Internal Use Only

TR-I-0370

Multi-Agent Communication and Commitment and The BEHOLDER family of algorithms for scheduling multiple parallel uncertain processes under limited-resource conditions
マルチアゲントコミュニケーションンと約束とマルチポロセスのスケジュリング

John K. Myers

March 12, 1993

Abstract

This report deals with two main concepts: communication and commitment between multiple agents, and scheduling multiple parallel uncertain processes.

Communication between multiple agents in the real world requires many rules dealing with intentions, expectations, and commitments. It is important to understand these rules in order to understand how people act, and how computer agents should act. This report explores some of the theory behind these rules.

The second protion of this report discusses the theory behind scheduling tasks for multiple processes, and presents algorithms to deal with specific cases..

Multi-Agent Communication and Commitments and The BEHOLDER family of algorithms for scheduling multiple parallel uncertain processes under limited-resource conditions

John K. Myers

ATR Interpreting Telephony Research Laboratories
Hikari-dai 2-2
Seika-cho, Soraku-gun, Kyoto 619-02, Japan
myers@atr-la.atr.co.jp

Abstract

This report deals with two main concepts: communication and commitment between multiple agents, and scheduling multiple parallel uncertain processes.

Communication between multiple agents in the real world requires many rules dealing with intentions, expectations, and commitments. It is important to understand the rules of communication in order to understand how humans act, and to be able to construct computer agents that can interact well with humans and with each other. This report explores some of the theory behind these rules.

The second portion of this report discusses the theory behind scheduling tasks for multiple processes, and presents algorithms to deal with particular cases. In a parallel or distributed system with many processors (each of which can only run one task at a time), and many tasks to be run (where the number of tasks is greater than the number of processes), there is a need to select or *schedule* which tasks should be run on which processors. This need is especially important when the system is operating under conditions of limited resources–in other words, when the system needs to finish as soon as possible–and there is not enough time to run all of the tasks. Each task has a value, and the system must maximize the system's total value score before the system is terminated at the cutoff time. If the tasks are of uncertain duration or uncertain value, the problem becomes more complex. The scheduler must take responsibility for suspending tasks that are taking too long to complete, and for swapping them out and replacing them with new tasks.

Contents

List of Figures

1 Introduction

This report deals with communication between multiple agents, and commitments. If there is just one computer executing just one program that is guaranteed to run and then succeed, then there is no need of the theory presented in this report. Indeed, most of the computing that has been performed in the world up until now has been done under this paradigm. If, however, there are multiple agents in the picture (more than one person, or a person and a computer, or two or more computers), and if actions are not guaranteed to have a single outcome but could have one of several possible outcomes or could fail when executed, then the theories presented here become necessary. It is important to have an understanding of how agent-to-agent communication and commitments work, so that proper interactions can be designed and improper interactions understood, debugged, and corrected.

The theory in this report will be useful for three kinds of situations. These are:

Analyzing Human-to-Human Communication. Humans communicate and make commitments to themselves and to each other. In order to understand how this process works, predict what will happen when the process of making a commitment starts, and diagnose why misunderstandings happen and why people become upset, it is necessary to understand the communication process and how commitments work.

An immediate example for an interpreting telephone would be to analyze and understand a conversation between two humans (e.g., "I will send you the form."), and predict what utterances will occur next.

Designing Human-to-Machine Communication. In order for a machine to interact well with a human, it is necessary for the machine to understand what it means to make a commitment, and how commitments work. It goes without saying that the machine must also have intentions [Mye92c], especially in the case where the machine promises to attempt to perform an action in the future, but the machine is not guaranteed immediate success of the action. In order for the machine to be able to understand these concepts, the machine designer must understand these concepts and build them into the machine.

