Character Participation in Social Interaction

Robert Zubek

Northwestern University
1890 Maple Ave.
Evanston, IL 60201
rob@cs.northwestern.edu

Abstract

This paper introduces a technique for participation in certain kinds of social interactions. By modeling their temporal structure explicitly using a hierarchy of concurrent Markov processes, we can track the development of the interaction robustly, and drive it forward in a coherent fashion. The approach is computationally inexpensive, and exhibits desirable performance.

Introduction

Multiplayer games are filled with social engagement – players treat each other not merely as abstract game characters, but as living, social, emotional human beings. They converse and deal with each other just as they would in real life. But in trying to replicate these engagements in computer-controlled characters, we immediately run into numerous problems: conversations that guide human interactions rely on tacit social conventions and assumptions about the participants. This makes social interaction notoriously complex, ambiguous, and noisy.

In this work we will focus on a mechanism that attempts a range of such engagements believably, efficiently, and in spite of the lack of complex language processing skills. We accomplish this by focusing in particular on well-structured interactions. Certain kinds of interactions exhibit coherent temporal structure, though manifested through ambiguous language; we expect modeling this structure explicitly will provide background knowledge to the system that will be sufficient to disambiguate the language, and drive the interaction coherently forward.

In the following discussion we introduce *hierarchical interaction protocols*, a technique for modeling some of these more constrained social interactions. Under this approach, explicit models of interaction produce behavior that is both robust and very inexpensive computationally. This approach is currently being used to implement a language-based social interaction game, detailed below. The system follows structured conversation, exhibits desirable performance characteristics, and consumes a trivial number of CPU cycles.

Motivating Example

The Breakup Conversation is a simulation "game" currently being developed using hierarchical interaction protocols. We will use it to illustrate the interaction technique detailed in this paper.

The *Conversation* is an exploration of the dialogue that signals the end of a romantic relationship. The player connects with the simulated significant other over internet messenger, and the goal is to perform a successful breakup. Figure 1 presents an example.

```
Alice: can we talk?
Bob:   :)
Bob:   ok
Alice: it's kinda important
Bob:   okay
Alice: it's about us
Bob:   well
Bob:   i've got to go back to work
Alice: i don't know how to break this to you
Bob:   no!
Alice: but i don't think this is working out
Bob:   no we're not talking about this right now
Alice: please
Bob:   why are you doing it like this
Alice: i'm sorry
Bob:   what does that even mean?
Bob:   it's because i got no job huh?
Alice: i didn't mean to do it like this
Bob:   you dislike that i'm unemployed
Bob:   isn't it true
Bob:   but anyway
Alice: no that's not true
...
Bob:   please, honey, can you give me
       another chance?
Alice: no i don't think so
Bob:   you're just being cruel
Alice: i didn't mean to
Bob:   well then
Alice: look
Bob:   you think i'm not what you were looking for
Alice: i'm sorry
Bob:   i don't believe it
Bob:   can you give me another chance?
Alice: that's not working out
Bob:   that was such a mean-spirited thing
       to say to me
```

Figure 1. Conversation excerpt from a game prototype. Computer character 'Bob' shows movement from initial problem intimation, through panic and refusal, to attempting to guess what the problem is; a little later it engages several emotional maneuvers. *The Breakup Conversation*, version from May 2004.

The player begins by selecting parameters for the computer character: name, gender, caricatured personality type, and one of the possible relationship contexts. Then a chat window opens, and the player tries to successfully get through the breakup conversation. The interaction takes place entirely via typed English text, in real time, and takes the player on an exploration of breakup space.

The computer will have the knowledge of some patterns typical of a breakup conversation and the ways of getting through them, inspired by Berne (1968) as well as informal observation; 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 character's settings determine which patterns and transitions are preferred. The conversation ends once the computer character is successfully persuaded that the relationship is over.

