expressions in conversation, the authors thus present an alternative view of message production—as an organization of idiomatic expressions, whose sequential performance comes from their mutual dependencies:

> We found such an alternative in an image of communication situations as organized fields of thoughts, and messages as the result of thought selection and expression. Message structures arise as focus moves through the field of thoughts. Focus is driven by goals and guided by the route that the speaker formulates to move through the field. [. . .] The resulting message, rather than being a functionally unified act, is a collation of thoughts, each of which may have distinctive consequences and effects. (ibid, p. 66)

This approach recasts behavior and reasoning in different roles than plan-based language production. Under this model, the import of a linguistic element is based on its position in the conversation, based on what came before it; and the element constrains what can be done next. The larger interaction is then modeled as movement through this field of locally organized idiomatic expressions. This resonates greatly with structural approaches to interaction. The result, although implemented in a

connectionist network, is similar to a conversation graph, in that it makes explicit use of the mutual occurrence and constraints of higher-level conversational idioms.

2.4.2. Conceptual Structures

Another very influential approach comes from the intersection of cognitive science, cognitive psychology, and artificial intelligence, and is concerned with the structural representation of knowledge about everyday activity and everyday language use:

> Our focus will be upon the world of psychological and physical events occupying the mental life of ordinary individuals, which can be understood and expressed in ordinary language. [. . .] Here we are concerned with the intentional and contextual connections between events, especially as they occur in human purposive action sequences. (Schank and Abelson 1977, p. 4)

The overall approach encompassed a number of interrelated elements rooted in the conceptual dependency theory of understanding, which used a structural cognitive representation using a handful of general primitives (Schank 1972). From this, the approach extended to handle *scripts* for mundane everyday activity; *plans* with which novel situations can be understood, and existing scripts can be extended; and *themes* which organize an agent's *goals,* which in turn influence the creation of plans and engagement of scripts.

A number of systems were implemented to test this general approach: MARGIE (Riesbeck 1975) and ELI (Riesbeck and Schank 1976) systems for language processing, in which the sentence was fitted against a conceptual representation to determine word and utterance import; SAM (Cullingford 1981), the Script Applier Mechanism which processed newspaper stories using stereotyped story representations; PAM (Wilensky 1981), which explained classes of stories using goal representations and planning; and many others (Schank and Abelson 1977, Lehnert 1977, Schank and Riesbeck 1981).

Two elements of this approach are particularly relevant to our discussion of finite-state interaction. First is the concept of a script, a structural conceptual representation of stereotyped activity, which is a particularly powerful approach for dealing with the complexity of everyday interactions:

> We use specific knowledge to interpret and participate in events we have been through many times. Specific detailed knowledge about a situation allows us to do less processing and wondering about frequently experienced events. We need not ask why somebody wants to see our ticket when we enter a theater, or why one should be quiet, or how long it is appropriate to sit in one's seat. (Schank and Abelson 1977, p. 37)

This approach was further extended in ASK systems (Schank and Cleary, 1995), which used script-like conversations to motivate learning. Even though the question-and-

answer conversation itself was completely scripted, it was based on the insight that the situation developed in a particular manner, and this development constrained what was likely to be asked.

Second is the idea that language is so intimately related to the activity, that the latter should lead the processing of the former. One can notice that only a minimum of syntactic processing is necessary in a particular context:

> Probably the major theoretical hypothesis of conceptual analysis is the claim that a separate syntactic analysis phase is unnecessary in language understanding. (Birnbaum and Selfridge 1981)

The connection between these ideas, and finite-state dialog models discussed previously, is somewhat surprising. Even though the latter are much more limited than the conceptual dependency systems, they share a similar emphasis on the structure of interaction, and use the structure to inform language processing.

2.5. Personal Perspective: Towards a New Model

Having considered the wide spectrum of perspectives on interaction, I would like to shift gears for a moment, and discuss my own take on the issue, which grounds the work presented in this report at the intersection of the various related disciplines.

But let me begin with a personal anecdote, which I hope will illuminate the reasons behind my particular synthesis.

2.5.1. Structure in Interaction

Having immigrated to the United States as a teenager, a considerable period of my life was spent learning English. But unlike somebody learning the language as a child, I was unfortunately completely aware of my situation.

Without the language, interacting with people becomes a terrifying but fascinating activity. If you ever tried to get around a foreign country, you will immediately recognize the feeling. As linguistic understanding drops to zero, the numerous previously unnoticeable aspects of the situation suddenly come to the foreground, shedding new light on even the most mundane of interactions. Grocery shopping, for instance—I would go into a store and try to buy something, the cashier would scan the items, say something incomprehensible, and I would find myself having no idea what he said. What would you do?

Fortunately, there are ways of coping with people. The intonation will indicate if it's a question or a statement, the gaze will inform you whether you're expected to say something back. Maybe you can even recognize some terms or phrases that you already knew—although the way Chicagoans speak, you'll be lucky if you can tell where one word ends and the next one begins.

If there were no questions, it was enough to just pay and get out of there. Statements made at the end were usually just confirmations, and the “thank you” or “good evening” never carried any expectation to stay around and chat. There were also the occasional pleasantries that merited a response—statements with “have a” or “happy” or “pleasant” were always good candidates for a smile and a “you too”.

Questions and requests presented a larger problem: figuring out what was actually being asked. Fortunately, there was only a limited set of issues that routinely came up. I could be asked if I had smaller change, a discount card, an ID; if I remembered the price on some unmarked item or knew about today’s two-for-one sale; whether I wanted paper or plastic, whether this was all or there was anything else.

Recognizing these was tractable, since it only required capturing a few key elements and fitting them against the situation. Question form was important—“do you have”, “would you like”, “is this”. Figuring out the topic was also important—the ID, the bag, the item on the scanner. But even if only one of these matches succeeded, this was often enough, since the situation state itself was meaningful. Whether the utterance was made before activity commenced, as an interruption while scanning an item, while paying for the groceries, or after bagging them—the particular position was significant. It set the focus, it circumscribed the possibilities; all that was left was filling in the blanks.

These strategies did fail sometimes—with results only in retrospect hilarious—but most of the time they worked very well. It is surprising how well one can get by with only the most basic of language comprehension, when understanding is not yet fluent. Even without linguistic fluency one can get through situations very successfully, by relying on knowledge of the situation, and picking out relevant details. It is on this base of basic interaction that layers of linguistic sophistication can eventually deposit.

2.5.2. Interactions Tend Towards Structure

Recapitulating some of the previous discussion, sociology and linguistics suggests that common interactions can exhibit a great deal of structure, shared and stable across a number of particular conversations.

Within the utterance, on the low level, stability is clearly seen with speech acts. Particular acts usually have conventional expressions and surface form. While it has been suggested that implicatures work because of the logical entailments they encode (Searle 1975), the much simpler explanation is that they behave more like idiomatic fixed expressions, carrying meaning thanks to a usage convention (Morgan 1978). For instance, the standard "can you" or "do you want to" are requests thanks to being used *as* requests.

Stable structure can also be observed across utterances. Many classes of speech acts occur in sequences—such as questions and answers, requests and responses, and so on

(Schlegoff and Sacks 1973). Unfinished sequences can be nested, and position in the nesting changes communication interpretation—for example, when one expects an answer, even unintended utterances will be routinely interpreted as one.

Structure is also observed in larger sequences, such as in familiar interactions—buying items, asking clerks for information, asking for directions, coordinating trip details, etc. Some of the 'scripts' are so familiar, we don't even think about them. When the situation is well known and does not require novel strategies, people easily fall into highly predictable patterns (Linde and Labov 1975).

Somewhat surprisingly, even when the situation is not all that familiar, people *still* fall into patterns, due to the constraints of the situation. As section 2.4.1 mentions, O'Keefe and Lambert (1995) found that even in uncommon situations, people still communicated using stable structures based on "conversational idioms". The surface manifestations of these idioms were somewhat more varied than with speech acts, and the particulars of the form paid attention to the social roles and emotions of the participants, but analyzed with regard to what they accomplished in the situation they fell into stable and surprisingly constrained categories.

For just a single example, one experiment studied conversations between a hospital employee and a patient, about following a prescribed drug schedule. The surface form of the utterances was varied, but there were only 61 idioms, or conversational "moves",

that those utterances accomplished, such as imparting the importance of taking the drug, or asking the patient to wait for it to take effect. The sequencing between the idioms was highly meaningful—even though an expression of concern could potentially elicit countless response types, that did not happen, and only some responses that made sense in the situation were observed. Furthermore, in a manner very reminiscent of speech acts, it was not necessarily the raw semantics of the utterance that endowed it with meaning, but rather its place in the sequence and the success or failure at fulfilling its expected role.

And when meaning in dialog arises at least partly from its structure, it is an aspect we can attempt to exploit.

2.5.3. Virtual World Simplifies Ontology

Simplifications inherent in virtual worlds make this structure easier to analyze. In actual real-world interactions, the environment is huge and unwieldy; it contains all sorts of physical entities that require multiple levels of representation, and that can be manipulated in very complex ways.

Not so in game worlds. Due to deliberate design abstractions, as well as limitations of the interface, games routinely limit what can be done, supporting only a small set of discrete actions on a relatively small and well-defined set of discrete objects. Even though the visual representation of these worlds tends to be detailed and

photorealistic, the internal representation of the world is usually very coarse and selective; the vista is merely a backdrop, and most of the props are glued to the set. One can only manipulate objects the designer intended to be manipulable, in ways the designer intended they be manipulated.

Games are not comprehensive simulations of the real world. They do not have the infinite granularity of the physical universe, and do not force living entities to construct their own complex and contextual representations from scratch. Instead, they come with an ontology already built in, with prefabricated representations already imposed by the game's design. In games, all objects are explicit, and their affordances few and available for inspection. Granularity is only as detailed as it needs to be given the expected gameplay and the allowances of the development process.

This can simplify dialog modeling. A tightly constrained world model imposes limits on what can be done within the universe of the game; it becomes easier to represent conversation if the number of topics and activities is limited. Structure is easier to find in circumscribed contexts.

Due to this limitation, it can also be easier for the player to leave the boundaries of the game context—for example, to start a conversation about something completely unknown to the system. However, these violations can be recognized as such, and the designer can provide appropriate mechanisms to deal with them.

In the end, because games are simpler, and the designer has total control over the universe, it makes structural approaches that much more appealing. Indeed, as we have already observed, structural approaches, albeit very simple ones, are already very popular in game production.

2.5.4. Proceduralizing Émigrés

Being a stranger in a foreign land gives you an immediate, visceral understanding of the situated nature of communication. And it suggests an approach of how language-based communication could be implemented in artificial agents.

We know a considerable amount about getting through common social situations. We also carry highly contextual knowledge of language use in these situations. The intersection of this knowledge and language use suggests an intriguing, if inherently contextual, model of dialog. We can endow our systems with the ability to get through interactions even without full understanding of what is being said, by representing the details of human situations and the language used to get through them. Then more complicated reasoning can be built on top of such situated foundation.

What can be successfully proceduralized in this manner, and how such proceduralization should proceed, are still largely unknown. I hope this work presents a positive development towards answering these questions.

Chapter 3. Markov Modeling of Interaction.

Maďar se bavil s Němcem zvláštním způsobem, jelikož znal z němčiny jenom jawohl *a* was? *Když Němec mu cosi vykládal, Maďar kýval hlavou a říkal* Jawohl, *a když se Němec zamlčel, řekl Maďar* Was? *a Němec spustil znova.*

The Hungarian was having fun with the German in his own way, as the only German words he knew were *jawohl* and *was?* When the German was explaining something to him, the Hungarian kept nodding his head and saying *Jawohl*, then when the German finished talking, the Hungarian asked *Was?* and the German started all over again.

— Jaroslav Hašek. *Osudy Dobrého Vojáka Švejka za Světové Války.* IV-3.

Now I turn to the formal analysis of hierarchical structural models of interaction. First, to ground the discussion, I present the stochastic tool of hidden Markov models, and discuss their role in modeling interaction elements. Next, I introduce my extensions to the HMM approach that afford coupling between different independent models. Third, I show how this coupling can be used to represent the types of interdependence found in hierarchical approaches.

The result is a conversion of hierarchies of interdependent spaces, into a collection of coupled parallel spaces. While the outcome retains the causal dependence between the elements, it also allows for temporal independence. The following section then shows that this collection of parallel spaces is computationally equivalent to a flat HMM, but smaller and more efficient to update.

But first, a quick note on nomenclature. I use the term *sequences* to denote particular sequences of communicative elements that make up some particular interaction between humans, the term *protocols* for the unknown interpersonal mechanisms by which humans produce the meaningful sequences, and *processes* for the stochastic processes that represent these interaction sequences, albeit without any of the human knowledge about what goes on. Subsequently, *models* will refer to the hidden Markov models that follow the processes.

3.1. Hidden Markov Model Overview

Consider an unknown social protocol that gives rise to some particular interaction. We can observe the results of its engagement, as a sequence of communicative elements exchanged between the participants, such as requests, evaluations, demands, and other types of communicative acts. When such interaction exhibits a stereotyped structure, it can be successfully represented as a stochastic process.

A Markov process is a stochastic process that exhibits the Markov property: its state is dependent only on a *finite history* of previous states. Many natural phenomena can be approximated faithfully and efficiently through such simplification. Without loss of generality, we now examine first-order processes, which exhibit dependence on just the immediately previous state.

A hidden Markov model (HMM) is a model of the stochastic Markov process, where the state of the process is not necessarily directly observable. Even though the state cannot be observed, we can estimate it based on a history of evidence.

A discrete finite-state HMM is conveniently visualized as a directed graph. Edges are labeled with two factors, the probability p of state transition, and the probability q of observing some linguistic action during this transition; this linguistic action serves as evidence of the state transition.

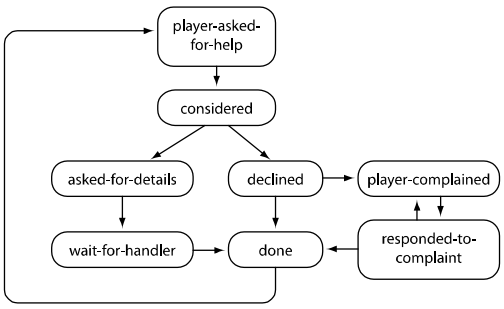


Figure 3-1. Topological view of a sample state space (sans edge details).

3.1.1. HMM DEFINITION

The formal definition is as follows. Let $\mathbb{P} = [0, 1]$ be the subset of $\mathbb{R}$ representing valid probability values. Let $\mathbb{M}$ be a collection of hidden Markov models, such that each $\mathbf{M} \in \mathbb{M}$ is defined as a tuple $\mathbf{M} = \langle S, A, p, q \rangle$, where:

$S = \{ s_1, s_2, \ldots \}$ is the set of discrete states, with a unique initial state s_1,

$A = \{ \gamma_1, \gamma_2, \ldots \}$ is the set of discrete communicative actions that can be observed,

$p : S \times S \rightarrow \mathbb{P}$ is the state transition probability, and

$q : S \times S \times A \rightarrow \mathbb{P}$ is the expected action observation probability.

To indicate the semantics of the probability functions, we will write p (s' | s) signifying probability of transition to s' from s, and we require that $\forall s \in S, \sum_{s' \in S} p(s' \mid s) = 1$. We also define q (γ | s, s') as the probability of action a being performed at transition from state s to s'. Note that this defines action as dependent on transition.

3.1.2. Belief Distribution Computation

Conversation state is not directly observable—we cannot assume to know where exactly we are at any given time—but we can estimate it given an HMM and a history of observations.

The belief distribution $b : \mathbb{Z} \times S \to \mathbb{P}$ is a function over states in the model, signifying the probability that the underlying process is at each of the different states at that point in time. The notation $b_t(s)$ denotes the probability of being in state s at the discrete time slice t, under the constraint that $\forall t \in \mathbb{Z}, \sum_{s \in S} b_t(s) = 1$.

For the initial time slice $t = 1$, define $b_t(s_1) = 1$, and $b_t(s) = 0$ for all $s \neq s_1$.

We can calculate b_t using the popular *forward algorithm* for hidden Markov models (Jelinek 1997, Jurafsky and Martin 2000). Consider a sequence of actions $\gamma_1, \gamma_2, \ldots$, etc. The probability of the entire model accepting this sequence is:

$$P(\gamma_1, \gamma_2, \ldots, \gamma_t) = \sum_s b_t^*(s)$$

This is the sum of the probabilities of all possible state sequences accepting these actions. Each individual sequence is defined as the probability of ending up in state *s* after accepting the sequence of actions. Because the system is Markovian, we can define this probability recursively, using the finite history window instead of the complete history.

$$b_t^*(s) = \sum_{s_i \in S} p(s \mid s_i)\, q(\gamma_t \mid s_i, s)\, b_{t-1}^*(s_i)$$

Our probabilistic model will likely be imperfect, and for large values of *t* this value may lose precision, so it will be necessary to renormalize it. If we choose positive values $\eta_{S,1}$, $\eta_{S,2}$, etc. for the state set S such that $\eta_{S,t} \sum_{s_i \in S} b_t^*(s) = 1$, then the normalized belief distribution function is:

$$b_t(s) = \sum_{s_i \in S} p(s \mid s_i)\, q(\gamma_t \mid s_i, s)\, b_{t-1}(s_i)\, \eta_{S,t} \qquad (1)$$

Since each time slice requires its own normalization value, we omit the index *t* of η_S unless an ambiguous context demands it. This formula will be the basis for our usage of Markov models.

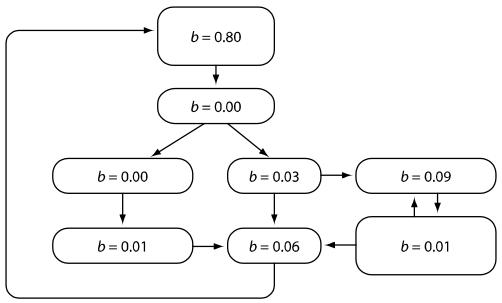


Figure 3-2. Belief distribution over the sample state space.

3.1.3. Action Observation on States

Discussion thus far treated action observation as a function of node transitions. In practice, however, it is much easier to use a Markov model with observation as a function of nodes. The two representations are interchangeable, and can be easily derived from each other.

The informal intuition is that we invert the state space, turning each transition into a state, and each state into a transition, thus tying action observation to states. More formally, following Jelinek (1997), we transform a transition-based model into a state-based model as follows. Define a new model **M***, whose state space is a product of the old state space with itself:

$$S^* = \{ (s, s') : s, s' \in S \}$$

Then define the action observation and transition functions as follows for each $s^*, s^{*'} \in S$:

$$q^* (\gamma \mid s^*) = q (\gamma \mid s, s')$$

$$p^* (s^{*'} \mid s^*) = \begin{cases} p (s'' \mid s') & \text{if } s^{*'} = (s', s'') \text{ and } s^* = (s, s') \\ 0 & \text{otherwise} \end{cases}$$

At first it may seem that the resulting state space will be a square of the original. This will be a matter of the edge density of the original graph. In practice, however, most of the newly generated states end up dead, and their elimination reduces the new space to a size comparable with the source.

The belief computation equation (1) can now be easily updated to work with state-based observation (Roy, Pineau and Thrun 2000, Zubek 2004). We omit the asterisks from the above, since the definition of q shows we are dealing with state-based observations:

$$b_t(s) = \sum_{s_i \in S} p (s \mid s_i) q (\gamma_t \mid s) b_{t-1} (s_i) \eta_{s,t} \qquad (1b)$$

My own implementation is based on state-based observations. Thus, in the following discussion, I will switch into the state-based model as needed, omitting the asterisks except when ambiguous context demands it.