For instance, this theory would be useful when designing an intelligent interpreting telephone that acts as a "broker" between two humans instead of as a "pipeline" [OCP90, Ovi88]. We can imaging the following conversation:

```
Office:  "Tell him I want to meet him at the conference desk."
Machine: "I will tell him that."
Office:  "Also tell him that he needs to bring some money with him."
Machine: "I will tell him that, too."

Machine: "The office would like to talk with you about how to register."
Caller:  "Yes, well, I'm very confused.  I want you to ask him which city
          I should fly in to."
```

At this point, the machine has promised the office that the machine will communicate two specific sentences of information. However, the caller has directly requested the machine to perform a specific translation. This is a problem, and there are a number of ways for the machine to deal with it. In any case, it is necessary for the machine to have intentions, so that the information which is flowing back and forth does not get lost. The machine must form the intentions to communicate the two sentences' worth of information from the office to the caller, and also the one sentence from the caller to the office. The machine could take care of the caller, and drop down into a subdialogue about what the caller does not understand, how the office can help, and what the caller needs to know. Then, after that, the machine could come back and fulfill its intention of communicating the office's information about the conference desk and the money. Or, the machine could go ahead and communicate the office's information first, and then have to remember its intention to come back to the caller's problem, explore that, and communicate the results of the caller's information to the office. Other alternatives are probably possible. An advanced machine would know when the conversation changes enough so that communicating the previous information does not make sense any more (say, for instance, that the caller will be flying to a different conference in a different city, and will not need to meet the office person at the first conference desk). In any case, a good understanding of intentions and commitments is necessary.

All of this assumes a straightforward machine with no other constraints or resource problems that has been directed to be as helpful as possible. If the machine has limited resources, or has been given a directive to be as efficient as possible (say that the machine is very expensive, service is charged by the call, and other callers are waiting), then the machine may have to choose whether a requested piece of information to be translated is important enough or is too trivial to be translated. In this case the machine must consider breaking commitments to translate, or not making commitments, and then manners as to how or whether to inform the speaker that something has not been translated.

Designing Machine-to-Machine Communication. Recently the problem of "distributed computing" has become popular, in which, instead of one large computer, many medium-sized computers get together and work on a problem. If the execution of a computing action is infallible, then there is no need for intentions, commitments, or major communications. For instance, if a computer can send a printer a file and the printer will guarantee to print the file, then there is no problem; the computer can "fire and forget" the file over to the printer. If, however, the printer is faulty and can run out of paper, or if the printer is faulty and can sometimes print ugly spots on the page, or if the job is too large but needs to be done in a hurry, then sometimes a computer might want to send the job to a different printer after learning of no paper, or send the same job to two different printers to get better chances of having one printer print the job correctly, or split the job up and send it to two or more printers that are free at the moment. In these cases, it is necessary to have advanced communications, and it is useful to understand the theory of intentions and commitments. Of course, this example of printers and print jobs is explained simply because it is easy to understand; in real distributed computing, the printers would be computers, and the distributed print jobs would be computational packets to be executed. Advanced communication in distributed computing requires a good theory of commitments.

Rather than talk about "agent Alpha" and "agent Beta", and then have difficulties referring to them, this report will use the short form of agent "Al" (masculine) and agent

"Beth" (feminine) for examples.

2 Why Is Multi-Agent Communication Important?

Of course, understanding how people communicate with each other is important. And, it is also important to be able to design communications between people and machines.

However, perhaps the most important reason for investigating multi-agent communication is the coming new generation of portable computers requiring distributed fallible resources. In current designs for next-generation computers, a person will use a notebook computer that has a local hard-disk, internal memory, processor, and screen, but does not do all of its computation by itself. It is linked by radio or IR to a computer resource network of many large support computers. Instead of just sending mail messages or messages containing data, a notebook computer will also send support computers requests for computation and requests for storage and memory usage. However, there will be problems, since these requests are fallible (communications may fail, or the computer may be too busy to handle the requests), and can take an uncertain amount of time. The notebook computer will probably want to send the same request to two or three support computers for verification, and to ensure that at least one computer can receive the message and take the job. It is necessary to develop a theory of communication and commitments, as presented here in this paper.

3 Types of Messages

There are many types of messages that can be communicated. A complete theory would attempt an exhaustive list; here this report merely mentions some of the more important types.