The overall interaction is modeled by decomposing breakup conversations into a hierarchy of simpler protocols, corresponding to the individual components. This composite representation tracks conversation progression on multiple levels simultaneously. 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.

Implementation

The technique works by modeling the interaction as a hierarchy of partially observable Markov decision processes (POMDPs) that continually evaluate the situation and drive it forward.

The overall system works as follows. Because we can never observe directly where the conversation is at any given moment, we have to maintain multiple guesses about the ongoing situation, represented as a collection of POMDPs. A single partially-observable Markov decision process is a probabilistic state space that estimates the position of a particular protocol, and suggests actions (illustrated in figure 2).

Each state is annotated with communicative expectations – for example, specifying that at the beginning of a greeting one person should say a familiar greeting phrase, the other should respond, then one of them can ask about the health of the other, and so on.

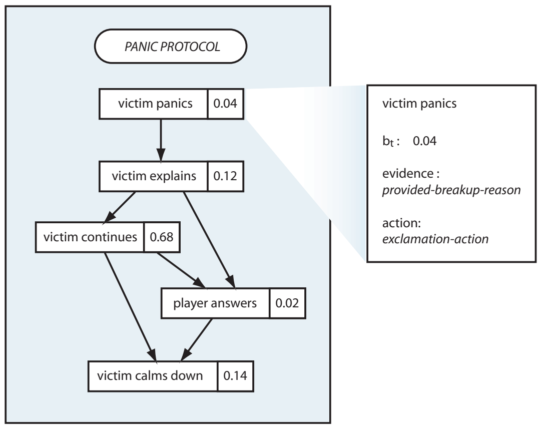

Figure 2. State space example. b_t is a variable that holds position belief for each state. Each state includes evidence and action expectations.

At every iteration of the control loop, incoming text is treated as communication evidence, and categorized using shallow parsing and pattern matching. Then the history of observed evidence is taken into account, and the current state of each process is reassessed. The result is an updated belief of where we are in the overall situation. Finally, the processes suggest what actions should be performed based on their new beliefs.

In this manner, even though the state of the interaction cannot be observed directly, it can be estimated based on the history of communication. Unfortunately, our observations will be uncertain, because language is noisy and ambiguous. In practice, however, this turns out to be quite satisfactory – the hierarchical, probabilistic representations helps us cope with that.

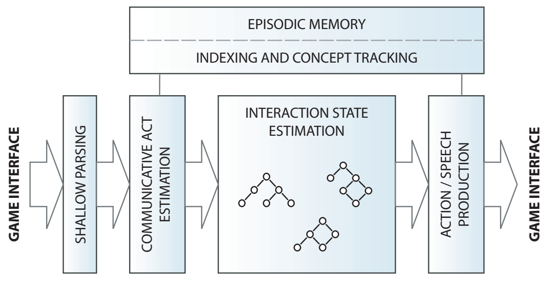
EPISODIC MEMORY
INDEXING AND CONCEPT TRACKING
GAME INTERFACE
SHALLOW PARSING
COMMUNICATIVE ACT ESTIMATION
INTERACTION STATE ESTIMATION
ACTION / SPEECH PRODUCTION
GAME INTERFACE

Figure 3. System Architecture.

The system architecture is roughly as presented in figure 3. Of course, the diagram is simplified; the major point is that data flows through the system in a simple, single pass, and a single iteration of the control loop is therefore quite straightforward.

Control Loop

The system continually recomputes the position belief of all processes, reassessing the overall situation and suggesting actions to be performed.

A single iteration performs the following three steps:

1. Communicative act categorization. This first step categorizes the incoming utterance into a number of possible *communicative acts*: assistance operations (such as request, or help), social standing operations (insult, praise), speech acts, particular conversational moves, and others as necessary for given situation.

Categorization is done mainly with shallow parsing and pattern matching, in order to guarantee fast performance. Simple parsing suffices thus far, thanks to the rich context model provided by the probabilistic hierarchy; if this turns out to be insufficient, a more robust recognition mechanism can always be brought in to replace it. This leads to reasonable speed and robustness of the system, although more subtle expressions are, of course, completely lost.