3.1.4. Action Performance

An action performance policy is also necessary, to specify the actions to be generated by the system based on the belief distribution.

This is commonly approached as a matter of specifying some decision policy $\pi : (S \to \mathbb{P}) \to A$, which maps belief over the states to the space of actions. The policy is often calculated from the model and some action reward function—ideally, one tries to find an optimal policy, which always produces the best action. However, an optimal policy can be hard to find, depending on the problem at hand, and the issue of policy production and optimization will have to remain unaddressed here.

For the purposes of this section, I do not commit to any particular policy or policy production mechanism. It will suffice to say that a complete system will include *some* decision mechanism that drives action production. Later, I will discuss my particular choices and implementation in section 5.2.

3.2. Additional Architectural Choices

We should describe two additional architectural choices, which work in tandem with the hidden Markov models, to enable engaging and complex performance. Those two are: a method for evidence observation decomposition, such that it could be estimated as a product of simpler components, and a method for implementing a modicum of topic retention.

3.2.1. Action Observation Decomposition

Belief computation requires that the function $q(\gamma \mid s)$ be provided—it is the conditional probability of observing some communicative action given the state. We can use Bayes' rule to derive this conditional from simpler elements.

To refresh, we follow Bayes' rule:

$$P(\gamma \mid s)\, P(s) = P(s \mid \gamma)\, P(\gamma)$$

Substitute as follows:

$$e(\gamma) = P(\gamma)$$

$$g(\gamma, s) = P(s \mid \gamma) / P(s)$$

$$= c\, P(s \mid \gamma), \text{ where } c = 1 / P(s) = 1 / \sum_{\gamma \in A} P(s \mid \gamma)$$

$$q(\gamma \mid s) = P(\gamma \mid s)$$

This way we get a usable decomposition of the evidence observation function:

$$q(\gamma \mid s) = e(\gamma)\, g(\gamma, s)$$

The function *e* represents *evidence estimation*: it is the confidence level that some communicative action γ was actually observed from the system input, independently of the state. This is used to categorize the current input into likely speech acts, interpersonal actions, and other classes of expression, based on state-independent surface features. For example, the probability of a request should be quite high for a phrase beginning with *"can you",* but rather low for a phrase beginning with *"you can".* We will sometimes subscript *e* with the time value, to emphasize the dependency on the changing system inputs.

Note that it's quite possible for a given input to fit into a number of distinct categories simultaneously. An emotive gesture such as *"Rob nods"* can be interpreted in a number of ways: simultaneously as a possible sign of affirmation, as a greeting, as a positive response to a yes-or-no question, as a turn-taking acknowledgement in a conversation, and so on. Each of these evidence values can be non-zero independently of the others. This is valid behavior of the system, and it reflects the ambiguity of the inputs.

The *expectation function g* is the probability of that type of evidence leading to the given state. This ties the observed evidence to the specifics of the situation. For example, given the observation of a request, function *g* specifies whether the new state is likely given the request. This way, a protocol can specify which communicative acts accomplish movement through its different stages.

The multiplication of *e* and *g* accomplishes ambiguity resolution. Conceptually, *e* represents the likely interpretation of the input utterance, while *g* specifies whether such interpretation makes sense at that state of conversation. If the utterance fails to fit the interpretation, or the interpretation is unexpected, one of *e* or *g* will return a negligible or zero value, and therefore so will their product. The resulting value of *q* thus represents the *intersection of what was recognized with what was expected*.

While this separation may seem unnecessary, it aids the designer. The separation of act estimation on one hand, and act expectations on the other, allows for more flexible authoring. The communicative act abstraction helps deal with more ambiguous situations—and as we will see when discussing implementation, the evidence function *e* can be quite expressive when separated from the specifics of the particular model.

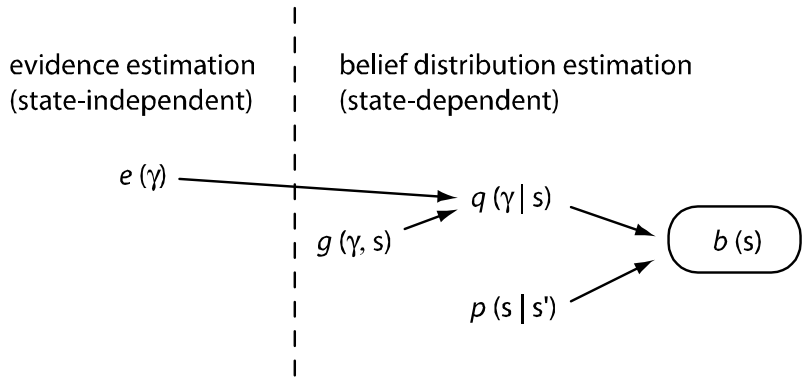


Figure 3-3. Functional view of belief distribution computation.

The separation also achieves a useful architectural simplification. Evidence estimation is independent of states and models; therefore it can be computed independently of the belief estimation, as seen in Figure 3-3. This will assist us in abstracting out interdependent elements.

3.2.2. Topic Retention

Topic retention is required, so that different spaces could share a reference to external elements such as conversational objects: for example, some item under consideration is mentioned, and subsequent utterances can refer to it without mentioning it explicitly.

Thus, in addition to the structure of the interaction, some memory of the topics of the interaction must also be maintained.

Some dialog models deal with this problem by embedding the different variants directly into the state space. For example, in the conversational robot by Roy, Pineau and Thrun (2000), the conversation includes separate states for the different TV stations one could talk about, such as *want-NBC-info*, *want-CBS-info*, or *want-ABC-info;* for the different locations where the robot could be ordered to go, such as *send-robot-to-kitchen, send-robot-to-bedroom,* and so on. This is a standard propositional expansion: each combination of variants and values is explicated in the state space.

Another approach, used in planning-based dialog systems, is to use predicate calculus representations, implementing the variants as straightforward, unrestricted memory-based variables. Unfortunately, this cannot be done easily with finite-state HMMs since they are weaker than Turing machines.

Instead, finite-state models can use situational variables to store and access variant values (based loosely on deictic representation by Agre and Chapman, 1987). This is essentially a task-sensitive global variable binding mechanisms, made up of a set of task-oriented variables (e.g. *the-topic*, *the-goal*, etc.) and an apparatus for monitoring and maintaining bindings.

The operational insight is that only a handful of task-specific variables will be used at any given point in time. For example, if we only need to track "the topic right now", then there is no need to implement this using multiple, general variables. The number of these variables can be easily assumed to be finite and small, without imposing excessive restrictions on the system designer. Furthermore, since the elements they track are inherently dynamic, one must ensure the variables are not out of step with reality—it is assumed that the tracking is computationally inexpensive, and the value of each variable is re-evaluated every time it needs to be accessed.

The system uses situational variables to track references to abstract conversational topics. We require that the system carry a structured representation of the different topics that can felicitously be talked about in the course of the interaction. The set of variables, and the structured set of possible bindings, are both relatively small and known in advance; thus we could model context-sensitive variables using a finite-state binding mechanism (Horswill 1998).

The particular implementation will be discussed in section 5.2. For now, it is important to emphasize that the state of the situational variables can be available for inspection by the system. Variable binding, in general or to some particular value, can be treated as a new action γ', its observation can be estimated through an estimator e (γ'), and used to advance or modify sub-interactions using inspection mechanisms described below.

3.3. Hierarchical Parallel HMMs

The following describes my extensions to the HMM approach that allow for the easy implementation of hierarchical-parallel activity.

First, I consider the decomposition of a Markov model into separate components. Second, I show how to implement state coupling between the components, without changing the basic nature of the models. Third, I show how hierarchical dependencies can be reduced to this coupling. With dependencies reduced, the resulting system becomes computationally equivalent to a collection of small, parallel HMMs.

3.3.1. Cartesian Decomposition

Large state spaces make belief computation expensive. However, it is often possible to decompose them into a number of smaller and simpler ones.

As has been mentioned before, two HMM decomposition approaches are popular: first, splitting the HMM state space into sub-regions, and second, finding different, smaller HMMs, whose combination reconstructs (or approximates) the original (Meuleau, et al. 1998, Boutilier, Dean, and Hanks 1999). I take the latter approach, also called the *cross-product* or *Cartesian* decomposition. Not all HMMs can be decomposed—but for those that can, the efficiency improvement is substantial.

Intuitively, a Cartesian product is one that can be decomposed into constituents, such that the cross product of their state spaces, and some modification of the probability functions, recreates the original Cartesian space. Figure 3-4 illustrates a sample Cartesian space, and its constituents.

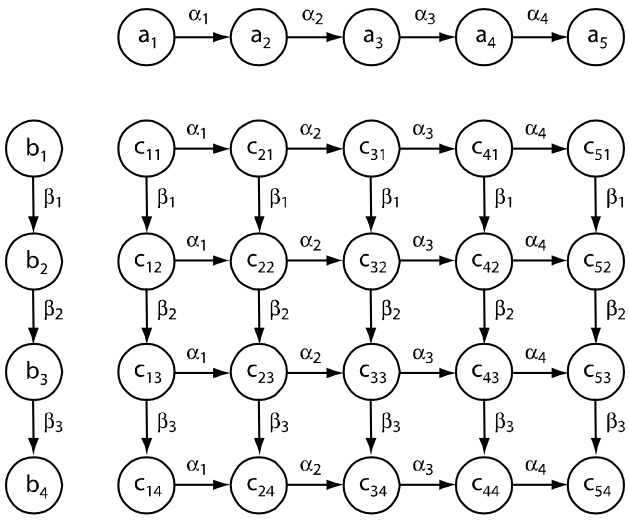


Figure 3-4. Cartesian model C, and its constituents, A and B.

Arrow between states indicates *p*, the probability of transition.

Symbol on arrow indicates *q*, the probability of production.

The constituents are completely independent HMMs, whose belief computation can be done in parallel, concurrently with each other. In section 3.4, I will show how the

constituents can be computationally equivalent to the Cartesian model under simple mapping functions; however, it is asymptotically faster to update their belief distributions than that of the product.

A model is considered *Cartesian* if it is in the range of the Cartesian composition operator $\times_C$. Define $\times_C : \mathbb{M} \times \mathbb{M} \rightarrow \mathbb{M}$ such that, given some $\mathbf{A}, \mathbf{B} \in \mathbb{M}$:

$$\mathbf{A} \times_C \mathbf{B} = \langle S_A \times S_B, A_A \cup A_B, p_C, q_C \rangle$$

Here, Cartesian evidence and transition probabilities p_C and q_C are defined as follows. Let ν be the number of spaces involved in the cross product; here, $\nu = 2$. Let the set $S_C = S_A \times S_B$ be indexed by the source state indices, such that $S_C = \{ c_{ij} : 1 \leq i \leq |S_A|, 1 \leq j \leq |S_B| \}$. Require that A_A and A_B be disjoint (if they are not disjoint, they can be made so by redefining either to be a new set, mapping one-to-one over the old one).

Then we define transition and action observation functions as follows:

$\forall$ $i, k \leq |S_A|, j, l \leq |S_B|, \alpha \in A_A, \beta \in A_B$:

$p_C(c_{ij} \mid c_{kj}) = p_A(a_i \mid a_k) / \nu$ (Transitions from the first constituent)

$p_C(c_{ij} \mid c_{il}) = p_B(b_j \mid b_l) / \nu$ (Transitions from the second constituent)

$q_C(\alpha \mid c_{kj}, c_{ij}) = q_A(\alpha \mid a_k, a_i)$ (Actions from the first constituent)

$q_C(\beta \mid c_{il}, c_{ij}) = q_B(\beta \mid b_l, b_j)$ (Actions from the second constituent)

Figure 3-5 illustrates how we calculate transition and evidence probabilities for some particular c_{ij}.

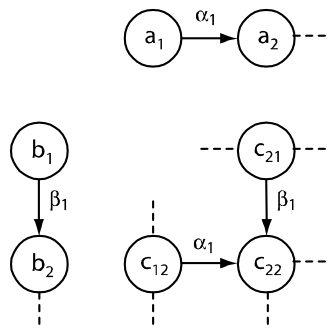


Figure 3-5. Cartesian product detail.

Observation: This preserves model validity, since $\forall c_{ij} \in S_C$, $\sum_{c_{kl} \in S_C} p_C(c_{kl} \mid c_{ij}) = 1$.

Proof: Let $m = |S_A|$, $n = |S_B|$.

For each $c_{ij} \in S_C$: $\sum_{k=1,l=1}^{m,n} p_C(c_{kl} \mid c_{ij}) =$

$$\sum_{k=1}^{m} p_C(c_{kj} \mid c_{ij}) + \sum_{l=1}^{n} p_C(c_{il} \mid c_{ij}) =$$

$$\sum_{k=1}^{m} p_A(a_k \mid a_i) / \nu + \sum_{l=1}^{n} p_B(b_l \mid b_j) / \nu =$$

$$2 / \nu = 1$$

■

We now turn to examine how constituent models can be coupled together.

3.3.2. Model Coupling

Independent HMMs are self-contained units, and their belief calculations do not take external elements into account—in other words, they do not afford the kind of interdependence required for hierarchical engagement. However, we can extend evidence estimation and action performance to implement a modicum of *coupling* between the spaces.

The term coupling is used ambiguously in the literature; it can mean any kind of a connection between models. For example, Brand (1997) uses coupling to denote joint probabilities across time slices, while Meuleau et al. (1998) couple their POMDPs through action policies, such that action production in one process affects the utility of productions in others. My approach is different still, although related in spirit to the work by Brand (1997) or Ghahramani and Jordan (1995), in that it couples belief distributions, rather than action policies.

I would like to suggest that state-to-state coupling be accomplished by turning belief distributions into something that can be observed. Information about other spaces can be encapsulated as evidence observation, and used to influence belief distributions or produce limited activity—in a sense, turning the HMMs inside-out, into a system that can observe itself.

This is accomplished by inspection—a mechanism by which the state belief distribution of one model can serve as evidence for belief calculation in another model in the system. Define it formally as follows. Given two models **A, B** $\in \mathbb{M}$ with states $s_A \in S_A$, $s_B \in S_B$, for each case when s_A *inspects* s_B, we augment A_A with a unique action γ' that signifies the inspection, and define e_t (γ') = b_{t-1} (s_B), and g (γ', s_A) = 1. Notice that inspection imposes the delay of one time slice.

One can also couple states via modification, by which state belief distribution of one model directly affects that of another within the system. Even though I moved away from this approach in favor of inspection, we can also consider it, for the sake of completeness. Given two models **A**, **B** $\in \mathbb{M}$ with states $s_A \in S_A$, $s_B \in S_B$, for each case when s_A *is modified by* s_B at *t*, we introduce a new action *a'* such that the performance of *a'* at time *t* defines b_t (s_A) = b_t (s_B), and without committing to a particular policy, we ensure that π_t (s_B) = *a'*. Note that action policies can only be evaluated after belief distribution update, so the effect of modification will only be useful at time *t+1*.

Both methods work across time slices—inspection uses the previous iteration to affect present belief, and modification changes present belief for use in the next slice. They happen, respectively, before and after belief estimation—it may be useful to conceptualize them as similar in spirit to the `:before/:after` methods of Common Lisp.

3.3.3. Types of Hierarchical Dependencies

Hierarchical engagement requires a representation of causal dependencies between the different elements of the hierarchy. We observe the following salient kinds of dependencies:

1. *State dependency.* This is when the state of one model influences the particular state of another. For example, a selling interaction may need to be rolled back a step if the player does not like the item.

2. *Model dependency.* This is when the state of one model influences whether an entire other model should be engaged or disengaged. For example, the activation of a selling interaction should enable a number of specialized protocols for barter and kissing up to the customer, but the end of the selling routine should disable them.

All possible dependencies are innumerable, but these two are crucial for hierarchical models, and can be reduced to coupling methods. The reduction proceeds as follows.

3.3.4. State Dependency

State dependency happens when state belief in one space influences state belief in another space. Consider two states, the controlling state $s_C \in S_C$, and the target state $s_T \in S_T$, such that the belief in the controlling state is intended to influence the belief in the target state.

We implement this dependency via inspection. Set s_T to inspect s_C via some modification function *f*: define some unique action γ_C to correspond to the inspection, and define $e_t(\gamma_C) = b_{t-1} \circ f(s_C)$, and $g(\gamma_C, s_T) = 1$. The result is that the target's belief value will be a function of the controller's previous belief value.

3.3.5. Model Dependency

Model dependency happens when the state of one space influences the engagement in an entire other state space; for example, some particular state belief leads to an entire different space being completely disabled, as in push-down hierarchical models.

We implement this dependency via inspection. Consider the target state space S_T and some controlling state s_C belonging to a different space. Allow S_T to contain a unique disable state s_0, with incoming links from all states, and an outgoing link to the initial state s_1; belief distribution over s_0 signifies confidence that the entire space is disengaged.

Specify that s_0 inspects s_C in two ways: there exist two unique actions γ_E, γ_D that correspond to the engagement and disengagement of the subspace, mediated through some function *f*. Define $e_t(\gamma_E) = b_{t-1} \circ f(s_C)$, $e_t(\gamma_D) = 1 - b_{t-1} \circ f(s_C)$, and expectations $g(\gamma_D, s_0) = 1$, and $g(\gamma_E, s_1) = 1$. The result is that the engagement and disengagement of the target space will be a function of the controller's previous belief value.

3.4. Cartesian Composition Details

I now examine the details of Cartesian composition, and show that belief updates on constituents are equivalent, but asymptotically faster than on a flat Cartesian model. As mentioned before, different approaches to Cartesian composition have been used in other work on the simplification of Markov models (Brand, Oliver, and Pentland 1997, Meuleau et al. 1998), and the following describes my particular approach.

The diagram of a Cartesian composite is reproduced below, in Figure 3-6.

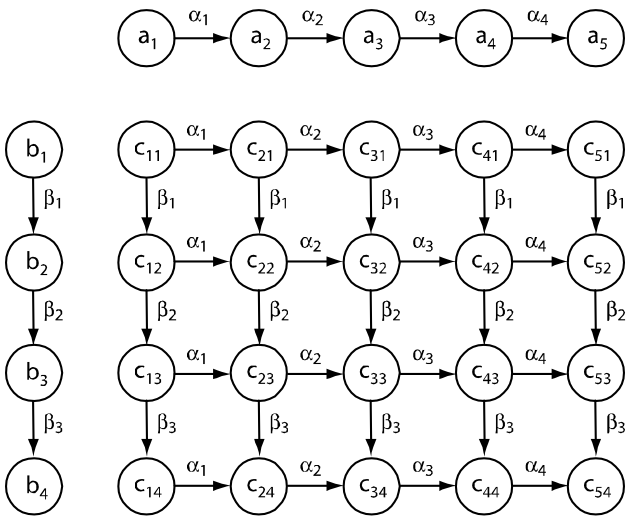


Figure 3-6. Cartesian model C, and its constituents, A and B.

3.4.1. Proof of Equivalence

Informally speaking, when I say that parallel models are equivalent to a Cartesian model, I mean that a Cartesian belief distribution can be reconstructed from individual parallel belief distributions, but *without actually having to construct the Cartesian model itself.* The formal definition is as follows.

Definition. For each $\mathbf{C} \in \mathbb{M}$, iff there exist $\mathbf{A}, \mathbf{B} \in \mathbb{M}$ such that $\mathbf{A} \times_C \mathbf{B} = \mathbf{C}$, we say **C** is separable into **A** and **B**.