- Broadcast vs. Directed
- General Information
- Partial Results (Specific Information, to be worked with)
- Requests for Information
- Requests for Actions
- Acknowledgement of Message
- Communication of Need (General Request for Help)
- Communication of Busy-ness (free, partially loaded, tied up)

4 A Theory of Commitment

4.1 The Four Kinds of Commitment

The word "commitment" has a cloud of confusion around it because in the past it has been used to refer to four different concepts. These are: Internal Commitment, Commitment Utterance, Verbal Commitment, and Interagent Commitment.

4.1.1 Internal Commitment

An Internal Commitment is a dedication inside the agent itself towards performing a particular action.

4.1.2 Commitment Utterance

A Commitment Utterance is an utterance or other communication (letter, computer message, etc.) that is the literal phrase, "I make a commitment to do *X*", where *X* is an action.

Often other phrases conventionally count as being a Commitment Utterance, such as "I will do *X*". Conventional generation depends on the situation and which agent is using which conventions. Thus, for instance, one agent may think that the utterance "I will think about possibly doing *X*," counts as a Commitment Utterance, whereas another agent may not think that way.

4.1.3 Verbal Commitment

A Verbal Commitment is the act of physically uttering or communicating a Commitment Utterance.

It is possible for two agents to agree that agent Al in fact performed an utterance, and yet not agree on whether the utterance was a Commitment Utterance or not.

It is possible for a Verbal Commitment to have been made, and yet not to have been received. It is possible for agent Al to believe that the Verbal Commitment that he performed was in fact received (heard) by agent Beth, whereas agent Beth did not receive the Verbal Commitment or did not believe that a Verbal Commitment was made.

4.1.4 Interagent Commitment

If an agent Beth believes that an agent Al has performed a Verbal Commitment in the presence of Beth, then Beth may believe that Al has an Interagent Commitment, or duty, to Beth to perform the action that Beth believes Al has committed to. In other words, Beth thinks Al ought to do the action.

Similarly, if Al believes that Al has performed a Verbal Commitment to Beth, and Al is sincere in his commitment, then Al may believe that Al has an Interagent Commitment to do action *X* for Beth. In other words, Al thinks Al ought to do the action.

Note that this definition does not say anything about mutual knowledge. Some people may think that the definition of a true Interagent Commitment requires validated mutual knowledge of the commitment. This is not a good definition because it is difficult, if not impossible, to obtain in practice, and it ignores the more relevant case of one-sided belief.

For instance, note that Beth may believe Al has made an Interagent Commitment to Beth, but Al may not believe that Al has made an Interagent Commitment to Beth. In this case, Al will probably not perform the action, whereas Beth will be expected Al to perform the action. This will probably lead to Beth becoming angry and mistrusting Al.

Similarly, Al may believe that Al has made an Interagent Commitment to Beth, whereas Beth may not believe that Al has made a commitment to Beth. This could result in Beth repeatedly asking Al for assurances, or in Al performing a wasted action that Beth does not need.

Note that it is possible for Al to make a Verbal Commitment and yet to not be sincere. In this case, Al will not form an Interagent Commitment.

In addition, the definition doesn't say anything about whether Beth thinks Al thinks Al ought to do the action. Thus, it is possible for Beth to believe that Al has an Interagent Commitment to Beth, and yet for Beth to believe that Al does not believe that Al has an Interagent Commitment to Beth. Such a case can get quite complex.

Note that an Interagent Commitment is different from an Internal Commitment. An agent Al can agree that he certainly ought to perform an action for Beth, and yet never get around to affirming to himself that he will in fact perform the action. However, this is a very fine line. In general, most all Interagent Commitments should form Internal Commitments.

Note also that an Internal Commitment is different from an Interagent Commitment. An agent Al can decide that he is definitely going to do something for himself, and yet never tell Beth about it.

5 Commitments have Graded Strengths

Commitments are *not* binary, as most other authors would have one believe. Each of the different types of commitments has a graded strength. For instance, consider the following commitments:

- I swear a blood oath that I will do that or die.
- I vow to do that.
- I will do that no matter what!!!
- I promise I will do that.
- I will do that.
- I will do that when I can.
- I will do that when I get the time.