Categorization produces a value a_t, specifying what communicative act or acts were observed at the given time.

2. Each POMDP tracks situation progress. After categorization, the system tries to estimate the participants' current position in each process. The state spaces are probabilistic, therefore position in the state space is really a probability of position. We call this a *position belief,* designated as b_t, distributed over all states.

Computing position belief is discuss briefly here, and covered in greater detail in the appendix.

The interaction designer specifies the state space for the given POMDP, including the probabilistic space transition function denoted as $\tau(s, s')$ for a transition from s to s'.[1] The designer also specifies communication expectations for each state, as a probabilistic relation between states and communicative acts, denoted as $e(s, a)$.

[1] Quick note on notation: probability distributions are written in the mathematical functional notation – for example, the belief probability for a given state s at time t would be denoted $b_t(s)$. Of course, this is not to imply they should be implemented as actual function calls. An efficient implementation represents all those distributions as matrices or vectors: $b_t(s)$ in C++ ends up as an array b[s], $\tau(s,s')$ becomes t[s,s'], etc.

Finding the new position means recalculating the position belief distribution over all states: $b_t(s)$ for every state s in state space S. Given that we have $b_{t-1}(s)$, the belief from last iteration, current belief is calculated as follows:

$$b_t(s) \quad = c_t e(s, a_t) \sum_{s_i \in S} \tau(s_i, s) b_{t-1}(s_i)$$

This can be understood as follows. The probability of being at some particular state s is: the probability of having been at some neighboring state (b_{t-1}) in the previous clock tick, times the probability of having transitioned (τ), added together over all neighboring states, summed and multiplied by the evidence (e) for being at the current state given what we've observed. c_t is merely a normalization constant, to make sure belief over all states adds up to one. Details of this formula are provided in the appendix.

All of these elements are very easily computed. Transition function τ and previous belief distribution b_{t-1} are a matter of lookup, and e is specified at design time, although finding the value of one of its inputs requires simple language processing. So the computation above can be represented as a sequence of vector operations (potentially using sparse representations), rendering the position belief calculations extremely fast.

3. Each POMDP suggests action production. Having found present location, action production then becomes very simple – in this case, implemented as template-based text generation. Based on the current belief distribution, each process suggests an action. All the actions are aggregated and arbitrated, using a mechanism such as a *winner-take-all* rule.

Performance

The simulation game has been in development since January, and is not yet complete. More details will be presented at the workshop; however, even just the intermediate results should prove interesting.

The engine itself is written in C++. Processes are defined in a custom definition language, which a Lisp-based engine converts into highly efficient C++ code for the interaction processes. This code gets compiled, along with the engine, into a stand-alone DLL. Separately, a third-party parser (the Link Parser, see Sleator and Temperly, 1991) is used for utterance processing, also as a separate DLL. Finally, the game itself is a GUI wrapper around both libraries.

Processing time for the engine is shown in figure 4, based on a number of tests on a 1.8GHz Pentium 4. The figures include parsing, processing, and text generation, but exclude user interface and game-specific elements. Input parsing and preprocessing takes roughly 0.01 second plus 0.003 second per word (with some variation). The rest of the system (state estimation and action generation, which are performed independently of input processing) take less than 0.001 second per iteration on average, and therefore do not even figure in performance calculations.