Definition. For some HMMs **A, B, C** $\in \mathbb{M}$ with belief function $b : \mathbb{Z} \times S_C \rightarrow \mathbb{P}$, given some belief mapping function $b' : \mathbb{Z} \times S_A \times S_B \rightarrow \mathbb{P}$ and a state mapping function $s' : S_A \times S_B \rightarrow S_C$:

A and **B** are *equivalent* to **C** under b' and s' if:

1. $\mathbf{A} \times_C \mathbf{B} = \mathbf{C}$ (constituency requirement), and
2. $b' = s' \circ b$ (belief equality requirement)

Observation. The belief equality requirement means that:

$$\forall\, a_i \in S_A, b_j \in S_B : b_t'(a_i, b_j) = b_t(s'(a_i, b_j))$$

Claim of Parallel Equivalence. For all Cartesian models as defined earlier, their constituents are equivalent to the Cartesian model under:

$$b_t^P(a_i, b_j) = b_t(a_i)\, b_t(b_j), \text{ and}$$

$$s^P(a_i, b_j) = c_{ij}$$

Before we attempt to prove this claim, we need a few extra tools.

Lemma 1. Initial belief equality under $\mathbf{b_t^P}$ and $\mathbf{s^P}$.

For any Cartesian model **C** separable into **A** and **B**, belief equality holds at time $t = 1$.

Proof. By definition, at $t = 1$ spaces S_A, S_B, and S_C begin at their initial states, a_1, b_1, and c_{11}, respectively. Therefore:

$b_t^P(a_1, b_1) = b_t(a_1)\, b_t(b_1) = 1$

$b_t(s^P(a_1, b_1)) = b_t(c_{11}) = 1$

$\forall\, a_i \neq a_1, b_j \neq b_1, c_{ij} \neq c_{11} :\ b_t^P(a_i, b_j) = b_t(a_i)\, b_t(b_j) = 0$

$\forall\, a_i \neq a_1, b_j \neq b_1, c_{ij} \neq c_{11} : b_t(s^P(a_i, b_j)) = b_t(c_{ij}) = 0.$

■

Lemma 2. Subsequent belief equality under b_t^P and s^P.

For any Cartesian model **C** separable into **A** and **B**, if only one production is observed between times *t-1* and *t*, previously existing belief equality is preserved.

Proof. Assume previous equality under b_{t-1}^P and s^P holds, such that $b_{t-1}^P(a_i, b_j) = b_{t-1}(s^P(a_i, b_j)) = b_{t-1}(c_{ij})$. Given $m = |S_A|$, $n = |S_B|$, and some observed production $\gamma_t \in A_C$, it follows by definition of b and $\times_C$ that:

$$b_t(c_{ij}) = \sum_{k=1,l=1}^{m,n} p(c_{ij} \mid c_{kl})\, q(\gamma_t \mid c_{kl,}\, c_{ij})\, b_{t-1}(c_{kl})\, \eta_C \qquad (2)$$

$$= \sum_{k=1}^{m} p(c_{ij} \mid c_{kj})\, q(\gamma_t \mid c_{kj,}\, c_{ij})\, b_{t-1}(c_{kj})\, \eta_C +$$

$$\sum_{l=1}^{n} p(c_{ij} \mid c_{il})\, q(\gamma_t \mid c_{il,}\, c_{ij})\, b_{t-1}(c_{il})\, \eta_C$$

$$= \sum_{k=1}^{m} p(c_{ij} \mid c_{kj})\, q(\gamma_t \mid c_{kj,}\, c_{ij})\, b_{t-1}{}^{P}(a_k, b_j)\, \eta_C +$$

$$\sum_{l=1}^{n} p(c_{ij} \mid c_{il})\, q(\gamma_t \mid c_{il,}\, c_{ij})\, b_{t-1}{}^{P}(a_i, b_l)\, \eta_C$$

$$= \sum_{k=1}^{m} p(a_i \mid a_k)\, q(\gamma_t \mid a_{k,}\, a_i)\, b_{t-1}(a_k)\, b_{t-1}(b_j)\, \eta_C / \nu + \qquad (3)$$

$$\sum_{l=1}^{n} p(b_j \mid b_l)\, q(\gamma_t \mid b_{l,}\, b_j)\, b_{t-1}(a_i)\, b_{t-1}(b_l)\, \eta_C / \nu$$

If $\gamma_t \in A_A$ then $\forall\, b_j, b_l \in S_B : q(\gamma_t \mid b_{l,}\, b_j) = 0$, and $b_t(b_j) = b_{t-1}(b_j)$,

we set $\eta_C = \eta_A \nu$, and consequently from (3):

$$b_t(c_{ij}) = \left[\sum_{k=1}^{m} p(a_i \mid a_k)\, q(\gamma_t \mid a_{k,}\, a_i)\, b_{t-1}(a_k) \right] b_{t-1}(b_j)\, \eta_A + 0$$

$$= \left[\sum_{k=1}^{m} p(a_i \mid a_k)\, q(\gamma_t \mid a_{k,}\, a_i)\, b_{t-1}(a_k)\, \eta_A \right] b_t(b_j)$$

$$= b_t(a_i)\, b_t(b_j)$$

If $\gamma_t \in A_B$ then $\forall\ a_i, a_k \in S_A : q(\gamma_t \mid a_k, a_i) = 0$, and $b_t(a_i) = b_{t-1}(a_i)$,

we set $\eta_C = \eta_B \nu$, and consequently from (3):

$$b_t(c_{ij}) = 0 + [\sum_{l=1}^{n} p(b_j \mid b_l)\, q(\gamma_t \mid b_l, b_j)\, b_{t-1}(b_l)]\, b_{t-1}(a_i)\, \eta_B$$

$$= [\sum_{l=1}^{n} p(b_j \mid b_l)\, q(\gamma_t \mid b_l, b_j)\, b_{t-1}(b_l)\, \eta_B]\, b_t(a_i)$$

$$= b_t(b_j)\, b_t(a_i)$$

■

Lemma 3. Total belief equality under b_t^P and s^P.

For any Cartesian model **C** separable into **A** and **B**, given any sequence of temporally disjoint productions, parallel spaces satisfy the belief equality requirement with Cartesian model under b^P and s^P.

Proof. The claim is true if $\forall\ t \in \mathbb{Z}, a_i \in S_A, b_j \in S_B : b_t^P(a_i, b_j) = b_t(s^P(a_i, b_j))$. By induction, if $t = 1$, then they are equal by Lemma 1. If $t > 1$, and they were equivalent at $t - 1$, then they remain equal by Lemma 2.

■

Finally, we prove the original claim of equivalence.

Theorem of Equivalence. For any Cartesian model **C** separable into **A** and **B,** the constituents are equivalent with the Cartesian model under b^P and s^P.

Proof. Both conditions of compositional equivalence hold:

1. $\mathbf{A} \times_C \mathbf{B} = \mathbf{C}$ by definition of separability.
2. $b_t^P(S_A, S_B) = b_t(s^P(S_A, S_B))$ by Lemma 3.

■

3.4.2. Performance Improvement

Parallel spaces offer asymptotically better performance than a Cartesian space. Recall that belief distribution over the entire space is a matter of computing (1b) over each state:

$$\sum_{s \in S} b_t(s) = \eta_{S,t} \sum_{s \in S} \sum_{s_i \in S} p(s \mid s_i)\, q(\gamma_t \mid s)\, b_{t-1}(s_i)$$

The computation of the expression $p(s \mid s_i)\, q(\gamma_t \mid s)\, b_{t-1}(s_i)$ for any combination of s and s_i is invariant on the number of states, and bounded by some $O(c)$. The computation of normalization value η is bounded by $O(|S|)$. Thus the calculation of belief distribution over the entire space $b_t(S)$ is bounded by $O(c\,|S|^2)$.

Let a Cartesian model **C** be given. Since it is in the range of $\times_C$, it can be decomposed into constituent models $\mathbf{M_1}, \ldots, \mathbf{M_n} \in \mathbb{M}$, such that $\mathbf{C} = \mathbf{M_1} \times_C \ldots \times_C \mathbf{M_n}$.

Let $m = max\ (|S_1|, \ldots, |S_n|)$ be the size of the largest state space of those models. It follows from the definition of $\times_C$ that $|S_C| \leq m^n$. Thus, belief computation over a Cartesian composition is bounded by $O\ (c\ m^{2n})$.

The parallel space model is more tightly constrained. Each parallel space computes its belief distribution individually, bounded by $O\ (c\ |S|^2)$, but now there are n separate computations of at most m-sized spaces. The cost of belief computation over parallel composition is bounded by $O\ (c\ n\ m^2)$.

3.4.3. Hierarchical parallel property revisited

The hierarchical parallel property complicates modeling; both causal interdependence and temporal independence are demanded of the system at the same time. Treating dialog as a composite of simpler, separate elements makes this tractable, due to better knowledge authoring and simpler computation.

Extending the HMM approach to include simple coupling allows us to build large sets of independent, coupled space models. These run inexpensively and in parallel, thus providing the necessary temporal independence. Coupling also allows for a modicum of hierarchical control, thus providing the desirable causal interdependence.

Chapter 4. Systems and Results.

Sweet is the remembrance of troubles when you are in safety.

— Euripides, *Andromeda*

Given the algorithmic description of the last chapter, I now examine systems implemented using the hierarchical-parallel approach, and the behavioral results of using a flat collection of coupled HMMs. Note this is the discussion of their behavior—details of implementation are presented in the following chapter.

4.1. Implemented Systems

I implemented two systems for two video game characters, concentrating on different strengths of the approach. *Xenos the Innkeeper* is a stereotypical computer non-player character (NPC) for a fantasy role-playing game, who sells items and offers hints about things to do in the game. The demo concentrates on fallback mechanisms and graceful performance degradation, thanks to multiple redundant representations of the actions supported by the system. *The Breakup Conversation,* on the other hand, is a conversational "sim game" that offers a parody of the familiar personal drama: the conversation at the end of a relationship. It demonstrates an implementation of broad competence—even in a situation as extreme and nuanced as a breakup conversation, the system can nevertheless find its way through.

Both systems share similar performance characteristics. The hierarchical-parallel spaces introduce multiple levels of representation of the same phenomenon, and therefore increase redundancy—even if some particular space loses track of the interaction, its parents retain broad understanding of what goes on, and can cope with

the situation. This allows for gentle performance degradation at edge of competence, and steering the conversation back to familiar grounds.

Colloquial language used in games is usually noisy and ambiguous, and the stochastic approach helps us greatly in its resolution. In particular, the separation of observations into evidence estimation and expectation matching was immensely beneficial. Informally speaking, evidence estimation presents the system with multiple guesses about the player's utterance, which are then matched against the state-specific expectations. In this manner, even ambiguous inputs become handled if they were both partly recognized, and expected in the situation; this aids in graceful recovery from confusing and uncertain situations.

The result is an interaction that is significantly more complex than standard deterministic finite-state (or pattern-matching) techniques, without a significant increase in processing cost. The approach manifests long-term consistency of a level unexpected in conversation trees, including coherence within a particular interaction 'stage', and coherent transitioning between stages.

4.1.1. Xenos the Innkeeper

Xenos the Innkeeper is an NPC for the fantasy game Neverwinter Nights. The character is a barkeeper, who sells items, introduces quests, or offers some bar goods via natural language conversation. The language utterances are deliberately ungrammatical, set up

to sound like “foreigner’s English” by the omission of pronouns and articles, to curb the player’s expectation of fluency. Figure 4-1 presents a screenshot of the character being engaged in the game.

Figure 4-1. Xenos the Innkeeper screenshot.

Xenos conversation fragment.
Computer's utterances and emotes in **boldface:**

Player says: hi there
Xenos says: greetings
Player says: whats going on
Xenos says: me just working, you know
Xenos nods: hmm
Player says: :)
Xenos says: i have something for you
Xenos says: what about a task?
Player says: what is it
Xenos glances
Player says: well?
Player says: what
Xenos says: the mayor has a letter for me
Xenos says: could you bring it from the mayor?
Player says: how much are you paying
Xenos says: i asked you a question
Xenos nods: you will get a free lunch
Player says: not enough
Xenos says: please?
Player says: i need a new sword first
Xenos smiles
Xenos says: i have for you sword
Player says: a good one?
Xenos says: it's a spectacular sword
Player says: how much
Xenos says: i'll trade for a health potion
Xenos nods: you get a free meal
Player says: i don't have one
Xenos says: too bad
Xenos says: me beg you
Player says: for what
Xenos smiles
Xenos says: i need to get a letter from the mayor
Xenos says: can you go bring it from the mayor?
Player says: fine
Xenos says: really?
Xenos says: what about an important letter?
Player says: i'll bring it
Xenos smiles: that's fantastic!
Xenos nods: well off you go then
Xenos says: you've got to hurry

Figure 4-2. Xenos conversation excerpt.

The system is composed of different protocols, concentrating around the agent's three core competencies: selling weapons, serving bar goods, and introducing possible quests. Figure 4-2 presents an excerpt from one such interaction.

In addition to the core models, the system contains more general, redundant models of interaction; for example, in addition to the particular models of a weapon request, it also includes models for dealing with requests in absence of a recognized topic, for dealing with topic mention in absence of a recognized speech act, and for dealing with player's utterance in absence of both recognized speech act and topic, based on its perceived emotive components. All of these are active at the same time, all the time, and monitor the ongoing interaction; if the player steps outside of the specific model's area of competence, for example by using an unexpected speech act, the interaction will fall back onto more general components, which will deal with the situation.

The result is an interaction system that exhibits the desirable hierarchical parallel property: the interaction retains hierarchical composition, into levels of sub-interactions which retain control over each other, but at the same time allows for the parallel and independent engagement of different levels of generality. The redundant representation softens the performance drop-off at the edge of competence, and can be used to mimic some of the human ways of dealing with error.

4.1.2. The Breakup Conversation

The *Conversation* is an application of the system to a very broad context. The situation is a parody of the conversation at the end of a romantic relationship—the player is expected to perform a breakup over instant messenger, with the computer playing the role of the soon-to-be-ex, who will attempt to guilt and emotionally manipulate the player to give in and abort the breakup. The title is a *sim* rather than a *game* in the strict sense of the word, in that it presents the interaction as a goal in itself, rather than being oriented towards a particular winning condition.

The player begins by selecting the personal information of the simulated soon-to-be-ex, then a window that simulates instant messenger opens, as illustrated in Figure 4-3 and Figure 4-4. The player engages in the natural language interaction, happening in real time entirely via typed text, which takes the player on an exploration of the breakup space. Figure 4-5 presents an example excerpt from such a conversation.

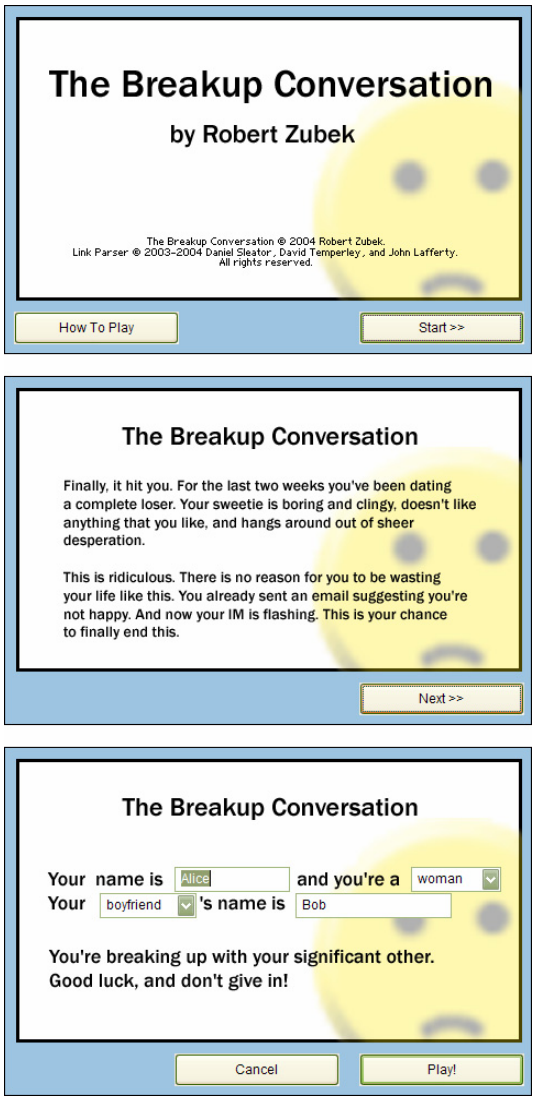


Figure 4-3. Breakup Conversation intro screenshots.

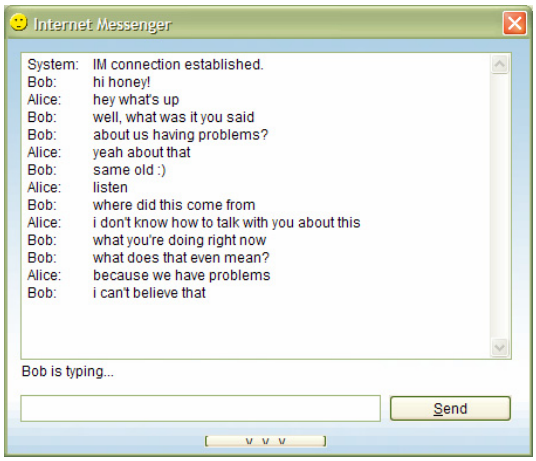


Figure 4-4. Breakup Conversation gameplay screenshot.

Breakup Conversation excerpt.
Computer's utterances in **boldface.**

Alice: hey there
Bob: how are you
Alice: not bad :)
Alice: and you?
Bob: what was that you said
Bob: that we should talk about the relationship?
Alice: yeah about that
Bob: :P
Alice: look
Bob: what did i do
Bob: look why wouldn't you just talk to me
Alice: no it's not you
Bob: oh really? then what?
Bob: what does that even mean?
Alice: it's me
Bob: i know it's me
Bob: why wouldn't you just talk to me about what's going on
Alice: i was thinking about us
Bob: it's because i stay at home
Bob: you loathe that i'm lazy
Bob: don't you
Alice: no i don't loathe you
Bob: it's not because of that?
Bob: how can you do this to me
Bob: then what is it?
Alice: i don't know
Bob: :(
Bob: i suppose i'm a naive loser
Alice: i'm feeling, well, like i'm suffocating
Bob: you're just messing with me
Alice: i need space
Bob: i can't believe that
...
Bob: you can't seriously be doing this
Bob: don't you love me?
Alice: i thought i did
Bob: but no longer?
Alice: i don't know
Bob: well then
Bob: do you hate me
Alice: you do things
Bob: do you despise me
Bob: did you really make up your mind?

Figure 4-5. Breakup Conversation excerpt.

Even though the situation is not stereotyped in the same way as a fantasy barkeeper's job, the system can still manage the conversation by recasting it into its own scripts. The computer has the knowledge of some patterns typical of a breakup conversation and the ways of getting through them: these include the "it's not you it's me" ritual, the "why are you doing this to me" blame, the refusals to discuss issues, and other ways in which people panic, reason, plead, lay guilt, and so on. The rituals are treated informally, and meant to be exaggerated and entertaining, rather than psychologically felicitous—they were inspired by Berne's interpersonal games (1964), as well as sitcoms and personal anecdotes. The computer agent will attempt a number of different guilt games, and the player's goal is to not give in. And while this approach is not capable of dealing with the player's invention of new background information or reasons, it is at least able to steer the conversation in a direction familiar to the system.