- I think I will probably do that.
- I'll do that if I'm not doing anything else.
- I might do that sometime, if I'm not busy.
- Perhaps I might do that.
- I'll look into doing that.
- I will certainly think very hard about doing that.

6 Intentional Action Refresher

How do agents choose to perform actions? This question has been discussed at length elsewhere [Mye92c,Mye92a,Mye91,Mye92b], but I will give a brief refresher of the relevant sections here.

Actions can be deterministic or nondeterministic. Agents have *proattitudes* consisting of *wants*, *requests*, and *morals* ("*shoulds*"), which are all internalized requests from one part of the agent's mind to the judgement/decision-making/scheduling part of the agent's mind for the agent to perform an action. Agents have a *moral filter* that prevents the agent from considering proattitudes that are bad or against the agent's principles. If the agent judges that the proattitude is worthy and chooses the requested action, and decides when and how to try to perform the action, the proattitude becomes an intention. Intentions are ordered both by time and by importance. If the agent is not busy and has some free time, and the agent believes that the agent *can* try to execute the intention[1], and the intention is the most important intention in the intention stack, then the agent *plumps for* the intended action and immediately attempts to try to execute the action (including making specific preparations for execution).

In order that an agent *can* start try to do an action, a number of requirements are necessary. The agent must have sufficient mental skill; physical capability; temporal resources; physical and material resources; fortune (good luck); conscience, feelings, and morals for the action; required knowledge; permission; enabling situational state variables; and subaction abilities. For a more in-depth exploration of the meaning of "can", see a companion report [Mye90c].

If an agent believes that the agent perhaps Can do an action, and the agents Plumps For the action, then the agent in fact starts trying to do the action. If the agent's model of the universe is sufficiently accurate and the agent is sufficiently lucky, then the action does in fact get performed and an outcome occurs, usually in a nondeterministic fashion. An agent with an internal commitment to a particular result will in general endeavor by continuing to attempt to perform the action until the intended result is perceived as being achieved, or until the chances of success are seen as being too slim and something more important to do comes along.

[1]Actually there are cases in which this requirement is not necessary. For a discussion, see [Mye92c].

The strength of the commitment governs the importance of achieving success, which governs the degree of endeavor. If an agent is strongly committed to achieving something, then the agent will continue to endeavor strongly and will tend to reject distractions.

In a successful agent, degree of commitment corresponds to degree of excitation given to the endeavoring/planning actions. Important things have much energy thrown at them, even if they seem to be a sure thing.

7 Deciding On and Scheduling a Newly Chosen Action

Given this background, how then do agents judge, choose, and decide on whether and when to perform a proposed action?

First, a proposed action is tested against the agent's ethics, morals, and social norms, to see whether the action is not good to do.

Next, the action is evaluated by comparison against compiled policies and interpreted weighted values, to see whether the action is important enough to be done, and whether it is important that the agent do the action or not. The expected results of the action are normally incorporated into this judgement. If the proposed action passes these tests, then the action is chosen to be done by the agent if possible and convenient.

In non-complex computer agents, the previous two paragraph steps can be skipped.

The agent already has a Contingency Plan, which is a tree of intended actions, possible outcomes, and resulting intended reactions, etc., that the agent is intending to execute at present. Each outcome has an evaluated *utility* attached to it. Each action fork has an *inertia* score associated with it, that is determined by the expected utilities of its possible outcomes, and how hard it was to create and evaluate the plan for that action. A newly proposed chosen action will also be evaluated and assigned an expected utility. If the chosen action does not conflict with any previous plans (because it uses idle resources), then the agent immediately decides on doing the action and schedules it for immediate execution. Chosen actions will usually conflict because most agents cannot do more than one significant thing at one time. If the chosen action conflicts with a previously intended plan and the utility of the action is less than the inertia of the plan, then the chosen action will be immediately rejected and assigned the status of a Sometime Future Intention, because the agent is already committed to doing something more important. If the chosen action conflicts but the utility of the action is more than the inertia of the previous plan, then the agent will compare the chosen action agains the previously intended plan and see what can be done. If the chosen action does not truly conflict with the plan, then the agent may be able to fit the chosen action into the plan by interleaving actions, or otherwise modifying the plan. If the action truly conflicts, then the agent must do a full evaluation again, and pick the chosen action or the previously intended plan, whichever has the highest utility[2]. If the utility of the chosen action is in fact higher, then the agent will change its previous intention and assign it the status of a Sometime Future Intention, while turning the newly chosen action into an intention.