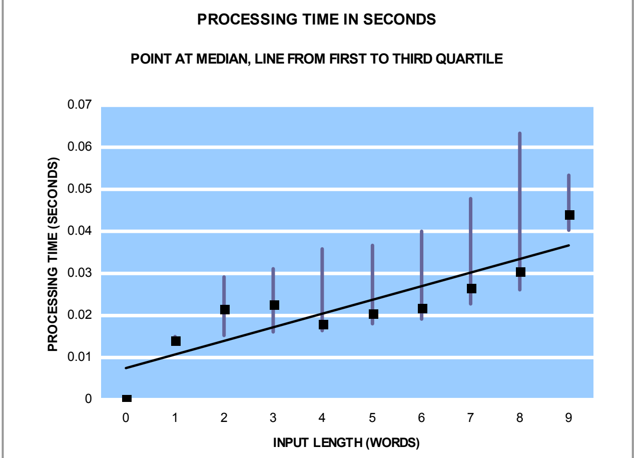

Figure 4. System processing time per iteration. Performance is roughly linear with input size, with acceptable variation. Current system runs one iteration per second. *The Breakup Conversation,* version from May 2004.

System Behavior

The hierarchical model introduces multiple levels of representation of the same phenomenon, and therefore increases redundancy – even if some particular process 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.

The concurrent activation of multiple protocols means that, if we can decompose an interaction into independent components, those can be implemented cheaply as separate processes. In practice, we find that this decomposition is possible for a number of desirable interactions.

Stochastic modeling of these processes allows the system to deal robustly with noisy and ambiguous inputs. Informally speaking, communicative act categorization present the system with multiple guesses about the player's utterance, and stochastic processes maintain numerous concurrent hypotheses about what goes on in the interaction. Communicative acts then help collapse the processes into 'correct' positions, and do so even if the significance of player's input wasn't completely understood. This aids in graceful recovery from confusing 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 game manifests long-term consistency, including coherence within a particular breakup 'stage', and coherence in transitioning between stages.

Finally, modeling the hierarchy as a network of concurrent POMDPs allows for very efficient implementation.

Related Work

This project is closely related to work on probabilistic finite-state dialogs. These techniques avoid complex processing by collapsing both the conversation and the task at hand into a unified finite state space, usually represented as POMDPs (for example, Singh, Litman, Kearns, and Walker, 2002). Hierarchical POMDPs are a great extension to this approach (for example, Roy, Pineau and Thrun, 2000), and subject of active research.

The approach is also indebted to the believable agents research, such as by the Oz group (for an overview, see Bates et al., 1991) or, more recently, Mateas and Stern (2002).

Chatterbots with minimal state representation should also be mentioned, such as Alice (Wallace, 2004) and MegaHAL (Hutchens and Alder, 1998). They occupy a similar niche but, unlike this project, do not model long-term structure of conversation.

Appendix: POMDP Details

POMDP Description. Let $A = \{a_1, \ldots, a_m\}$ be the set of discrete actions that can be performed by participants, and let each probabilistic interaction be a finite-state process description consisting of:

- $S = \{s_1, \ldots, s_n\}$, the set of discrete states,
- $\tau : S \times S \to [0,1]$, the state transition probability, where $\tau(s, s') = p(s' \mid s)$,
- $e : S \times A \to [0,1]$, the probability of observing some expected action a at the state s, or: $e(s, a) = p(a \mid s)$, and
- $\pi : S \times A \to [0,1]$, the probability of performing the action a at state s.
- $b : \mathrm{Z} \times S \to [0,1]$, the position belief, allowing for the shorthand: $b_t(s) = b(t, s)$.

We require of the transition probability that $\forall s \in S: \sum_{s' \in S} t(s,s') = 1$, and of position belief that $\forall t \in \mathrm{Z}: \sum_{s \in S} b_t(s) = 1$.

State Estimation. To find the agent's position in the state space, b_t is calculated using popular state estimation techniques for hidden Markov models (Jelinek 1997, Fox et al. 2001).

Probability of being in a given state at time t is dependent solely on the sequence of actions leading up to t:

$$b_t(s) \quad = p(s \mid a_t, a_{t-1}, \ldots, a_0, s_0)$$

This probability is unknown, but assuming independence of observed actions, we can explore the Bayes rule to transform the above equation:

$$b_t(s) = c_t p(a_t \mid s, a_{t-1}, ..., a_0, s_0) p(s \mid a_{t-1}, ..., a_0, s_0)$$

Here c_t is our denominator of Bayes' rule, and a normalization factor to ensure that $\sum_{s \in S} b_t(s) = 1$.