The overall interaction is modeled by decomposing breakup conversations into a hierarchy of simpler protocols, corresponding to breakup components. The highest-level protocols coordinate general 'stages' of a breakup – e.g. reasoning with the player about the breakup, making them feel guilty about it, and so on. Below them are protocols for getting through particular stages – for example, the guilt-laying stage will decompose into a number of strategies involving emotional blackmail and pleading for pity. At the bottom of the hierarchy we finally have very specific, low-level protocols: reacting to the player's evaluations, reacting to an apology, offering apology, making a

particular emotional blackmail maneuver, recognizing a rationalization, recognizing a breakup reason, trivializing the reason, rejecting the reason, and so on.

4.1.3. System Overview

To consider how the system accomplishes its effects, let us quickly examine its architecture. The following is just a quick overview—full details will be presented in the next chapter.

Recall that each HMM is represented using a separate, parallel state space, with optional coupling. The main engine is composed of a sequence of stages: estimation of all communicative actions, belief computation of each HMM, and action generation over all HMMs. In particular, at each iteration of the control loop, the following stages are executed in order.

- *Parsing and preprocessing* takes utterance from the player, and prepares it for Markov state estimation.
- *Evidence estimation* evaluates *e* for each speech act and communicative action, based on the outputs of the preprocessor. This includes evidence for coupling mechanisms and topic tracking.
- *State estimation* computes the new belief distributions for each parallel space, given the evidence and the last known state.

- *Action production and generation* evaluates action production policies for all spaces, arbitrates among them to produce a single output of the system; this gets translated into specific utterances or emotes, and sent to the game.

Figure 4-6 illustrates the architecture. Thick arrows correspond to data flow between the stages. Dotted lines, on the other hand, represent the effect of coupling. This can be thought of as "backward" influence over a time step boundary, since inspection links the evidence estimation to last iteration's beliefs, and modification links beliefs to last iteration's actions.

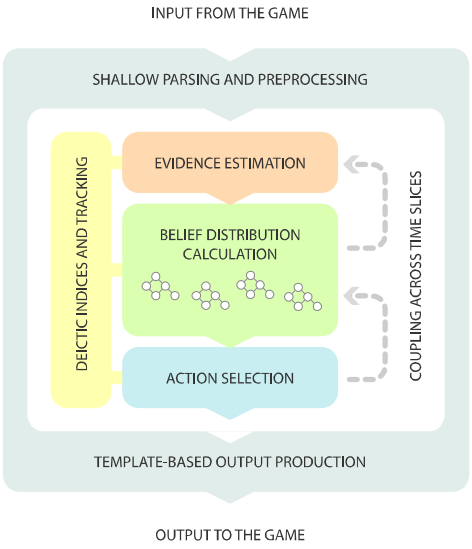


Figure 4-6. Engine architecture.

Thus the incoming utterance 'flows' through the system in a simple, single pass. First, the utterance is preprocessed, and turned into a list of words. Second, communication estimators and variable trackers trigger off the list of words, and the last known state of the system. Third, the belief is re-estimated based on the estimators and the last known state as well. Finally, action policies suggest an action to produce based on the new belief distributions. In other words, the stages form a linear pipeline, with each stage executed once to process the results of the previous stage.

Data flow across the entire engine is diagrammed in Figure 4-7. Details of computing just one state space pulled out from the overall system is then shown in Figure 4-8—this is the same space as previously introduced in Figure 3-1, rearranged for readability. Since each space is completely independent, this separation is easily done. The left side of the figure represents the effects of evidence estimation used by each state. Each estimator is implemented using pattern matching over the inputs, except for the inspection estimator *"is help-handler done",* which is a simple lookup of *b* from previous iteration (cf. implementation details in section 5.2.3).

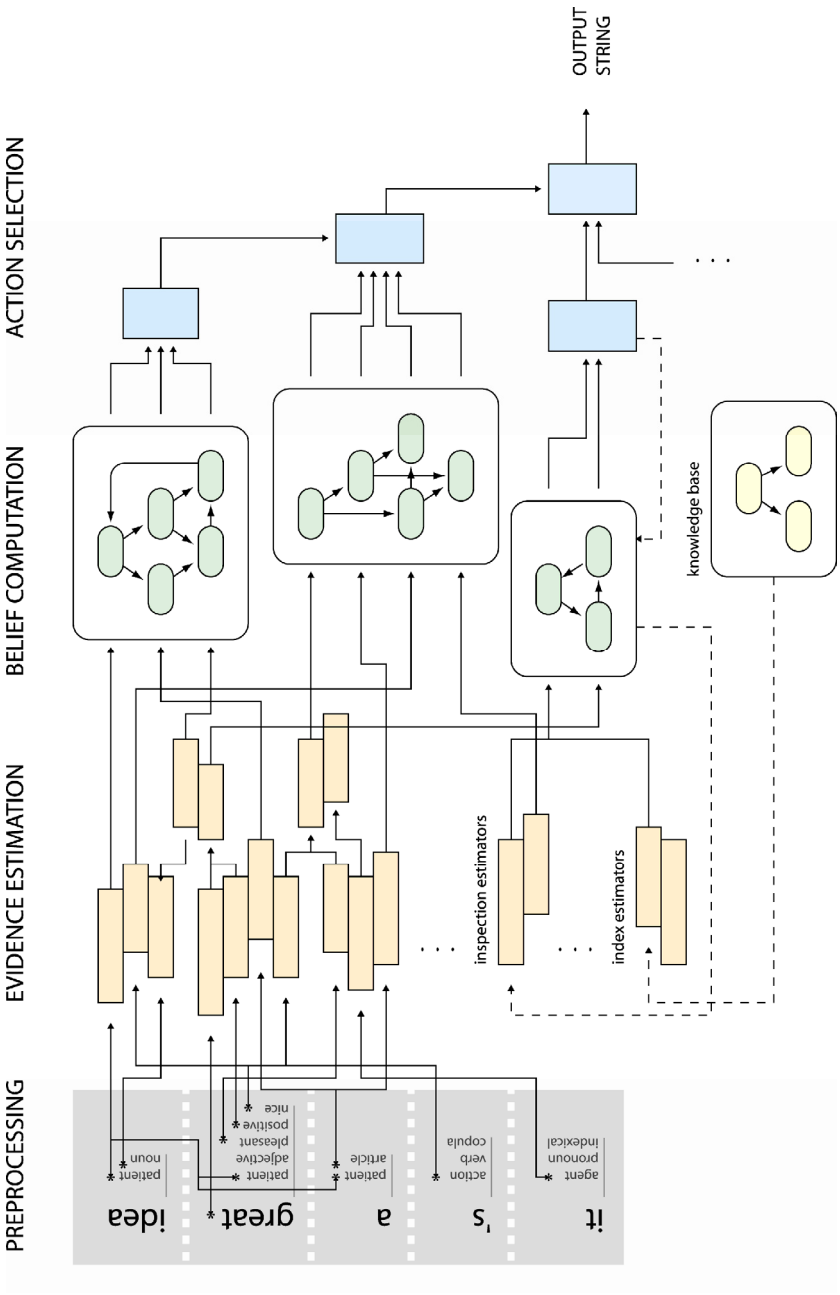


Figure 4-7. Data flow overview.

(Solid lines signify direct data flow, dashed lines signify coupling)

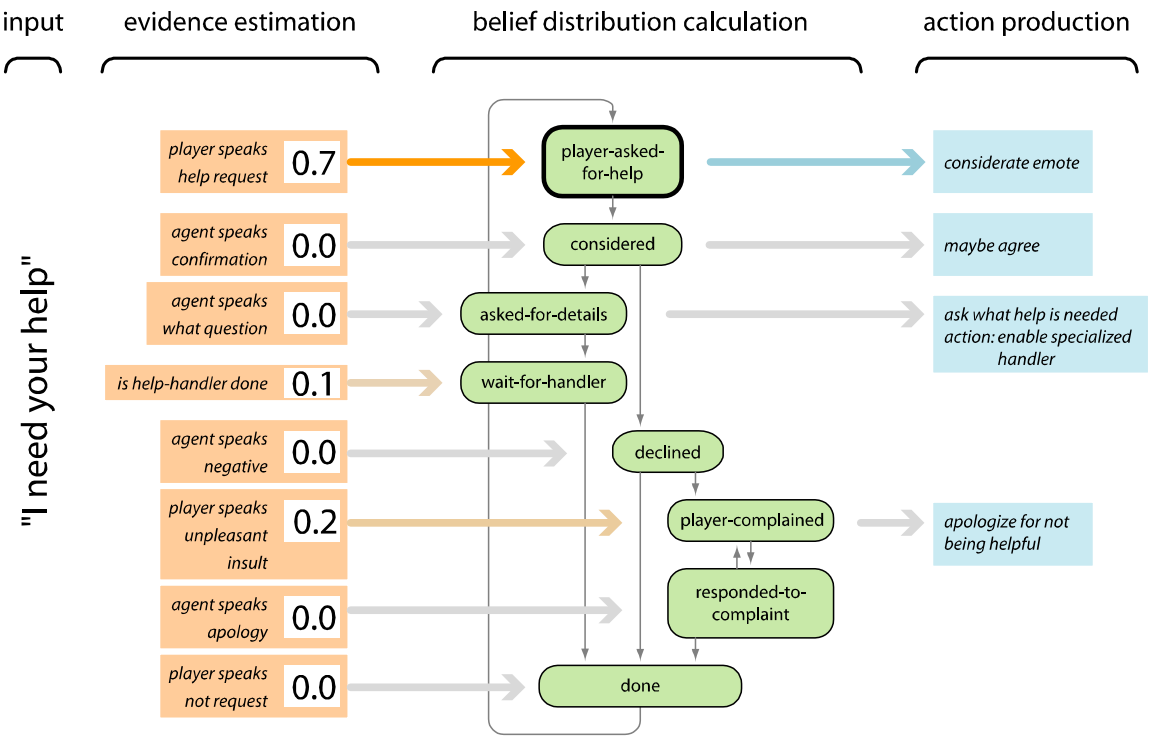


Figure 4-8. Data flow details.

Table 4-1 examines the different spaces used in the two implementations—it is a rough overview, meant to provide a sketch of the different sub-components of the overall interaction.

Shared Spaces

Low level monitors:
- Conversation timer
- Silence monitor
- Monologue monitor

Ambient movement:
- Fidget machine
- Turn enforcement
- Ambient emote
- Stock response
- Turn monitor
- Topic monitor

Insult management:
- Direct insult
- Indirect insult
- Insult accumulation

General routines:
- Topic recognized but not the form
- Question recognized but not topic
- Question about object
- Question about health
- Request general
- Request item
- Disagreement
- Player evaluation
- Agent evaluation
- Condemnation monitor

Conversation structure:
- Greeting
- Intro conversation
- Outro conversation

Breakup

Breakup intro:
- Allude to breakup
- Giving in monitor
- Guilting coordinator

Guilting: Self-pity
- Self-criticize
- Reject compliment
- "You must hate me"
- "Why are you mean"
- "Will you help me"

Guilting: Indignation
- "I thought you loved me"
- "I thought you cared"
- "I don't deserve this"
- "How can you do this"

Guilting: Pleading
- Beg for second chance
- Promise change
- "But I love you"

Guilting: Reasoning
- Guess at reason
- Demand reason
- Evaluate reason
- Treat as excuse
- Deny reason

Panicking:
- Silence
- Impatience monitor
- Resignation monitor
- Rejection monitor
- Start panic

Xenos

Quests:
- Quest monitor
- Perform quest injection
- Deal with agreement
- Deal with rejection
- Rush the player

Special routines:
- Job request
- What question
- Where question
- Payment question
- Evaluate object
- Barter for object

Table 4-1. Outline of the different spaces used in the two implementations.

4.1.4. System Performance

Two figures that follow illustrate the performance of *The Breakup Conversation,* version from November 2004. This latest implementation includes 958 estimator fragments, and 75 state spaces, ranging from 1 to 11 states each, whose union contains a total of 417 states. Estimator fragments are individual word or sequence tests, possibly reused across a number of estimators, as well as combination operations.

Latest version of *Xenos* includes 629 estimator fragments, and 38 spaces, which together consist of 234 states. *Xenos* is demonstrably less complex than the *Conversation;* at the same time, it is more difficult to benchmark, because it runs as part of a larger game server. Therefore, I only present performance numbers for the *Breakup Conversation* implementation.

The results presented below show average processing time taken per iteration of the system, as a function of input length. The data was based on 1150 iterations on a 1.8GHz Pentium 4. The system ran one iteration per second on the average.

Performance benefits greatly from the system's straightforward design. Figure 4-9 demonstrates the overall performance of the engine, including parsing using the Link parser, and all stages of the engine—but excludes user interface and presentation elements.

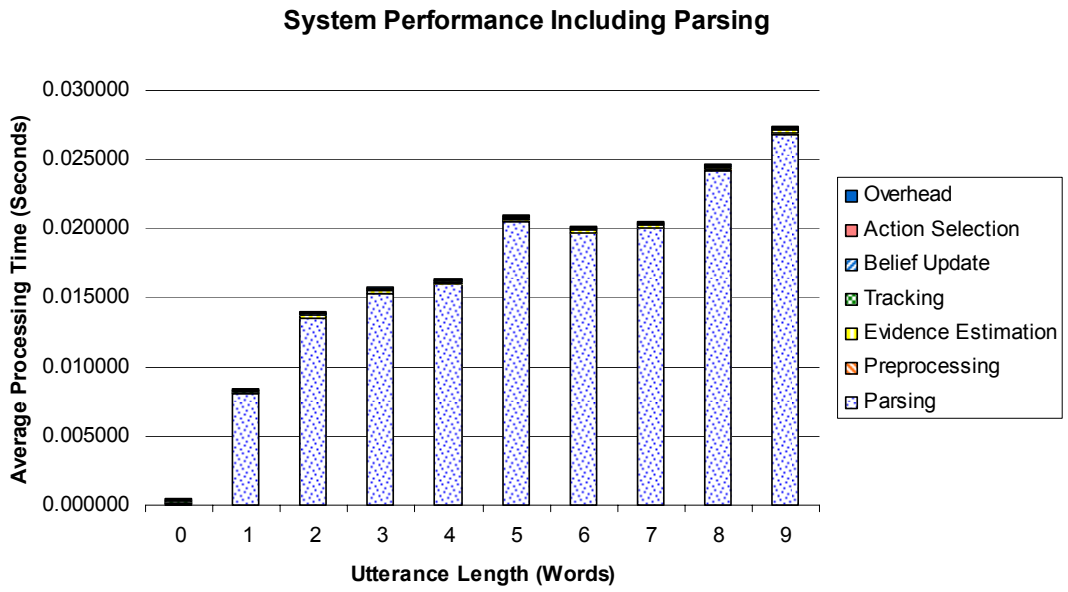


Figure 4-9. System processing time, including the parser.

The Breakup Conversation, version from November 2004.

As the figure shows, the system's performance is roughly linear with utterance length, has the average base cost of 10 milliseconds, plus 3 milliseconds per word. Processing is dominated by parsing in the external Link parser.

Figure 4-10 illustrates the performance of my part of the system—it shows the timing results for all parts of the system *except* parsing.

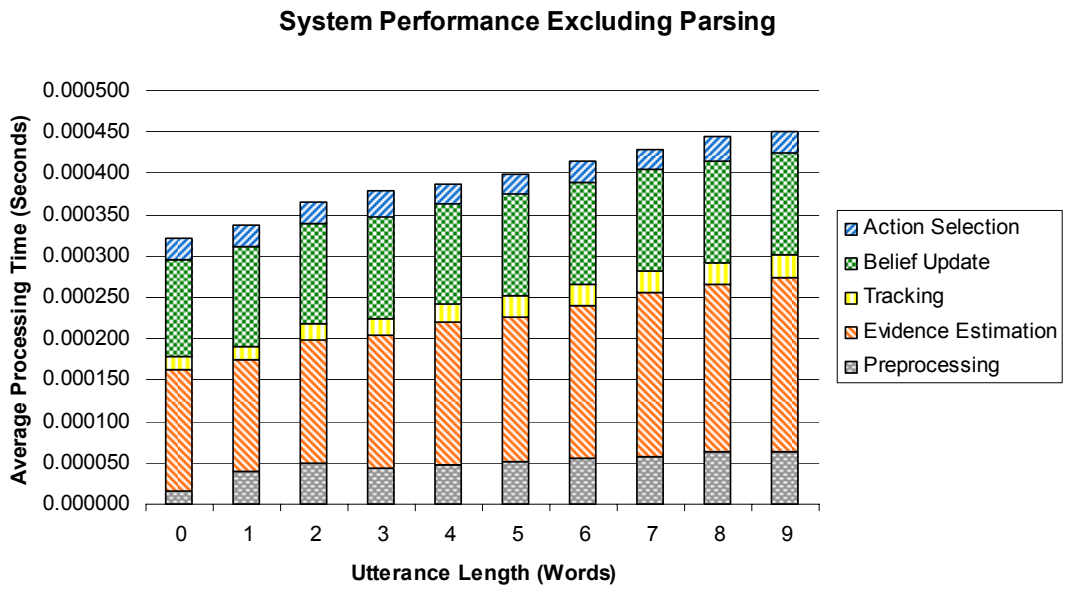


Figure 4-10. System processing time, excluding the Link parser.

The Breakup Conversation, version from November 2004.

The average processing cost for the entire hierarchical parallel system, including evidence estimation, belief update, and action production, is less than 500 *micro*seconds per iteration.

Additional technical details about engine implementation are provided in the next chapter; now we turn to the details of how hierarchical effects are achieved using the previously introduced coupling.

4.2. Behavioral Results: Interleaving, Fallback, Recovery

I can now discuss the details of hierarchical coupling, and how it allows for the implementation of interleaved interaction, as well as fallback and recovery mechanisms.

First, I would like to illustrate hierarchical coupling in greater detail. Then I describe two particular classes of errors supported by the hierarchical parallel approach. First is the case of getting lost in interaction, resolved by fallback on simpler representations of activity. Second is the case of failing to advance in interaction, which is addressed by reflexively monitoring the state of the system.

4.2.1. Hierarchical Coupling

Parent-child dependency requires that parents be able to enable or disable the children, and children be able to inform the parents about their progress. Figure 4-11 illustrates how this dependency is used to implement hierarchical engagement in *The Breakup Conversation.* First, the overall interaction is decomposed into stages, then each stage into several specific "themes", such as pleading for mercy or reasoning about what happened. Each theme includes a number of parallel protocols that handle the different aspects of the theme, and a starter space that jump-starts the theme if necessary (for example, making some sobbing accusation jump-starts self-pity).

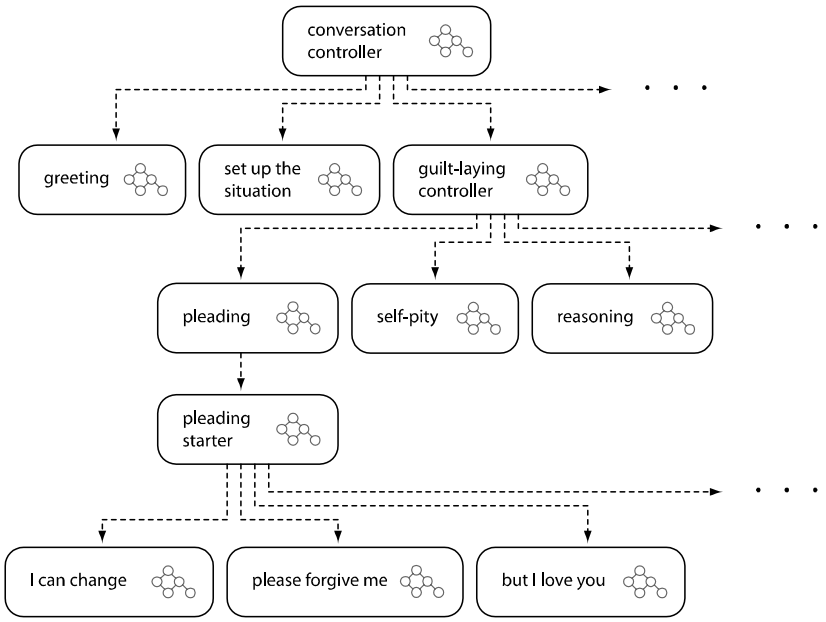


Figure 4-11. Fragment of space hierarchy from the Breakup Conversation.