[2]Utility will in general not be a fixed first-order number but will be an uncertain number with a second-order confidence distribution. This advanced effect may be ignored in first-order systems.

Thus, to summarize, whether an agent internally commits (forms an intention) to do a proposed action depends on five things:

- If the agent thinks the action is "bad" or not;
- If the agent thinks the action is important enough to be done;
- If the agent thinks that the agent should do the action;
- How important the action is;
- How busy the agent expects to be, and how important the activities that the agent has already planned to do are.

Whether an agent actually plumps for (starts trying to do) an intended action depends on:

- What other important things the agent is intending to do;
- Whether and when the agent can do the intended action;
- How busy the agent is with other interruptions.

Whether an agent actually finishes an action that the agent has started depends at least on:

- How long the action is expected by the agent to take;
- How difficult the action appears to be to the agent;
- How busy the agent is with other things;
- How much good or bad luck the agent has in executing the action;
- How busy the agent is with other interruptions;
- The resources available to the agent;
- The lifetime of the agent.

8 How a Computer Agent should Handle a Request

What happens when an agent is requested to perform an action (now or in the future)? Given the previous discussion, it is important for a computer agent to consider a request carefully and look at whether that agent can comfortably accommodate the request or not. How should this work?

A request for an action from outside works basically the same as any other proattitude coming from inside the agent. The request is evaluated as to morality and utility, judged, and possibly chosen; the request is then deliberated upon, and possibly decided upon and scheduled; the request becomes an intention; if the agent has enough free time, execution of the action starts; if the agent is lucky, execution finishes successfully; if the agent is not lucky, the agent must keep trying as long as is necessary and/or reasonable.

However, there are some additional theoretical considerations that must be examined because the request comes from outside the agent. The main problem is the beneficiary. In an internally-generated proattitude such as a "want" or a "should", the beneficiary is some part of the agent, i.e. one of the demons in the agent's Society of Wants and Shoulds. Except in unusual circumstances, the agent will be very familiar with this demon, and will know and take into account its general reliability and the relationship of the agent with that demon. For instance, if one demon is always asking the agent to go to bed on time, the agent will tend to pay attention to that demon; whereas if another demon is always asking the agent to eat too much chocolate cake, the agent will tend to discount requests from that demon.

The problem lies in the relationship between the agent and the requestor. Does the requestor have power over the agent? Is the requestor a friend of the agent, or part of the agent's family/tribe? Could the requestor return the favor later? How powerful is the requestor? How powerful is the agent? What will happen if the agent considers the request, or turns it down immediately? These must all be considered by the agent. These factors can perhaps be modeled succinctly by one number, the *importance* of the agent.

The agent must consider its own *reputation* among other agents, and what its needs are in this area.

At the same time, the agent must consider the agent's *self-reputation* or *self-concept.* What does the agent like to think of itself as doing? Does the agent like to model itself as someone that can help other agents in a particular area, or as someone too important to help people? Does the agent think that it is a specialist in a particular area, and have an interest in maintaining that self-image successfully? These are all powerful motivators for humans.

Tied in with the concept of self-image is the concept of "conventional roles", normally associated strongly with employment. Why does a policeman chase robbers? Why does the man in the office send out applications to callers who request them? The quick answer is, "Because that's his job", or, in other words, his conventional role. But this answer is just begging the question, and it gives little useful information. The real answer is, "Because he wants to maintain his self-respect; and he believes that if he does his job well, then he will maintain his self-respect; and he believes that chasing robbers or sending out forms are actions that are instances of his doing his job well." This answer allows us to make strong predictions about what this agent will do and will commit to if asked.