Markov assumption is then used to simplify the result. Details are omitted due to space constraints (see Jelinek, 1997), but using the Markov assumption and total probability theorem we arrive at the following:

$$b_t(s) \quad = c_t p(a_t \mid s) \sum_{s_i \in S} p(s \mid s_i) b_{t-1}(s_i)$$

Or, in previously defined notation:

$$b_t(s) \quad = c_t e(s, a_t) \sum_{s_i \in S} \tau(s_i, s) b_{t-1}(s_i)$$

Given probabilistic state estimation, recognizing which processes are engaged becomes trivial. It simply requires a special state to represent disengagement. Let each interaction include a unique initial state s_0, and corresponding transitions. The state space will be treated as disengaged when $p\ (s_0) = 1$. Creating rules for engaging or disengaging the entire state space then becomes a matter of specifying appropriate values of $e\ (s_0, A)$, $\tau\ (s_0, S)$, and $\tau\ (S, s_0)$.

Action Production. Each state is annotated with actions to be performed, including communication and self-adjustment. State-based policy π determines which action will be produced. For many states and actions it can be the case that $\pi\ (s, a) = e\ (s, a)$. However, due to the complex nature of these interactions, as well as complex aesthetic requirements of entertainment products, we do not intend for the policy to be learned automatically.

Deictic Representation. One element had not been mentioned before. The system uses a semantic network to store additional knowledge about the setting, and *deictic markers* (Agre and Chapman, 1987) to coordinate between probabilistic processes and the network. Deictic markers extend the essentially propositional POMDPs, allowing for limited relational representation. Unfortunately, due to space constraints, this is as much as we're going to say about the semantic network or deixis here.

Acknowledgements

I would like to thank my colleagues at the computer science department, and especially Ian, Robin, and Ayman, for their insightful comments and critique of this work.

Bibliography

Agre, P., and Chapman, D. 1987. "An Implementation of a Theory of Activity." In *Proceedings of AAAI-87.* Menlo Park, CA: AAAI Press.

Bates, J., Loyall, B., and Reilly, W. S. 1991. "Broad Agents." In *Proceedings of the AAAI Spring Symposium on Integrated Intelligent Architectures,* Stanford University, March 1991. SIGART Bulletin. Vol. 2. No. 4. August 1992.

Berne, E. 1968. *Games People Play: The Psychology of Human Relationships.* New York, NY: Grove Press.

Fox, D., Thrun, S., Burgard, W., and Dellaert, F. 2001. "Particle Filters for Mobile Robot Localization". In Doucet, A., de Freitas, N., and Gordon, N., eds. *Sequential Monte Carlo Methods in Practice.* New York, NY: Springer.

Hutchens, J., Alder, M. 1998. "Introducing MegaHAL." In D. M. Powers (ed.) *Proceedings of NeMLaP3/CoNLL98,* 271-274. Somerset, NJ: Association for Computational Linguistics.

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

Mateas, M. and Stern, A. 2002. *Architecture, Authorial Idioms and Early Observations of the Interactive Drama Façade*. Technical Report CMU-CS-02-198, School of Computer Science, Carnegie Mellon University.

Roy, N., Pineau, J., and Thrun, S. 2000. "Spoken Dialogue Management Using Probabilistic Reasoning." In *Proceedings of ACL-2000*, Hong Kong.

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

Sleator, D., and Temperley, D. 1991. *Parsing English with a Link Grammar.* Technical Report CMU-CS-91-196, School of Computer Science, Carnegie Mellon University.

Wallace, R. S. 2004. *The Anatomy of A.L.I.C.E.* http://www.alicebot.org/anatomy.html. Last access 5/24/04