Coupling can mimic synchronous or asynchronous control. In asynchronous control, the parent enables the child, and lets it run independently of itself. It can disable the child, of course, perhaps based on the activation of some other space, but this is not necessary. In Figure 4-12, the relationship between the *guilt-laying controller* and *pleading* space is an example of asynchronicity—the parent controls the child, but itself does not depend on the child's state.

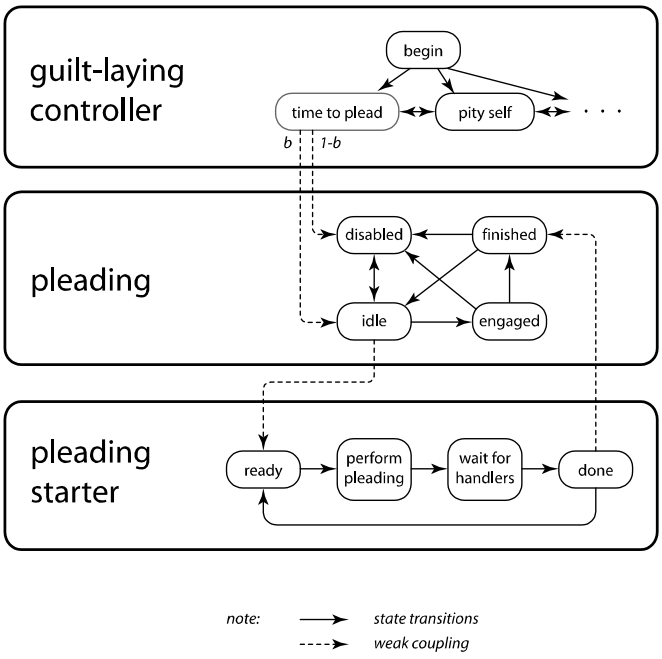


Figure 4-12. Coupling details among three different spaces.

Such interleaved interactions require the parent to include a controller state s_C that enables or disables the entire child space. Per previous description, the belief in state s_C becomes an activation signal for the child, and allows it to transition into the initial state s_1.

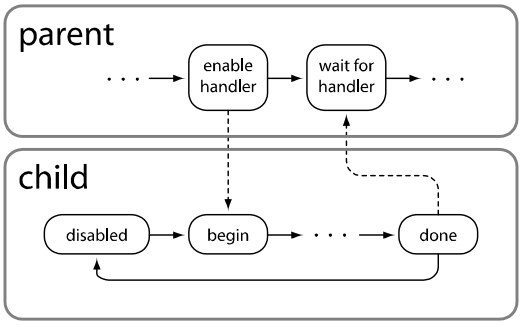


Figure 4-13. Nested dependency.

Synchronous coupling can be implemented by adding a reverse inspection, from the child to the parent. In addition to enabling the child, we add a special monitoring node to the parent which inspects some end-node of the child space; in this manner, the parent effectively 'blocks' until the child has reached some satisfactory end state. In Figure 4-12, the relationship between *pleading* and *pleading-starter* is an example of synchronicity.

The parent space includes a controller state s_C, with only one out-edge leading to a return-checking state s_R. Let the child space be enabled by the activation of s_C through inspection. Reciprocally, the child space includes some end state s_E that is being inspected by s_R. Activation of s_C enables the child, as well as transfers parent activation

to s_R; the parent will "wait" at s_R until the child reaches its end state s_E. See Figure 4-13 for illustration.

4.2.2. Techniques for Fallback

To err is human, and to recover even more so. Interactive systems must be able to deal with inevitable communication problems in a believably human-like way, resolving them without breaking the player's immersion in the interaction. One ought to avoid the mechanical answer of "I don't understand", leaving the human to do all the work. In actual interactions, we do not simply give in; rather, we use the knowledge and expectations about the situation to resolve miscommunication.

Players will inevitably attempt sub-interactions for which the system is not prepared—but those nevertheless need to be resolved amicably. People do not give up on misunderstood conversational moves, and neither should our agent.

Suppose that Xenos hears an odd utterance that it cannot possibly understand: "What's the population of United States", or "I hate it when the slithy toves gyre and gimble in the wabe". The shopkeeper agent knows about the United States about as much as about slithy toves—which is to say, nothing at all. But it would not be very sporting to just admit failure, or worse yet, ignore the utterance altogether.

Instead, the agent ought to attempt recognize as much as it can about what goes on in the interaction. Even if the slithy toves are not in the lexicon, one can detect from the sentence surface that it's an evaluation: the negative beginning "I hate" followed by the clause is a dead give-away. Thus it should be recognized as a negative evaluation of something unknown, and dealt with as such—replying, "I don't know enough to judge, really", or perhaps "what have they done to you?"

If the player is even more obscure, for example using an IM neologism such as "I h8 slithy toves", the evaluation might not get recognized either. But at least the "I" in the agent role that can be used, and the overall utterance treated as some kind of an information statement. This in turn can trigger a segue that retains a little bit of the subject continuity—e.g., "By the way, I think you might like this sword I just got."

This is implemented naturally using hierarchical parallel approach. Each type of utterance is represented by several different spaces, modeling different levels of generality. Their action production is arranged so that, if several of them suggest activity, productions from the more specialized handlers win over the more general ones.

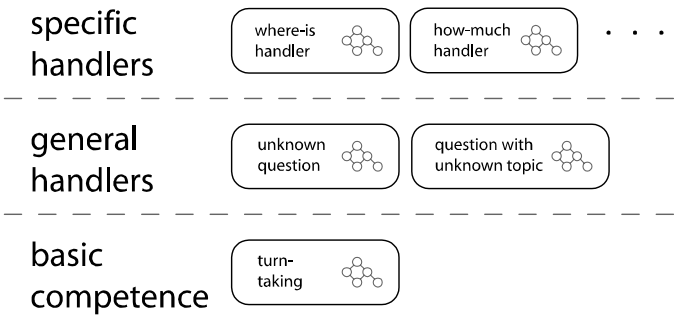


Figure 4-14. Fallback example with redundant question handlers.

For example, question answering happens on multiple levels. The most specific handlers look for particular questions such as “how much is” or “where is”, and deal with them appropriately. Less specifically, a generic question handler deals with situation where the topic was recognized, but not the question type—and it asks for clarification, such as: “what about the job?” Finally, at the least specific level, if the meaning of the question was completely misunderstood, the system must realize that something was being asked, and respond uncertainly—perhaps shrug or try to change the topic. Figure 4-14 illustrates a sample layout, with multiple redundant question-handling spaces.

4.2.3. Techniques for Recovery

Failure to advance in the interaction is another common possibility. Artificial systems often respond to failure by resetting the stage and trying again, but a believable agent must be careful about doing this.

Suppose that the Breakup Conversation had been going on for a while, but the pleading cycle reached an impasse—the player does not appear to be navigating through it. How to deal with the situation? It would be unacceptable to just restart the pleading from the beginning, or even worse, to keep repeating the last phrase hoping to hear something different. The situation has to be handled as it stands, ambiguous, and misunderstood.

Instead, the system must realize that it is not advancing. This requires an intermediate monitor space, including a special *stuck* state, which inspects the different spaces in the pleading cycle. As long as the pleading spaces are running, the belief in the stuck state keeps increasing a little with every time slice. At the same time, a *reset* state is set up to monitor when the stuck state crosses some acceptable threshold, at which point another cycle is enabled and ordered to take over.

It may even be beneficial to build an estimator of entropy, which would be sensitive to the entropy of the belief distribution for a given space. The useful extension has been

implemented in the system, but is not being used; I found it sufficient to monitor a success state, without concern over the shape of the overall belief distribution.

4.2.4. Annotated Conversation Excerpts

The following are examples of hierarchical and parallel engagement.

The diagrams are presented as timelines. On the right is a verbatim copy of a conversation with the system; it includes each utterance of the player and the agent, one after another. Above the diagram, some of the spaces responsible for the behavior are illustrated, and in the center, vertical lines correspond to their engagement in the situation; thick lines mean active engagement, thin dotted lines mean the space is active but not advancing, and no line means the space is inactive. On the left side, I include my own commentary on what happens in the system.

First, a simple illustration of interleaving. Figure 4-15 presents an example of the simultaneous overlapping activation of several different themes from the Breakup Conversation: pleading, self-pity, and guilt accusation. They are all active at the same time, and the player's actions as well as successes and failures of the other handlers, result in their reactivation or deactivation.

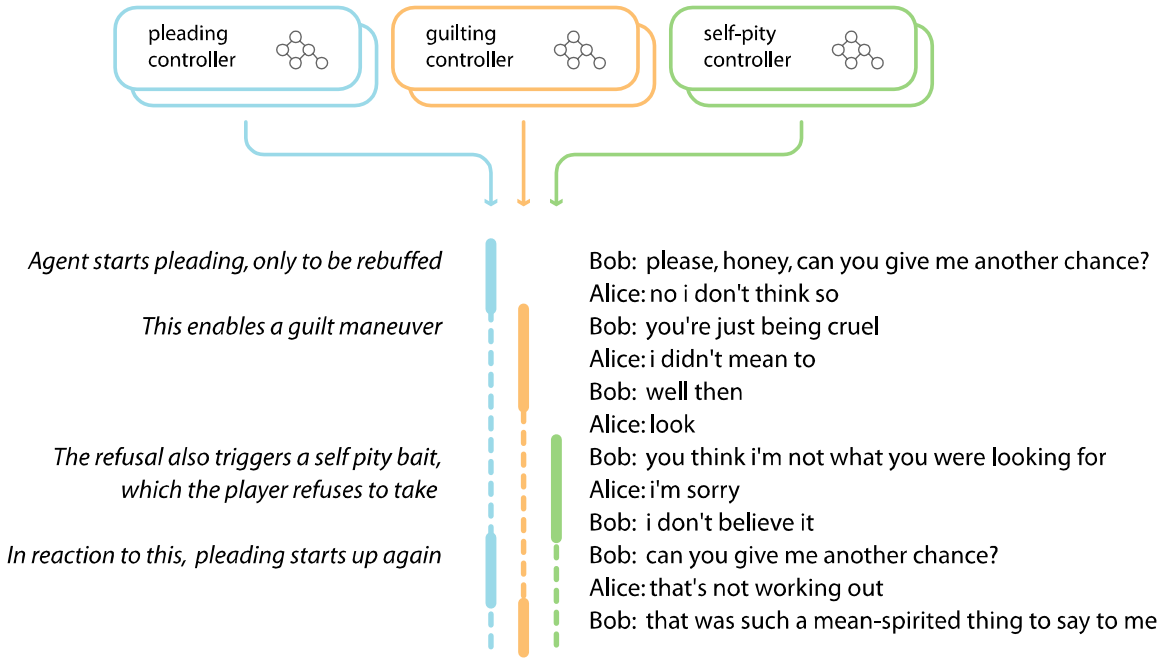


Figure 4-15. Several interleaving protocols.

Interleaving can also occur on a much longer time scale. In Figure 4-16, a quest acquisition protocol is started along with an item purchase conversation; however, the player does not pick up on it, so quest introduction remains activated but does not advance. However, the player's utterance after the purchase is recognized in context of quest acquisition, and the space handles it.

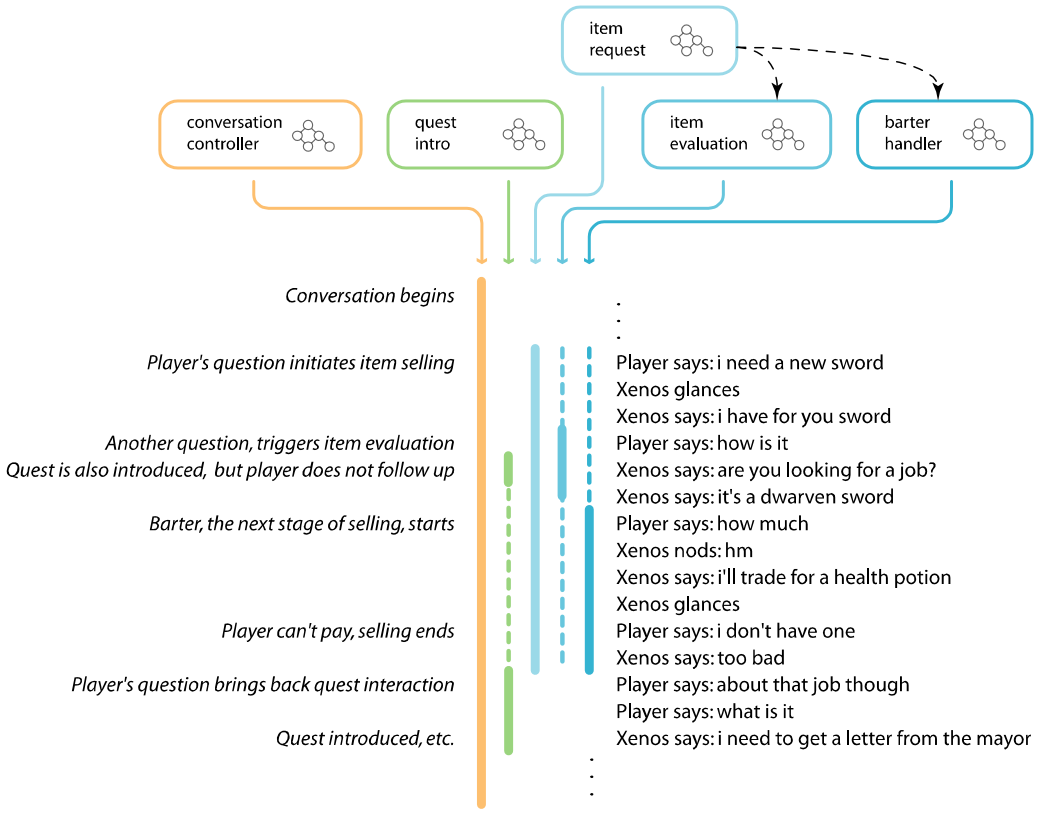


Figure 4-16. Interleaving on a longer time scale.

Interleaving can also implement robust error handling. As seen in Figure 4-17, one can arrange several categories of handlers, of different levels of specialization, to be all coactive but dormant most of the time. When a question or another utterance arrives which requires handling, all of the matching handlers activate—but only the most specific one actually results in output generation.

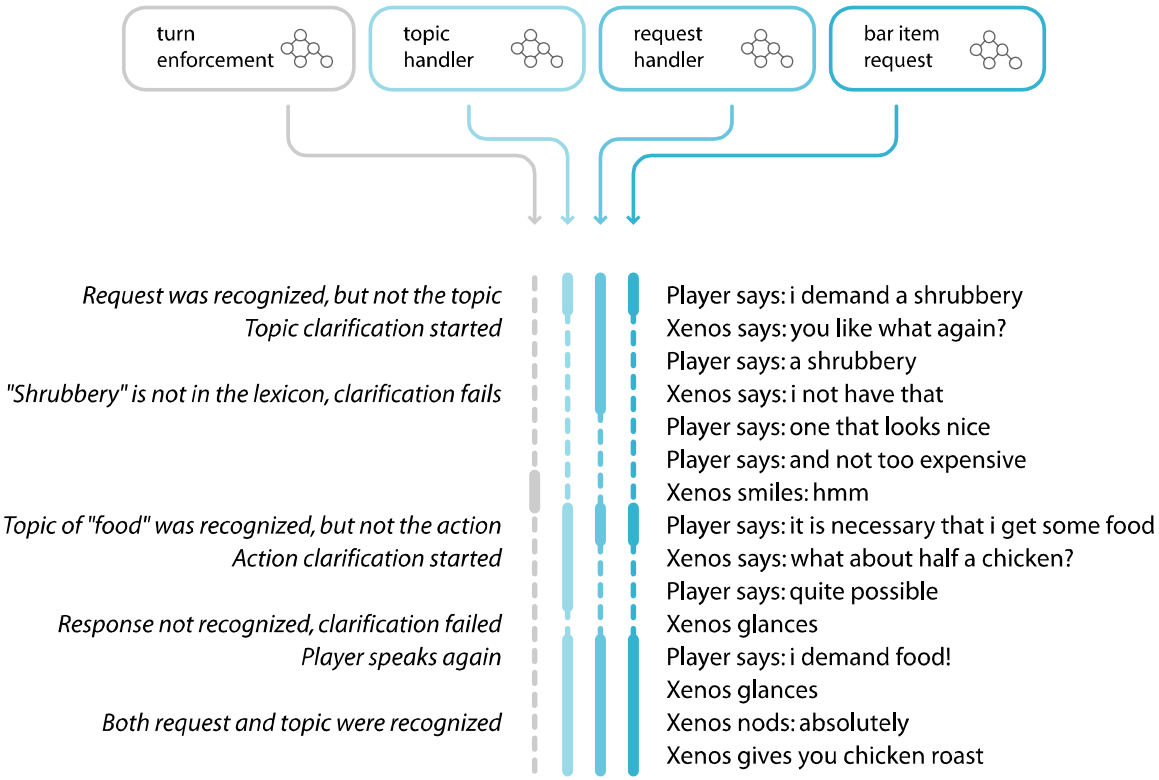


Figure 4-17. Fallback to general handlers.

Next example, Figure 4-18, shows how the situation can affect evidence processing. Statements such as “thanks” or a “what is” question are usually handled by general question-answering or request-response machines. However, certain situations imbue these standard utterances with new meaning—for example, “what’s up” is most

definitely not a question about what's on the ceiling, but a greeting routine—and specific handlers must be able to override them.

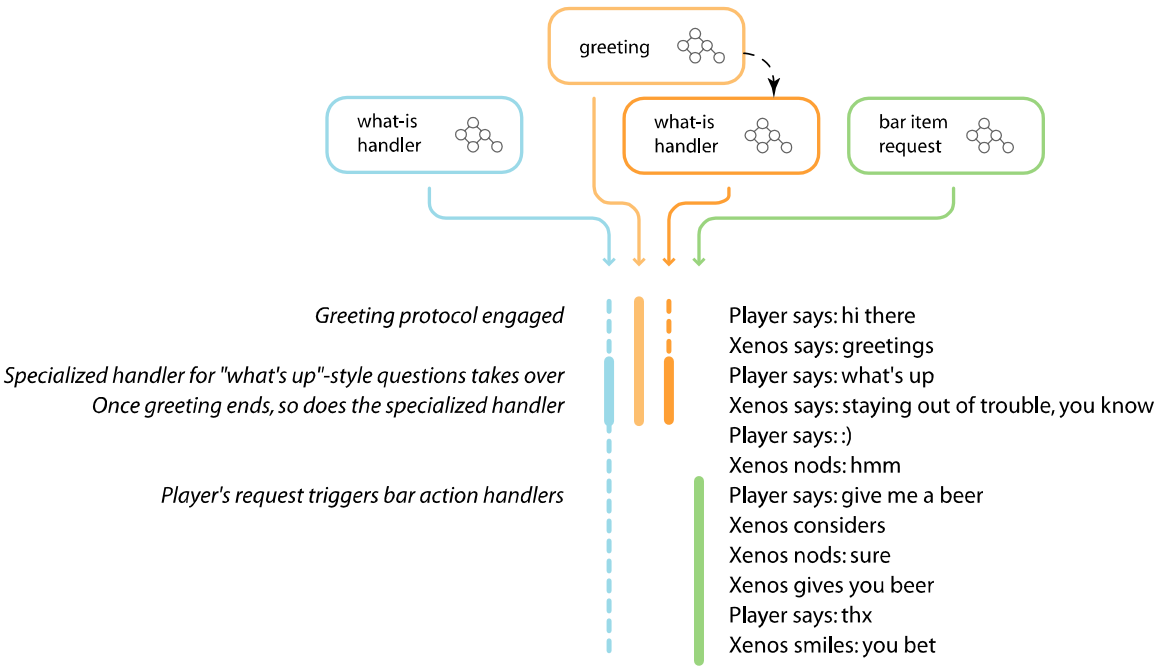


Figure 4-18. Situation-specific handling.