9 Interpersonal Importance and Self-Respect

People have a need and a drive for self-respect.

People like being around other agents who make them feel important. If the agent does not make the person feel important, because the time of the person is a limited quantity, the person will seek out agents that do make the person feel important, and tend to ignore the previous agent.

An unsolved problem is why agents continue to need compliments and "strokes" to reinforce their self-importance and self-respect. One theory is that this belief decays. The agent's self-importance decays, and must be reinforced. Getting strokes (complements, communications) reinforces self-respect.

10 Transfer

One of the major problems associated with working with multiple agents is that there is, as of yet, no logic that is strong enough (or perhaps weak enough) to represent the most crucial occurrence in interactions: *transfer.* Without multiple agents, there is no transfer. Transfer is best illustrated by examining the concept of *transfer of belief.*

10.1 Transfer of Belief

Why do we usually believe what people say to us? No single-valued logic can represent this accurately–it involves an almost magic "leap of faith"–a transfer of belief. How does this work?

Agent Al states, "The cat is on the mat". From this, agent Beth can logically infer the following statements: (1) Al says, "The cat is on the mat". (2) Given straightforwardness, Al wants Beth to believe that the cat is on the mat. (3) Given sincerity, Al believes that the cat is on the mat.

However, it is impossible to get from (1), (2), or (3), to where we want to go, which is (4) Beth believes that the cat is on the mat, using logic alone. It is necessary to invoke an arbitrary assumption of "transfer of belief", along with (2), to get to (4).

Sometimes this transfer of belief will happen, and sometimes it won't. Whether it happens or not depends partially on the *opinions* by Beth of the *reliability* and *expertise* of Al in the matter of the statement, along with the *reasonableness* of the statement, and the implications and positive and negative consequences of believing the statement, disbelieving the statement, or suspending belief.

Compare the case of agent Al saying to agent Beth, straightforwardly and sincerely, "I come from the planet Jupiter." It is hopefully obvious that (1), (2), and (3) hold, but that (4) will [in almost all cases] not hold.

Thus, from (1) we can infer (2); from Beth's knowledge of (2), and an arbitrary assumption that transfer of belief takes effect, we can infer (4). There is no way to get from (2) to (4) directly without introducing an arbitrary assumption.

10.2 Transfer of Importance

A similar process holds for Transfer of Importance. If agent Al says, "X is VERY IMPORTANT to do", there is no logic that can prove that agent Beth will also think that "X is VERY IMPORTANT". Agent Beth will probably evaluate the situation, the beliefs of agent Al, the experience and knowledge of Al, the competence of agent Al, and Beth's "friendship" towards Al, and come to some conclusion as to the importance of X. However, there is no guarantee that this will be evaluated as "VERY IMPORTANT"; it could be that Beth comes to believe that "X is SORT OF important".

10.3 Transfer of Need

A similar process also holds for Transfer of Need. If agent Al says, "I VERY MUCH need you to do X", then agent Beth has to go from "Al believes that Al VERY MUCH needs me to do X" to "I believe that I SORT OF need to do X".

Note that it is possible for Beth to agree that X is important to be done, and yet not agree that Beth should be the one to do it. Beth could think that a specific someone else should do action X, or Beth could think that an unspecified someone else should do action X; in either case, Beth is too busy or does not think she is competent enough or simply does not feel like doing X herself.

11 How does a Request work?

Thus, when a request happens, there are typically a number of processes that go on.

First, the requestor Al evaluates the expected probability of the requested agent Beth being able to successfully perform the request. In addition, the value of the performed action is estimated. Finally, Al looks at the social, time, and resource costs of making the request.

Al decides whether it is in fact worth it to make the request right now, or whether Al should just keep silent (go without, try to do it himself, or ask somebody else).

If this decision comes up positive, Al makes the request.

Assuming that the communication channel is not significantly noisy, Beth receives the request. Beth evaluates the Transferred Importance and the Transferred Need.

This step can involve some negotiation if Beth is uncertain as to the exact status of Al. Al may attempt to portray himself as needing help more than he does; Beth may test this status.

If Beth decides that the request is of sufficient importance and sufficient need that Beth should try to do it if possible, Beth then evaluates her capability in doing the action and her expected chances of success, along with her expected duration of the action and her expected costs.

The evaluated action is compared against Beth's current time-table. If Beth is too busy doing something else, and/or the action looks too hard to do, then Beth may decline. Otherwise, Beth may accept the request and attempt to honor it.

Beth forms an internal commitment.

Beth should then attempt to communicate this to agent Al by making a verbal commitment.

12 A Generic Design for a Committing Computer Agent

A generic committing agent must therefore have a number of modules. There should be a pool of beliefs, and action recipes; a set of goals; a set of intended future actions; a set of current actions that are being processed at the present; a memory of past actions, for responding to queries; a system of communication channels in and out; an input messages queue; and a history of past messages, both in and out. In addition, the agent should have a pool of outstanding commitments to others, outstanding commitments from others, and a history of past commitments in and out. A simple demonstration system has been implemented on the Sequent computer.

13 Manners for Commitments

Similar to the Transfer problem, there is no way to *prove* how a real-world agent will act or react in a given circumstance. However, it is useful if agents have general rules that they try to follow when possible. For example, "If you actually make an internal commitment, then you should make a verbal commitment". These cannot be represented by any of the previous forms of logic:

Necessary Implication: A necessarily implies B. Obviously wrong.

Logical Implication: A implies B. Also wrong.

Computer "if" statements: If A, then B. Too strong.

Expert system or production system rules: If A, then do B. Also too strong.

A new ethical logic is required, with rules of the form, "If A, then I 'should' do B". Because this is a new kind of rule, which is not logically necessary and cannot be proven, I call it a "manner". If you know that an agent has a particular set of manners, then you can tell what that agent will probably do unless it gets into trouble. It is impossible to prove that that agent will behave in a particular way, but this is in fact a limitation due to the real-world nature of the problem, not a flaw in the logic. All you can say is that "s/he will do it if s/he will do it".

Left open for future research are the problems of defining what "should" means. There are at least four: the moral should, the ethical should, the habitual should, and the probably-likely should. These must be differentiated in a proper system so that false conclusions are not drawn.

Manners for commitments follow:

1. If you have no internal commitment, then you should not make a verbal commitment [Don't lie about what you commit to doing].
2. If you believe that you have made a verbal commitment, then you should make an internal commitment [Honor your word].[3]

[3]Note that this is slightly causally backwards, which might cause problems.

```
Marked text from Terminal 3 (TELNET) -- ATR-SQ on INTERNET [VT100]      Page 1

[5] <Initial lwp> (test1)
(setq office (make-agent "Office"))
(setq caller (make-agent "Caller"))
(request caller office '(send ,office ,caller form))
Request.  Caller is thinking about requesting Office to perform action Office sends Caller th
e form.
Does Caller believe it is worth it to make this request?  (T/NIL):t
Caller sends a message to Office requesting the action.

(MSG #<AGENT #xa03c69> #<AGENT #xa035c1> (REQUEST #<AGENT #xa03c69> #<AGENT #xa035c1> (INTEND
 #<AGENT #xa035c1> (SEND #<AGEN
T #xa035c1> #<AGENT #xa03c69> FORM)))).
CALLER: Office, I request Office intends that Office sends Caller the form.
Does Office accept the request?  (T/NIL):t

Office sends a message to Caller accepting the action:

(MSG #<AGENT #xa035c1> #<AGENT #xa03c69> (INTEND #<AGENT #xa035c1> (SEND #<AGENT #xa035c1> #<
AGENT #xa03c69> FORM))).
OFFICE: Caller, I intend that Office sends Caller the form.

(MSG #<AGENT #xa035c1> #<AGENT #xa03c69> (COMMIT #<AGENT #xa035c1> (SEND #<AGENT #xa035c1> #<
AGENT #xa03c69> FORM))).
OFFICE: I commit to Office sends Caller the form.

Office believes that Office is committed to the action:
(BEL #<AGENT #xa035c1> (COMMIT #<AGENT #xa035c1> (SEND #<AGENT #xa035c1> #<AGENT #xa03c69> FO
RM)))
Office believes that Office commits to Office sends Caller the form

Caller believes that Office is committed to the action:
(BEL #<AGENT #xa03c69> (COMMIT #<AGENT #xa035c1> (SEND #<AGENT #xa035c1> #<AGENT #xa03c69> FO
RM)))
Caller believes that Office commits to Office sends Caller the form

Can also talk about past messages.

(MSG #<AGENT #xa03c69> #<AGENT #xa035c1> (PREV-MSG (MSG #<AGENT #xa035c1> #<AGENT #xa03c69> (
COMMIT #<AGENT #xa035c1> (SEND
#<AGENT #xa035c1> #<AGENT #xa03c69> FORM))))).
CALLER: Office said to Caller that I commit to Office sends Caller the form.

#<AGENT #xa03c69>
[5] <Initial lwp>
```