These specialized handlers are enabled and disabled by the parent space, but run in parallel with other handlers and override their outputs.

Figure 4-19 presents an example of a straightforward hierarchical engagement. As the guilt situation progresses, the main controller enables increasingly specialized spaces

to deal with the changing situation—they behave as if they were properly nested, and once the final space is finished, all of them exit.

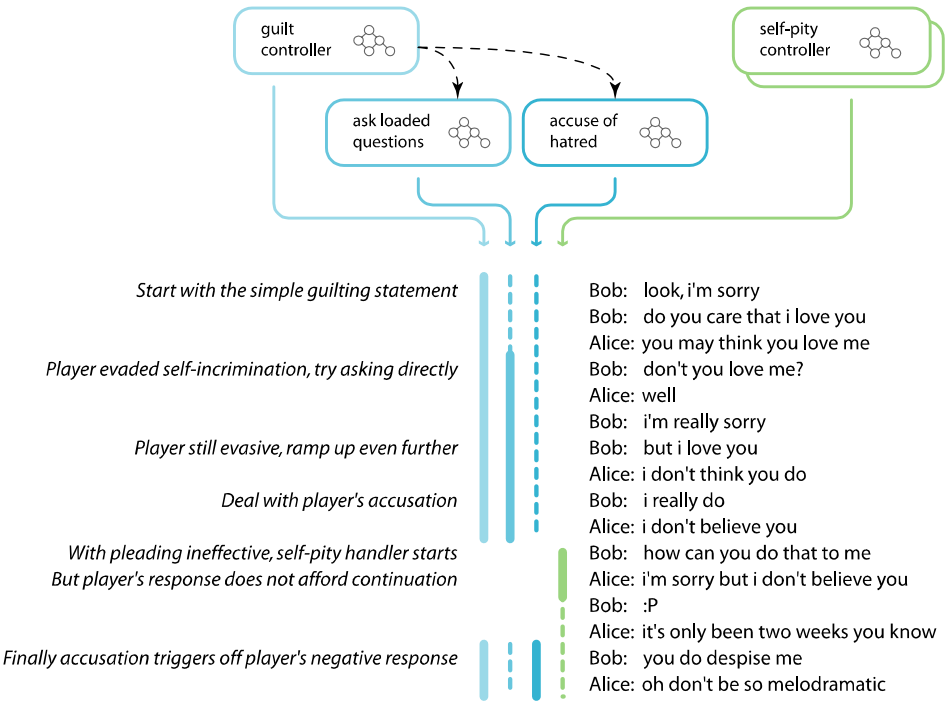


Figure 4-19. Hierarchical engagement.

Chapter 5. Architecture and Implementation.

C'est le temps que tu as perdu pour ta rose qui fait ta rose si importante.

It's the time you wasted on your rose that makes your rose so important.

— Antoine de Saint-Exupéry, *Le Petit Prince*.

Both *Xenos the Innkeeper* and the *Breakup Conversation*, the systems described in the previous chapters, were implemented using the same engine. In this chapter, I describe how to implement such an engine as a set of simple computation stages. The description below combines information on how to build such an engine from scratch, interleaved with background description of my particular implementation choices, and followed by an entire section on optimizations made possible by the system's architecture. I hope the detail will be sufficient to enable a complete and efficient re-implementation.

5.1. Engine Overview

The overall engine is a pipeline of several simple stages, invoked one after the other, each making use of the previous states' outputs. The following is a quick overview of the stages—we will examine them in great detail momentarily.

1. *Shallow parsing and preprocessing.* Inputs to the system arrive as a tuple of speaker's name, emote, and utterance; for example: ‹ player, says, "it's a good idea" ›. The utterance goes through very simple parsing, word tagging, and semantic markup. The result is a flat word list, tagged with information about each word.

2. *Evidence estimation.* Based on the flat word list, a number of estimators are run, to determine the value of evidence estimation $e(\gamma)$ for the different values of γ.

For example, “it’s a good idea” could be used to express confirmation, but it’s not likely to express a greeting or an insult; so the value of *e* (*confirmation*) should be pretty high, while the value of *e* (*insult*) and *e* (*greeting*) should be low or zero. The product of this stage is a distribution of probabilities over the set of possible communicative acts and other types of evidence.

3. *Concept tracking.* Some of the evidence estimators can then be used to adjust bindings of situational variables. For example, if the player had said, “what about the job”, that should be used to adjust the variable *the-topic* to point to a node describing a quest. The product of this stage is updated variable bindings, and updates to any estimators that use them as evidence.

4. *Belief computation.* Each belief distribution is now updated. The expectation and evidence functions are combined to form the state evidence function *q*, and used along with state transitions *p* and previous belief b_{t-1} to compute the new belief.

5. *Action selection.* Based on the new belief distribution, a number of actions are being proposed. These are arbitrated, and one action is finally produced by the system, resulting in a tuple ‹ agent, *emote*, *utterance* ›.

Finally, after the performance of these stages, the system’s output is sent back to the game, and also fed back to the system, becoming an input of the next iteration.

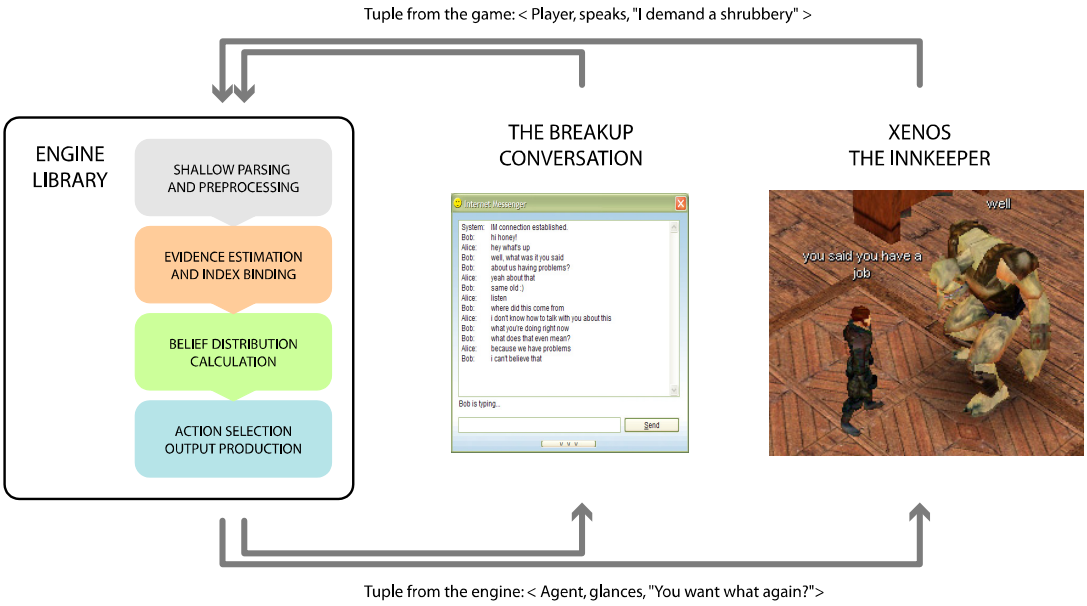


Figure 5-1. Engine interfaced to game clients.

As can be seen in Figure 5-1, the system can be easily located in a stand-alone library, which can then be used by client programs. *The Breakup Conversation* is a simple IM GUI that connects to the engine library. In *Xenos*, on the other hand, a custom interface library for the game *Neverwinter Nights* was written, which hooks up the library to a multiplayer game server.

Authoring a character using such an engine requires the designer to specify four different types of information. Most generally, some set of state spaces need to be

designed, which correspond to the different sub-interactions in which the agent could engage. Second, we need a set of estimators, which will convert user's utterances into evidence for where the conversation is going. Third, we need an action selection system, which will produce some output based on the state of the conversation. Additionally, we need a simple knowledge representation system, to keep track of topics of conversation across different sub-protocols.

Let us now turn to the details of how each stage works, and what it produces.

5.2. Engine Stages

5.2.1. Relaying Input From the Game To the Engine

The engine receives events produced by the player, as well as those it produces itself. Each event is a tuple of three elements: speaker's name, emote, and utterance. The name is self-explanatory; the utterance is a string containing what the player said (if anything); finally, the emote is a catch-all element for emotive actions or gestures (such as *"Rob waves: hi there"* or *"Rob frowns: I said no."*). This simple treatment allows for limited representation of embodied action, without a commitment to a particular ontology of activity.

When such a tuple is received, the name and action are stored verbatim, but the utterance is shallowly parsed as follows.

5.2.2. Parsing and Preprocessing

The preprocessing stage takes the utterance as a string, and converts it into a tagged list of words and stems. The tags should include: part of speech (POS) information, semantic roles, and a modicum of pragmatic information, such as the possible affective or social significance. Figure 5-2 presents an example of desirable preprocessing output.

Figure 5-2. Preprocessor output: word list with semantic roles and tags.

This is easy to do in two passes. First, the utterance is parsed using an off-the-shelf parser, to find POS tags and a parse tree. Then, second pass converts the parse tree into a flat list of stemmed words, marks semantic roles, and attaches appropriate additional tags based on dictionary lookup.

In my implementation I used the Link Parser, by Temperley, Sleator, and Lafferty (2005). It is a relational probabilistic parser, analyzing utterance structure by satisfying structural constraints imposed by each word. The parser is the only element of the system I didn't build; my modifications of it extend to marshalling output into a specific word list format, as described below. Fortunately, the parser source code can be downloaded from http://www.link.cs.cmu.edu and the license allows for unrestricted use in all manner of contexts.

Relational parsing works by applying constraints between types of words, and reconstructing the structure of the sentence based on a process roughly resembling constraint satisfaction. The result can be a linked graph of words, or a constituent tree. Along with structural reconstruction, this parser also returns POS tags for all of the words, performed heuristically based on word morphology and role in the link structure.

Figure 5-3. Parser output: part of speech tags and constituent tree.

After parsing, each word is annotated with its likely semantic role. These denote each word's role in the activity being considered: for example, which words describe the agent performing the action, the patient on which the action is performed, the method with which the action occurs, and so on. These sometimes overlap with grammatical categories such as subject, direct object, indirect object, and so on.

My system only supported four roles: *action, agent, patient,* and *method.* These are naively derived from constituent tree positions using rules shown in Table 5-1.

Semantic Role	Structural mapping	Constituent Tree Trace
+agent	Subject of main and subordinate clause	S → NP VP → S → NP SBAR → S → NP
+patient	Direct object of main and subordinate clause	VP → NP SBAR → VP → NP
+action	Verb	S → VP SBAR → S → VP
+method	Prepositional clause object (indirect)	VP → PP VP → SBAR → VP → PP

Table 5-1. Semantic role mapping

Where S is a sentence, NP a noun phrase, VP a verb phrase, PP a prepositional phrase, and SBAR a subordinate clause (after Temperley, Sleator, and Lafferty 2005).

We may also want other types of word information and categories, such as additional grammatical categories or some sort of pragmatic information, which are not provided by the parser. This is done by letting the user define additional categories.

For example, words may need to be categorized based on their possible emotional or social significance; “wonderful”, “lovely”, and “nice” would all be tagged as *pleasant* adjectives, “bad”, “horrible”, or “nasty” as *unpleasant*, and number of swear words as *insulting.*

Categories can also be used to specify useful high-level syntactic constructs, which are not provided by the parser. For example, it is often useful to group together “who”, “where”, “why”, and “when” as examples of interrogative *wh-words.* Other types of useful categories include: *indexicals,* such as “it”, “this”, and “those”, and the set of *copulae,* such as “am”, “are”, and “is”. These need not be disjoint, of course—“isn’t” belongs both to *copulae* and *negative* sets, for example.

The categories, in a sense, describe pragmatically important subsets over the set of all words. It is also useful to allow set operations on categories; for example, such that the tag *negative* could be described as a union of *insulting, unpleasant,* and others.

In my own implementation, the set of categories and their members are enumerated at design-time by the system developer; this allows for some useful optimizations (to be described in section 5.3). At the same time, this limits the system’s categorization

abilities to straightforward lookup. A more comprehensive implementation should make use of more complex relations between words, such as semantic relations expressed in databases such as WordNet (Miller 2005), or analysis of affective effect.

Finally, the preprocessor stems each word using the Porter stemmer (Porter 1980), and stores the stem alongside the word in the list. The stem will be used for matching by evidence estimators.

The result of this stage is a flat list of stemmed words, tagged with part of speech information, semantic roles, and other category information. The constituent tree is discarded.

5.2.3. Evidence Estimation

Evidence estimation stage takes the tagged word list, and computes the probability of having observed the different types of evidence given the utterance. The types of evidence include speech acts (for example, *insult, question,* or *denial*), communicative acts that can be accomplished through speech or emotes (for example, *greeting* or *negation*), and conversational idioms observed in the task (for example, *I-care-about-you* expression, or an *impatience-expression*).

The result of estimation is the *e* function distribution—a mapping from evidence types γ to probability values, given the word list, such as illustrated in Table 5-2. It is

normalized, either in this stage or during belief distribution update. The following is a description of how these evidence types can be defined and evaluated.

Utterance: “it’s a good idea”

Evidence type:	*greeting*	*compliment*	*agreement*	*insult*	. . .
Probability:	0.0	0.9	0.1	0.0	. . .

Table 5-2. Communicative act estimation example for an utterance.

Categorization of speech into communicative acts is a painfully complicated process, and the solution could be made highly complex. However, rather than attempt a strong and fail-proof solution, I found it quite sufficient to attempt a weak and fast solution: just a broad categorization based on the surface features of the utterance.

In a few simple cases, word lookup can be enough. For example, “hello” is very likely used as a greeting, and the verb “beg” used to describe a request. The mapping does not need to be one-to-one, of course—a word such as “yeah” can indicate a number of actions, including an agreement, an acknowledgement, a conversational maintenance

expression, or even just a generic way of taking a turn. So sometimes, word-based lookup is enough.

In most cases, however, one looks for sequences or combinations of several words or tags. Speech acts commonly have very distinct, idiomatic structure—for example, "can you *verb-phrase*" is generally used to denote a request. Similarly, the grammatical structure "do you *verb*" or "*wh-word copula*" are usually different types of questions. Finally, a number of colloquialisms and fixed expressions must be recognized as such—for example, "what's up", which is an idiomatic greeting and not a question.

As the system grew, I found it useful, from the authoring point of view, to also allow for the combination of individual simpler types of evidence into complex ones: for example, allowing the user to define *greeted-or-complimented* as a straightforward max of *greeting* and *compliment* estimators.

Testing for sequences of words or tags can be done using simple pattern-matching techniques. In my implementation, I developed a custom pattern language, similar to limited regular expressions, which could match based on word stems or combinations of tags. The language performed a match over the utterance word list, and returned a probability value. Such tests could also be combined into conjunctions and disjunctions, as necessary for the task at hand.

The result of a surface match is ambiguous and only roughly accurate, but it appears to be sufficient—the probabilistic belief computation stage helps with disambiguation, since it uses situational expectations to eliminate irrelevant or erroneous evidence.

In addition to surface estimation, two other types of evidence are supported. First, evidence from variables, will be described in the next section. Second, evidence from the last known belief distribution, is our implementation of the inspection mechanism, as described in section 3.3. Recall that this refers to looking up belief over a particular state of some particular HMM, such as "the *inactive* state of *self-pity-controller*" or "the *player-counters* state of *barter-for-thing*". It is implemented straightforwardly, as a value lookup of that state's last belief distribution, $b_{t-1}(s)$.

The result of all the evidence estimation stage is a distribution of probability values over the set of all possible evidence types.

5.2.4. Situational Variables and Topic Retention

Variable binding also results in evidence evaluation, but it uses a slightly different mechanism, which we should discuss first.

As discussed in section 3.2.2 on topic retention, the system ought to retain a modicum of topic information across numerous particular sub-interactions. This can be

implemented as a handful of variables, available to be bound to the different topics of conversation.

These variables are task-sensitive and agent-specific, and their binding can be re-evaluated on a regular basis, based on the task at hand. Each variable is controlled by several binding routines called *trackers,* which try to keep it bound to relevant values. It is convenient to have a handful of different variables with different tracking mechanisms, operating on different time scales. In my latest implementation, the situational variables include: *the-topic* corresponding to the topic of conversation, *the-current-topic* that represented a more transient version of the previous, *the-action, the-agent, the-patient,* which tracked which nodes might have been referred to in the different parts of the last utterance. Additionally, *Breakup Conversation* included *the-reason* which pointed to an event relevant to the breakup, and *Xenos* included *the-quest* variable for talking about some quest information, and *the-item* variable pointing out an item to be bartered.

The variables can be bound over a set of values, or concepts, representing the topics that can be discussed—these range from simple atomic elements, such as *player* or *sword,* to more complex concepts such as *letter-courier-quest.* The collection of all concepts is a minimal knowledge representation of the game world.

My implementation allows for the addition of properties attached to concepts, which are used in action production (described later in this chapter), and relations, which bind together different concepts. The result is a simple semantic network describing an ontology of game entities, events that can be talked about, and relationships between them.

Variable state can now be used as types of evidence. Some types of evidence my implementation supports include: whether a variable is bound, whether it is bound to a node of particular type (for example, if *the-topic* is bound to a *quest* node), and whether it is bound to a node with a particular relation (for example, if *the-topic* has an agent relation to the *player* node). These tests are performed, as specified by the designer, and used to fill in the appropriate evidence estimators.

5.2.5. Belief Update

After evidence estimation, the belief for each HMM is updated. Recall that the belief distribution for any particular state can be calculated using formula (1b) from section 3.1:

$$b_t(s) = \sum_{s_i \in S} p(s \mid s_i)\, q(\gamma \mid s)\, b_{t-1}(s_i)\, \eta_{s,t}$$

In terms of implementation, each HMM can be thought of as a structure that includes the following information:

- The previous estimated belief distribution b_{t-1} (mutable),
- The transition probability p over pairs of states (constant), and
- The evidence expectation g over pairs of states and evidence types (constant).

The belief update computation is quite straightforward. A naïve implementation could look as follows. For each target state s, we iterate over all possible source states s_i, summing up the products of each previous belief in the source state $b_{t-1}(s_i)$ and the probability of having transitioned $p(s \mid s_i)$. At the same time, we sum up the products of the different evidence expectations $g(\gamma \mid s)$ with the evidence estimations $e(\gamma)$ over the set of evidence types. Multiplying the resulting two numbers together, we get the raw belief estimate for the state—conceptually, it is a product of the likelihood of the new state given where we thought we used to be, and given what we thought we observed. After computing this for all states, normalize the entire distribution such that is sums up to one, to control for loss of precision—the normalized distribution is the updated belief value.

This naïve implementation gets the point across, but would be exceedingly inefficient. The optimization section includes more details on how belief over an entire state space can be accomplished in two matrix operations and a normalization.

The result of this stage are new belief distributions for every HMM in the system.

5.2.6. Action Production

Finally, based on the belief distribution, an action production layer decides what utterance to produce or what action to perform in the world, and sends that output back to the game.

The set of actions to be produced includes: a natural language utterance, a description of an action (such as an emote: wave, smile, etc.), or a modification to a variable or a model belief distribution; the latter implements coupling via self-modification.

To know what action to produce given the total state of the system, the system needs some action policy, which maps from the state of the system, to action performance in the world. As mentioned earlier, many such policies are possible, and finding an optimal one can be highly problematic.

In my implementation, instead of an optimal policy, I make use of a satisficing approach instead—a policy that produces "good enough" actions given the state of the system, instead of spending the resources on finding the best one (Nourbakhsh and Genesereth 1996).

To simplify implementation, this approach is architecturally decomposed into three parts: evaluating local policies, arbitrating among them, and actually generating the output for the game based on the chosen action.

Local policies are local to each space: they map from belief distribution to action. In my implementation, they are accomplished with lookup tables based on a *winner-takes-all* evaluation—each state can be annotated with at most one action, and the policy produces the action of whichever state had the largest belief confidence value. While non-optimal, such policies are satisfactory in producing good actions for all situations.

Figure 5-4 shows the policies diagrammatically, marked with the letter π. In my implementation, they produce tuples ‹ *a, b* ›, which include actions, and the belief level of the state responsible for that action.

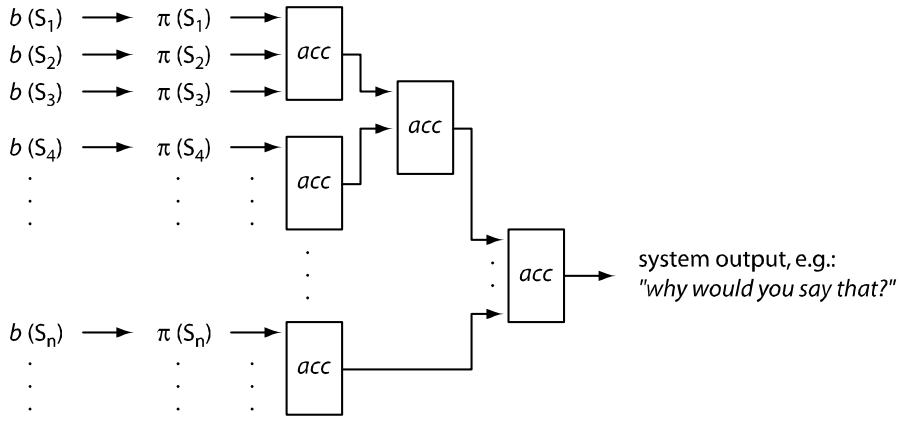


Figure 5-4. Sample action arbitration network.

Just like state spaces, local policies are all evaluated at the same time, and will provide competing evaluations of what action should be produced. It will be necessary to arbitrate among them.

Arbitration is performed by a collection of "accumulators", which fold some binary choice operator over a set of inputs: each takes a number of action-belief tuples as inputs, and selects one of them as output. By hooking them up into a circuit-like directed acyclic graph with a single output—with multiple sources and one sink, to borrow circuit terminology—we can build a structure for producing a single action out of the many inputs.

For the individual accumulators, the selection mechanisms I used in my implementation include: the *first* function, which selects the first in the input list with non-zero belief value, similarly to a fixed priority policy of subsumption (Brooks 1986); the *max* function, which selects the input with the maximal belief value, similar in spirit to voting approaches (Rosenblatt 1995); or the *all* function, which actually creates a new output whose action is a concatenation of the input actions, and whose belief value is their max.

Using such functions, the designer can build an arbitrary arbitration network. As for its topology, I found it is useful for the network to recapitulate the shape of the space hierarchy, selecting more specialized outputs before the generalized ones. In

particular, all of the behaviors on the same level of specificity should be grouped together—for example, all of the turn-taking, simple reactions and ambient movement in one accumulator, general question-answering in another, then specific question-answering in an accumulator different still. Those accumulators should then be arbitrated via the *first* function, which selects the most specific action (if available) before the less specific one; this helps improve internal redundancy and system robustness. However, this is certainly not enforced by this specification.

Finally, one action is produced by the network, and an output generator inspects the suggested action, and attempts to produce it. In the case of self-modification, the action is performed right away. In the other cases, strings for the utterance and the emote are generated and returned to the game.

Text for both the utterance and the emote is generated using simple template matching. The action can specify template chunks, on the level ranging from single words, through small structures, up to phrases and other arbitrarily large sequences, and means of combining them. Templates can also access nodes referenced by the situational variables, read their properties (such as name or description), and insert those into the output being produced. In addition, all template elements can be randomized, to allow for variation, and prevent the repetition of similar text blurbs.

The result of this stage is an output of a tuple of three elements: agent's name, emote, and utterance, and an optional self-modification of the system's internal state.

5.3. Pre-compilation and Optimizations

The simple pipelined nature of the system lends itself to numerous optimizations. This section describes some of the more interesting optimizations I used in my own implementation.

5.3.1. Pre-compilation

In order to achieve many of the optimizations listed below, I built a separate pre-compilation step for the system creation. The pre-compiler would take high-level descriptions of estimators, state spaces, action policies, and so on, perform optimization on them, convert them into simpler data structures, created custom code as necessary to populate and maintain them appropriately—and output all of this information as a set of C++ files which got compiled along with the static part of the engine.

The pre-compilation step afforded additional optimization steps, and resulted in extremely efficient code. On the down side, however, the additional compilation effectively precluded the possibility of dynamic data binding and runtime modifications to the engine.

5.3.2. Parsing and Preprocessing

In the preprocessor, a number of components can be optimized. First, observe that the set of tags is known at design time. Therefore we can implement tagging—and tag matching—as bitmask operations. If we map each tag into a bitmask position, we can perform tag tests as bitwise mask operations. Testing if a word matches a disjunction of tags is a matter of bitwise AND with an appropriate mask, and testing for a non-zero result; matching a conjunction is a matter of bitwise AND followed by testing for equality with the mask.

5.3.3. Evidence Estimation

Two elements of evidence estimation can be easily optimized: its evaluation and lookup.

As mentioned before, I used a limited pattern-matching language to evaluate evidence probabilities. The language itself was optimized to make use of bitmask operations, and to remove the need for backtracking. As a more limited language, it did not have the power of more robust pattern matchers, but most of the operations translated into a single pass of bitmask tests over the word list. And more complex types of evidence could be assembled out of such smaller pieces.

During evaluation, it was also beneficial to support a kind of common sub-expression elimination—since complex evidence usually was made of similar simpler elements, only differently arranged (for example, both *request* and *question* can refer to the same "could you *verb*" pattern). It helps to pull those out, such that they be evaluated once, and the value memoized for later reuse. The dependencies are known at design time, so the optimization can happen before the system runs.

The other easy optimization is speeding up evidence lookup. In a naïve implementation, evidence could be implemented as a hash table or some other generalized map. Since the set of all evidence types is known at design time, however, we can represent the *e* function as a simple array, and convert all name-based lookups into array references whose index is known at compile time. This turns evidence lookup into a constant-time operation.

5.3.4. Concept Representation

I found it convenient to assume that concepts for the situational variables will be immutable at runtime—that is, their property and relation signatures can be assumed not to change.

This made them easy to translate directly into object-oriented C++ class definitions. Properties translated directly into member variables, and relations into pointers to

other nodes. Once again, this simplified storage, but at the cost of making the system more rigid, since C++ does not support dynamic class redefinitions.

5.3.5. Belief Update

Belief computation can be performed as a sequence of straightforward matrix multiplications. We first optimize the computation of state evidence q, and then the entire belief update.

Suppose we have |S| states and |A| evidence types. As mentioned before, both sets are completely specified at design time, including the names of all of their members, such that the results of the estimation function e can be stored in a simple array. The evidence expectation function g is immutable, and can be stored as a matrix of size |A| × |S|. Their multiplication computes the evidence function q.

In my implementation, however, I noticed that the g matrix was sparse, because each state expected at most one evidence type. Therefore it was possible to do partial evaluation of the sparse matrix multiplication: represent g as a simple vector of size |S|, along with a piece of precompiled C++ code that, during update, looked up the appropriate cells of e and g arrays, and stored their product in the q array. Instead of the special compilation code, one could also use one of the popularly available linear algebra libraries, which often include optimized sparse matrix multiplication routines.

Once we have q, we can compute new belief values using a sequence of simple operations. Both b_{t-1} and q can be stored in one-dimensional arrays of size |S|, and p in a matrix of size |S| × |S|. The update over the entire space then becomes a matter of a matrix multiplication, an element-wise vector multiplication, and a normalization. (Notice that in our case, normalization is just a piecewise division by the *sum* of the components—not by a sum of the *squares* of the components. We're decidedly not in a Euclidean space.)

As a sequence of operations, this translates to:

$$b_t^* = (b_{t-1} \times p) \cdot q$$

$$\eta = 1 / \sum_{s \in S} b_t^*(s)$$

$$b_t = \eta \cdot b_t^*$$

The resulting b_t can then be stored for reuse in the next iteration.

5.3.6. Action Production

Finally, action production is optimized in a straightforward manner, by delaying the actual production until when it is absolutely necessary. Each space policy suggests an action-belief pair, but only the belief element is necessary for arbitration—therefore the 'action' component can remain unevaluated until all arbitration has finished, and only one, final action remains. The action is evaluated at the very end—because

evaluation can be expensive, including numerous string operations, lookups, and so on—finally producing an output that is sent back to the game.

Chapter 6. Conclusions and Future Work.

Och, nie mam wątpliwości, że to premiera.

I cokolwiek uczynię,

zamieni się na zawsze w to, co uczyniłam.

Oh, I have no doubt that this is the opening night.

And whatever I do,

will be forever changed into what I have done.

— Wisława Szymborska, *Życie na poczekaniu*

6.1. Looking Back

In this report, I outlined an approach to participation in social interaction for the purpose of implementing engaging interactive entertainment characters. We can now recapitulate some of its most salient aspects.

6.1.1. Report Summary

The problem of social interaction is that of interacting with an unpredictable player in a desirably entertaining manner. Techniques used for this approach, however, must fit within an intersection of constraints peculiar to the production of entertainment artifacts. First, the behavior of interactive characters must be easy to author and tweak; it must be easy for the designer to not only create a representation that exhibits desirable runtime behavior, but also to predict the runtime behavior given the representation at hand. Second, the behavior must be believable; characters must behave like reasonable living creatures, acting attentively and intelligently given the fiction of the virtual world, in concert with—and in spite of—what the unreliable and unpredictable player is doing. Third, the behavior must be computationally inexpensive; agents must be responsive to the activity around them, but the developer must accomplish this within a fraction of the processing power available on commodity gaming platforms.

A number of finite-state techniques are currently being used to accomplish some social interaction within the above constraints. Unfortunately, while optimized for efficiency and ease of authoring, the approaches tend to be limited in their support for believable behavior. In particular, they tend to provide little or no support for robust decomposition of complex activity, one that allows for both hierarchical causal dependence and parallel temporal independence between the components. This I introduced as the *hierarchical-parallel problem* of interaction.

To resolve this problem, I extended existing finite-state techniques to support both hierarchy and parallelism requirements of interaction. I pursue this by adopting the hidden Markov models to represent the process of social interaction, and introduce *coupling* extensions to HMMs, which allow for easy and efficient representation of hierarchy within a collection of parallel state spaces.

The resulting system improves on the existing finite-state techniques, popular in game production, by enhancing believability: it presents techniques for robust situation tracking, recovery from error, and gentle performance degradation. At the same time, it fits the constraints of allowing for tight authorial control over the form and shape of the interaction, as well as an efficient implementation.

6.1.2. System Benefits

Two systems were built using this approach, and they demonstrate the benefits of the technique in distinct ways. First, *Xenos the Innkeeper,* is a fairly typical barkeeper character for a fantasy role-playing game, which introduces player quests and sells items. The second demo, *The Breakup Conversation,* is a "sim game" that parodies the breakup of a relationship, and is played entirely over instant messenger.

The systems demonstrate believable and effective performance, and I analyze some of the particular effects in greater detail, to show the benefits of the hierarchical coupling of parallel models. The decomposition results in a very efficient implementation. Not only are the components asymptotically faster than their composite, but they enable a very simple and efficient implementation. Details of how to accomplish this were presented, and the computational benefits demonstrated.

The two implementations share a large number of general models, in spite of being used for very different agents in different games. This demonstrates the authorial benefits of the approach, as the component hierarchies can be general, and reused across implementations. The general models can then be supplemented with specialized handlers for the specific characters, increasing system robustness.

Thanks to the component hierarchies, and redundant models on multiple levels of generality, the systems behave very robustly in the face of the player's noisy and complex natural language input.

6.1.3. System Limitations

I would also like to say a few words about the failure modes of these systems. In both cases, the two most important problems are topic retention, and keeping the conversation within the system's area of competence.

Topic retention difficulty stems from my approach to topic tracking, using global variables that are bound based on simple input triggers and system state. Unfortunately, the binding was not sufficiently robust, and the system would sometimes get distracted by accidental topic mentions that did not actually merit a switch. In a future implementation, a more robust approach to topic tracking should be used instead.

The second problem was keeping the conversation within the system's area of competence. With *Xenos,* this was not too much of a problem—the fantasy barkeeper scenario circumscribed the communicative possibilities.

The Breakup Conversation, on the other hand, was much more problematic, because given the context of a breakup, the possible space of "things to talk about that made sense

within the situation" was huge. The system got easily confused when the player started inventing past relationship memories or new reasons for the breakup. My solution was to build the character so that it controlled the conversation, and routinely forced it back to familiar territory. Unfortunately, this also gave the interactions a neurotic and domineering feel, because the system did not seem to pay attention to the player's contributions to the background story.

The *Breakup* situation is an example of the "worst-case" scenario for this kind of a technology, because the player felt obliged to contribute to the fiction of the breakup, but the system was not capable of dealing with that. Player's inventiveness is a common failure mode for free-form language systems—for example, both Façade (Mateas and Stern 2004) and the numerous ALICE bots (Wallace 2000) experience similar difficulties.

One could attempt to control player's inventiveness by limiting the reasons for invention (for example, by specifying a detailed back-story between the characters, so that that the player and the system knew everything about the situation). Given a strong enough reasoning system, one could also attempt to deal with the inventions, by adding a more complex episodic memory that could handle the new information.

In the end, systems such as this one work best when the context is familiar, stereotyped, or otherwise explicated, and where innovations are not expected, or can be reasonably handled.

6.2. Looking Forward

Many new solutions raise more questions than they answer. It seems this work is no exception. In particular, I would like to concentrate on some directions that seem important to develop further.

6.2.1. Learning and Model Acquisition

As the scope of the world increases, content creation becomes a dominant problem—manual design for all of the content, such as agent behavior, becomes quite expensive. Instead, it would be highly beneficial to have a method by which their parameters—or indeed, complete model descriptions—could be acquired automatically. The development of some learning approach, by which a description of desirable behavior could be constructed automatically, perhaps through observation of human activity, will be of utmost importance.

Unfortunately, the approach presented here introduces two unusual difficulties.

First is the ever-present authoring problem. Model descriptions have to be available for deliberate modification and debugging, such that they produce desirable runtime

behavior. This means that the developer must have clear understanding of the semantics of the model description, and the particular learning technique will have to make that available. It is not enough to simply acquire model parameters from training data; their meaning has to be accessible as well.

Second, the characteristics of the system, in particular coupling and the use of situational variables, makes learning difficult. It is not clear how automatic model acquisition should work for a collection of coupled HMMs such as this one, especially given the particular coupling technique presented here; and as I mentioned before, learning in presence of variables is already known to be quite difficult. Thus, the first step would be to ignore such difficulties, and concentrate on learning each individual model separately; only then can the issues of coupling and tracking be properly addressed.

6.2.2. Ontology of Basic-Level Representations

The modular decomposition into smaller interaction sub-protocols encourages representation reuse. It seems quite possible to develop a set of standard, general sub-interaction models, which could be shared across a class of individual agents. After all, unspecific activities such as answering questions, dealing with requests, or getting in and out of conversation, should generalize across different individuals. Such general

descriptions could then be augmented with character-specific interactions, used to make particular agents interesting and unique.

It would be quite useful to develop an ontology of basic-level interaction models—basic-level in the sense that they are not tied to particular individual's details, but at the same time not so abstract as to be useless. In this work, I attempted such an ontology, but the results were not sufficiently complete to warrant detailed explanation. However, it seems that it should be a very beneficial next step.

6.2.3. Stochastic Modeling of General Activity

This work concentrated on the treatment of social interaction as a stochastic process, and its subsequent modeling using HMMs. However, communication and social interaction are not the only phenomena that can be usefully recast as stochastic processes.

Interactive entertainment is, almost by definition, focused on the player's interaction with the game world, and on the design of desirable runtime behaviors to support this interaction. This invites the decomposition of all player activity as a stochastic process, possibly decomposable into a collection of several related sub-processes.

Many genres explicitly support this. In first person shooters, the mechanics are quite simple, which allows designers to model player behavior as a mixture of a few behavior

modes—level exploration, combat, searching for ammo, etc. And even in more complicated genres, such as real-time strategy games, the player can be seen as engaging in a number of different activities, from exploring the map, gathering resources, building up military or production infrastructure, to creating defensive and offensive units, securing beneficial terrain, attacking enemy resources, and so on.

These kinds of engagement can be modeled as processes, and used to track the player's progression through the game, and possibly predict what they might do next. Recasting this behavior as a stochastic process should prove straightforward and inexpensive.

Chapter 7. References

Agre, P. E. 1995. Computational research on interaction and agency. *Artificial Intelligence*, 72. Amsterdam : Elsevier.

Agre, P. E., Chapman, D. 1987. Pengi: An implementation of a theory of activity. *Proceedings of AAAI-87,* pp. 268-272. San Jose, CA: AAAI Press.

Agre, P. E., Horswill, I. D. 1992. Cultural support for improvisation. *Proceedings of AAAI-92.* San Jose, CA: AAAI Press.

Allen, J. F. 1993. Natural language, knowledge representation, and logical form. In Bates, M., Weischedel, R. M., *Challenges in natural language processing.* Cambridge: Cambridge University Press.

Allen, J., Byron, D., Dzikovska, M., Ferguson, G., Galescu, L., Stent, A. 2001. Towards Conversational Human-Computer Interaction. *AI Magazine,* 22 (4), pp. 27-38.

Allen, J. F., Miller, B. W., Ringger, E. K., and Sikorski, T. 1996. A robust system for natural spoken dialogue. *Proceedings of the 34th Annual Meeting of the ACL,* pp. 62-70.

Allen, J. F., Perrault, C. R. 1980. Analyzing Intention in Utterances. *Artificial Intelligence,* 15: 143-178. Reprinted in Grosz, et al. 1987.

Alshawi, H. 1987. *Memory and context for language interpretation.* Cambridge: Cambridge University Press.

Appelt, D. E. 1985. *Planning Natural Language Utterances.* Cambridge: Cambridge University Press.

Austin, J. L. 1962. *How To Do Things With Words.* Cambridge, MA: Harvard University Press.

Bates, J., Loyall, B., Reilly, W. S. 1991. Broad Agents. In *Proceedings of AAAI Spring Symposium on Integrated Intelligent Architectures.* Menlo Park, CA: AAAI Press.

Berne, E. 1964. *Games People Play: The Psychology of Human Relationships.* New York: Grove Press.

Bioware. 2002. *Neverwinter Nights.* Computer game. New York: Atari.

Birnbaum, L., Selfridge, M. 1981. *Conceptual Analysis of Natural Language.* In Schank and Riesbeck, 1981.

Bonneau-Maynard, H., Rosset, S. 2003. A Semantic representation for spoken dialogs. *ISCA Eurospeech '03,* Geneva.

Bouilier, C., Dean, T., Hanks, S. 1999. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research* 11, pp. 1-94.

Brand, M., Oliver, N., Pentland, S. 1997. Coupled hidden Markov models for complex action recognition. *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition (CVPR97).*

Brand, M. 1997. Coupled hidden Markov models for modeling interacting processes. *Learning and Common Sense Technical Report 405,* Media Lab. Cambridge, MA: Massachusetts Institute of Technology.

Brooks, R. A. 1986. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation,* 2 (1), pp. 14-23.

Cassell, J. 2000. Nudge Nudge Wink Wink: Elements of Face-to-Face Conversation for Embodied Conversational Agents. In Cassell, Sullivan, et al., 2000.

Cassell, J., Bickmore, T., Campbell, L., Vilhjalmsson, H., Yan, H. 2000. Human Conversation as a System Framework: Designing Embodied Conversational Agents. In Cassell, Sullivan, et al., 2000.

Cassell, J., Sullivan, J., Prevost, S., Churchill, E. 2000. *Embodied Conversational Agents.* Cambridge, MA: MIT Press.

Cohen, P. R. 1995. Dialogue Modeling. In Cole, et al. 1995, pp. 234-240.

Cohen, P. R., Morgan, J., Pollack, M. E. 1990. *Intentions in Communication.* Cambridge, MA: MIT Press.

Cohen, P. R., Levesque, H. J. 1990. Rational Interaction as Basis for Communication. In Cohen, Morgan, and Pollack, 1990.

Cohen, P. R., Perrault, C. R. 1979. Elements of a Plan-Based Theory of Speech Acts. *Cognitive Science* 3(3), pp. 177-212. Reprinted in Grosz et al., 1986.

Colby, K. M. 1973. Simulations of Belief Systems. In Schank, R. C., Colby, K. M., eds. *Computer Models of Thought and Language.* San Francisco: W. H. Freeman.

Cole, R. A., Mariani, J., Uszkoreit, H., Zaenen, A., Zue, V. (eds.) 1995. *Survey of the State of the Art in Human Language Technology.* National Science Foundation, et al. Reprinted by Cambridge University Press, Cambridge, 1996.

Cullingford, R. 1981. *SAM.* In Schank and Riesbeck, 1981.

Cycorp, Inc. 2005. OpenCyc 0.9. Downloadable software, available at http://www.opencyc.org/ (online resource, last accessed March 2005).

Fikes, R. E., Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence,* 2, pp. 189-208.

Fine, S., Singer, Y., Tishby, N. 1998. The Hierarchical Hidden Markov Model: Analysis and Applications. *Machine Learning,* 32: 41.

Finney, S., Gardiol, N. H., Kaelbling, L. P., Oates, T. 2002. The Thing That We Tried Didn't Work Very Well: Deictic Representation in Reinforcement Learning. *ICML-2002 Workshop on Development of Representations.*

Garfinkel, H. 1967. *Studies in Ethnomethodology.* Malden, MA: Blackwell Publishers.

Ghahramani, Z., and Jordan, M. 1995. Factorial Hidden Markov Models. *Proceedings of the Conference on Advances in Neural Information Processing Systems (NIPS),* vol. 8, pp. 472-478.

Goffman, E. 1963. *Behavior in Public Places.* New York: The Free Press.

Groden, M., Kreiswirth, M. 1997. *The John Hopkins Guide to Literary Theory and Criticism.* Online edition, http://www.press.jhu.edu/books/hopkins_guide_to_literary_theory/ (last referenced October 2004).

Grosz, B. J. 1977. The Representation and Use of Focus in a System for Understanding Dialogs. *Proceedings of IJCAI-77,* pp. 67-76. Los Altos, CA: William Kaufmann. Reprinted in Grosz, Jones, and Webber 1986.

Grosz, B. J., Jones, K. S., and Webber, B. L. (eds.) 1986. *Readings in Natural Language Processing.* Los Altos, CA: Morgan Kaufmann.

Grosz, B. J., Joshi, A. K., Weinstein, S. 1995. Centering: A Framework for Modeling the Local Coherence of Discourse. *Computational Linguistics,* 21 (2), pp. 203-225.

Grosz, B. J., Sidner, C. L. 1990. Plans for Discourse. In Cohen, Morgan, and Pollack, 1990,

Heritage, J. 1984. *Garfinkel and Ethnomethodology.* Cambridge, MA: Blackwell Publishers.

Herrmann, T. 1983. *Speech and Situation: A Psychological Conception of Situated Speaking.* Berlin: Springer-Verlag.

Horswill, I. D. 1998. Grounding Mundane Inference in Perception. *Autonomous Robots* 5, pp. 63-77.

Hutchens, J. 1998. Introducing MegaHAL. *NeMLaP / CoNLL Workshop on Human-Computer Conversation.* ACL.

Hutchens, J., Barnes, J. 2002. Practical Natural Language Learning. *AI Game Programming Wisdom*, pp. 602-614. Hingham, MA: Charles River Media.

Jelinek, F. 1997. *Statistical Methods for Speech Recognition.* Cambridge, MA: MIT Press.

Jordan, M. I., Ghahramani, Z., Saul, L. K. 1997. Hidden Markov Decision Trees. *Proceedings of the Conference on the Advances in Neural Information Processing Systems (NIPS),* vol. 9. Cambridge, MA: MIT Press.

Jurafsky, D., Martin, J. H. 2000. *Speech and Language Processing.* Englewood Cliffs, NJ: Prentice Hall.

Kirby, N. 2004. Moderators Report, AI Roundtables 2004. Online resource, http://www.gameai.com/cgdc04notes.kirby.html (last referenced November 2004).

Lambert, B. L. 1992. *A Connectionist Model of Message Design.* PhD Dissertation. Urbana, IL: University of Illinois.

Lamel, L., Rosset, S., Gauvain, J.L., Bennacef, S., Garnier-Rizet, M., Prouts, B. 2000. The LIMSE ARISE System. *Speech Communication,* 31 (4), pp. 339-354.

LeBlanc, M. 2003, 2004. Game Design and Tuning Workshop. *Game Developers' Conference.* San Jose, CA: CMP Group. Online resource, http://8kindsoffun.com/ (last referenced February 2005).

Lehnert, W. 1977. A Conceptual Theory of Question Answering. *Proceedings of IJCAI-77,* pp. 158-164. Los Altos, CA: William Kaufmann. Reprinted in Grosz, Jones, and Webber 1986.

Levin, E., Pieraccini, R. 1997. A Stochastic Model of Computer-Human Interaction for Learning Dialogue Strategies. *Proceedings of Eurospeech '97,* pp. 1883-1886. Rhodes, Greece.

Levin, E., Pieraccini, R., Eckert, W., Di Fabbrizio, G., Narayanan, S. 1999. Spoken Language Dialogue: from Theory to Practice. *Proceedings of ASRU99*, IEEE Workshop, Keystone, Colorado.

Linde, C., Labov, W. 1975. Spatial Networks as a Site for the Study of Language and Thought. *Language,* 15 (4), pp. 924-939.

Litman, D. J., Allen, J. F. 1990. Discourse Processing and Commonsense Acts. In Cohen, Morgan, and Pollack, 1990.

Loyall, A. B. 1997. *Believable Agents: Building Interactive Personalities.* PhD Dissertation, School of Computer Science. Pittsburgh: Carnegie Mellon University.

Maes, P. 1990. Situated Agents Can Have Goals. In Maes, P., ed. 1990. *Designing Autonomous Agents,* pp. 49-70. Cambridge, MA: MIT Press.

Mahadevan, S., Ghavamzadeh, M., Rohanimanesh, K., Theocharous, G. 2003. Hierarchical Approaches to Concurrency, Multiagency, and Partial Observability. In *Learning and Approximate Dynamic Programming: Scaling up to the Real World*, Jennie Si, Andy Barto, Warren Powell, and Donald Wunsch (ed.). New York: John Wiley & Sons.

Mateas, M. 1997. *An Oz-Centric Review of Interactive Drama and Believable Agents.* Technical Report CMU-CS-97-156, School of Computer Science. Pittsburgh: Carnegie Mellon University.

Mateas, M., Stern, A. 2004. Natural Language Understanding in Façade: Surface-text Processing. In *Proceedings of the Technologies for Interactive Digital Storytelling and Entertainment (TIDSE) Conference,* Darmstadt, Germany.

Mauldin, M. 1994. Chatterbots, TinyMUDs, and the Turing Test: Entering the Loebner Prize Competition. *Proceedings of AAAI-94*. Menlo Park, CA: AAAI Press.

Maxis. 2000. *The Sims.* Computer game. Redwood City, CA: Electronic Arts.

McKeown, K. R. 1985. *Text Generation.* Cambridge: Cambridge University Press.

McTear, M. F. 1998. Modelling spoken dialogues with state transition diagrams: experiences of the CSLU toolkit. *Proceedings of the International Conference on Spoken Language Processing*, vol. 4, pp. 1223-1226. Sydney: Australian Speech Science and Technology Association, Incorporated

Meuleau, N., Hauskrecht, M., Kim, K, Peshkin, L., Kaelbling, L., Dean, T., Boutilier, C. 1998. Solving Very Large Weakly Coupled Markov Decision Processes. *Proceedings of AAAI-98*, pp. 165-172. Menlo Park, CA: AAAI Press.

Miller, G. A. (ed.) 2005. WordNet: a lexical database for the English language. Online resource, http://wordnet.princeton.edu/ (last accessed in February 2005).

Montague, R. 1973. The Proper Treatment of Quantification in Ordinary English. In J. Hintikka, J. Moravcsik, and P. Suppes (eds.), *Approach to Natural Language: Proceedings of the 1970 Stanford Workshop on Grammar and Semantics,* p. 221-42. Dordrecht: D. Reidel. Reprinted in Thomason, 1974.

Morgan, J. L. 1978. Two Types of Convention in Indirect Speech Acts. In *Syntax and Semantics,* vol. 9: *Pragmatics*, pp. 261-280. Orlando, FL: Academic Press.

Mori, Masahiro. 1982. *The Buddha in the Robot.* Boston, MA: Tuttle Publishing.

Murphy, K. P. 2002. *Dynamic Bayesian Networks: Representation, Inference, and Learning.* PhD Dissertation, Computer Science. Berkeley, CA: University of California.

Nilsson, N. J. 1994. Teleo-Reactive Programs for Agent Control. *Journal of Artificial Intelligence Research,* vol. 1, pp. 139-158.

Nourbakhsh, I., Genesereth, M. 1996. Assumptive Planning and Execution: a Simple, Working Robot Architecture. *Autonomous Robots Journal,* 3 (1), pp. 49-67.

O'Keefe, B. J., Lambert, B., L. 1995. Managing the Flow of Ideas: A Local Management Approach to Message Design. *Communication Yearbook 18*, pp. 54-82. Thousand Oaks, CA: Sage.

Pietquin, O., Dutoit, T. 2003. Aided Design of Finite-State Dialogue Management Systems. *Proceedings of the IEEE International Conference on Multimedia & Expo* (ICME 2003), Baltimore.

Pietquin, O., Renals, S. 2002. ASR System Modeling For Automatic Evaluation And Optimization of Dialogue Systems. *Proceedings of the International Conference on Acoustics Speech and Signal Processing,* ICASSP 2002.

Pineau, J., Roy, N., Thrun, S. 2001. A Hierarchical Approach to POMDP Planning and Execution. *Workshop on Hierarchy and Memory in Reinforcement Learning (ICML).* Williams College, MA.

Porter, M.F. 1980. An algorithm for suffix stripping. *Program,* 14 (3), pp. 130-137. In Sparck Jones, Karen, and Peter Willet, 1997, Readings in Information Retrieval. San Francisco: Morgan Kaufmann.

Rickel, J., Ganeshan, R., Rich, C., Sidner, C. L., Lesh, N. 2000. Task-Oriented Tutorial Dialogue: Issues and Agents. *AAAI Symposium on Building Dialogue Systems for Tutorial Applications.* AAAI Technical Report FS-00-01, pp. 52-57. Menlo Park, CA: AAAI Press.

Rickel, J., Marsella, S., Gratch, J., Hill, R., Traum, D., Swartout, W. 2002. Towards a New Generation of Virtual Humans for Interactive Experiences. *IEEE Intelligent Systems,* 17 (4), pp. 32-38.

Rickel, J., Johnson, W. L. 2000. Task-Oriented Collaboration with Embodied Agents in Virtual Worlds. In Cassell, Sullivan, et al. 2000.

Rickel, J., Lesh, N., Rich, C., Sidner, C. L., Gertner, A. 2001. Building a Bridge Between Intelligent Tutoring and Collaborative Dialogue Systems. *Proceedings of Tenth International Conference on AI in Education,* pp. 592-594. IOS Press.

Riesbeck, C. 1975. Conceptual Analysis. In Schank, R. C. (ed.), *Conceptual Information Processing.* New Holland, Amsterdam.

Roberts, M. J. 2002. *TADS 2 Author's Manual.* Online edition, http://teladesign.com/tads/ (last referenced October 2004).

Roche, E., Schabes, Y. 1997. *Finite State Language Processing.* Cambridge, MA: MIT Press.

Rosenblatt, J. 1995. DAMN: A Distributed Architecture for Mobile Navigation. *Working Notes, AAAI 1995 Spring Symposium on Lessons Learned for Implemented Software Architectures for Physical Agents.* Palo Alto, CA: AAAI Press.

Rosenschein, S., Kaelbling, L. 1986. The synthesis of digital machines with provable epistemic properties. *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning About Knowledge,* pp. 83-98. San Mateo, CA: Morgan Kaufmann.

Rouse, R. III. 2001. *Game Design Theory and Practice.* Plano, TX: Wordware.

Roy, N., Pineau, J., Thrun, S. 2000. Spoken Dialogue Management Using Probabilistic Reasoning. *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics.*

Rubin, J. 2003. Great Game Graphics. . . Who Cares? Presentation at the *Game Developers' Conference.* San Jose, CA: CMP Group. Online resource, http://www.gamasutra.com/features/20030409/rubin_01.shtml (last accessed February, 2005).

Saul, L. K., Jordan, M. I. 1995. Boltzmann Chains and Hidden Markov Models. *Proceedings of the Conference on the Advances in Neural Information Processing Systems (NIPS),* vol. 7. Cambridge, MA: MIT Press.

Schank, R. C. 1972. Conceptual dependency: A theory of natural language understanding. *Cognitive Psychology* 3 (4), pp. 552-631.

Schank, R. C., Abelson, R. P. 1977. *Scripts, Plans, Goals and Understanding.* Hillsdale, NJ: Erlbaum.

Schank, R. C., Cleary, C. 1995. *Engines for Education.* Hillsdale, NJ: Erlbaum.

Schank, R. C., Riesbeck, C. K. 1981. *Inside Computer Understanding: Five Programs Plus Miniatures.* Hillsdale, NJ: Erlbaum.

Schlegoff, E., Sacks, H. 1973. Opening up closings. *Semiotica,* 7 (4), pp. 289–327.

Schleifer, R. 1997. Structuralism. In Groden and Kreiswirth, 1997.

Scott, D., Kamp. H. 1995. Discourse Modeling. In Cole, et al. 1995, pp. 230-233.

Searle, J. R. 1965. What Is A Speech Act? Reprinted in Adams and Searle, 1986.

Searle, J. R. 1975. Indirect Speech Acts. In Cole, P., and Morgan, J. L., eds. Syntax and Semantics, vol 3: Speech Acts, pp. 59-82. New York: Academic Press.

Seneff, S. 1992. TINA: a natural language system for spoken language applications. *Computational Linguistics,* 18 (1), pp. 61-86.

Seneff, S., Polifroni, J.. 2000. Dialogue management in the Mercury flight reservation system. *Proceedings of ANLP-NAACL2000 Workshop on Conversational Systems,* pp. 11–16.

Sidner, C. Focusing in the Comprehension of Definite Anaphora. In M. Brady and R. Berwick, eds., *Computational Models of Discourse,* pp. 331-371. Cambridge, MA: MIT Press. Reprinted in Grosz et al., 1986.

Sibun, P. 1991. *Salix: a strategy-based architecture for text organization.* PARC Technical Report. Palo Alto, CA: Xerox Palo Alto Research Center.

Singh, S., Litman, D., Kearns, M., Walker, M. 2002. Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System. *Journal of Artificial Intelligence Research,* 16 (2002): 105-133.

Suchman, L. A. 1987. *Plans and Situated Actions: The problem of human machine communication.* Cambridge: Cambridge University Press.

Temperley, D., Sleator, D., Lafferty, J. 2005. *Link Grammar.* Online resource, http://www.link.cs.cmu.edu/link/ (last referenced March 2005).

Theocharous, G., Mahadevan, S. 2002. Approximate Planning with Hierarchical Partially Observable Markov Decision Processes for Robot Navigation. *IEEE Conference on Robotics and Automation* (ICRA). Washington, D.C.

Thomas, F., Johnston, O. 1981. *Disney Animation: The Illusion of Life.* New York: Abbeville Press.

Thomason, R. H. (ed.) 1974. *Formal Philosophy: Selected Papers of Richard Montague.* New Haven, CT: Yale University Press.

Traum, D. R. 2000. 20 Questions for Dialogue Act Taxonomies. *Journal of Semantics,* 17 (1), pp. 7-30.

Wallace, R. S. 2000. Don't Read Me: A.L.I.C.E. and AIML Documentation. Online resource, http://alicebot.org/articles/wallace/dont.html (last referenced November 2004).

Wallace, R. S. 2004. Zipf's Law. Online resource, http://www.alicebot.org/articles/wallace/zipf.html (last referenced November 2004).

Weizenbaum, J. 1966. ELIZA—A Computer Program for the Study of Natural Language Communication Between Man and Machine. *Communications of the ACM*, 9 (1), pp. 36-45.

Wierzbicka, A. 1987. *English Speech Act Verbs.* Orlando, FL: Academic Press.

Wilensky, R. 1981. *PAM.* In Schank and Riesbeck, 1981.

Woodcock, S. 2002. AI Roundtable Moderator's Report. Online resource, http://www.gameai.com/cgdc02notes.html (last referenced November 2004).

Woodcock, S. 2000. AI Roundtable Moderator's Report. Online resource, http://www.gameai.com/cgdc00notes.html (last referenced November 2004).

Woodcock, S. 1998. Artificial Intelligence Moderator's Report. Online resource, http://www.gamasutra.com/features/gdc_reports/cgdc_98/woodcock.htm (last referenced November 2004).

Woodcock, S. 1998b. Game AI: State of the Industry. *Game Developer,* October 1998. Also available online, http://www.gamasutra.com/features/19981120/gameai_01.htm (last referenced January 2005).

Young, S. 1999. Probabilistic Methods is Spoken Dialogue Systems. *Proceedings of the Royal Society,* September 1999. London.

Zipf, G. K. 1935. *The Psycho-Biology of Language.* Reprinted 1965, Cambridge, MA: MIT Press.

Zubek, R., Khoo, A. 2002. Making the Human Care: On Building Engaging Bots. *Proceedings of the 2002 AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment.* AAAI Technical Report SS-02-01. Menlo Park, CA: AAAI Press.

Zubek, R. 2004. Character Participation in Social Interaction. *Proceedings of the Challenges in Game AI Workshop,* AAAI Technical Report WS-04-04. Menlo Park, CA: AAAI Press.