Figure 1: Accepting the Request.

Marked text from Terminal 3 (TELNET) -- ATR-SQ on INTERNET [VT100]   Page 1

```
[5] <Initial lwp> (test1)
(setq office (make-agent "Office"))
(setq caller (make-agent "Caller"))
(request caller office '(send ,office ,caller form))
Request.  Caller is thinking about requesting Office to perform action Office sends Caller th
e form.
Does Caller believe it is worth it to make this request?  (T/NIL):t
Caller sends a message to Office requesting the action.

(MSG #<AGENT #xa09171> #<AGENT #xa08ac9> (REQUEST #<AGENT #xa09171> #<AGENT #xa08ac9> (INTEND
 #<AGENT #xa08ac9> (SEND #<AGEN
T #xa08ac9> #<AGENT #xa09171> FORM)))).
CALLER: Office, I request Office intends that Office sends Caller the form.
Does Office accept the request?  (T/NIL):nil

Office sends a message to Caller rejecting the action.

(MSG #<AGENT #xa08ac9> #<AGENT #xa09171> (NOT (INTEND #<AGENT #xa08ac9> (SEND #<AGENT #xa08ac
9> #<AGENT #xa09171> FORM)))).
OFFICE: Caller, it is not the case that Office intends that Office sends Caller the form.

Office believes that Office is not committed to the action:
(BEL #<AGENT #xa08ac9> (COMMIT #<AGENT #xa08ac9> (SEND #<AGENT #xa08ac9> #<AGENT #xa09171> FO
RM)))
Office believes that it is not the case that Office commits to Office sends Caller the form

Caller believes that Office is not committed to the action:
(BEL #<AGENT #xa09171> (COMMIT #<AGENT #xa08ac9> (SEND #<AGENT #xa08ac9> #<AGENT #xa09171> FO
RM)))
Caller believes that it is not the case that Office commits to Office sends Caller the form

Can also talk about past messages.

(MSG #<AGENT #xa09171> #<AGENT #xa08ac9> (PREV-MSG (MSG #<AGENT #xa08ac9> #<AGENT #xa09171> (
NOT (INTEND #<AGENT #xa08ac9> (
SEND #<AGENT #xa08ac9> #<AGENT #xa09171> FORM)))))).
CALLER: Office said to Caller that it is not the case that Office intends that Office sends C
aller the form.

#<AGENT #xa09171>
[5] <Initial lwp>
```

Figure 2: Rejecting the Request.

3. If you believe that another believes that you have made a verbal commitment, then either: you should believe that you have made a verbal commitment; or: you should explain to the other that you do not believe so. [Worry about others.]

4. If you make an internal commitment, then you should make a verbal commitment [Don't be closed-mouthed].

5. If you make an internal commitment, then you should believe that you have made one [Know what you think].[4]

6. If you (objectively) make a verbal commitment, then you should believe that you have made a verbal commitment [Know what you say].[5]

7. If you make an interagent commitment, then you should not change your mind later unless necessary [Keep your word].

8. If you make an interagent commitment and then have to break it, you should notify the committed-to agent as soon as possible [Be open].

All of these assume a benign environment with cooperative beneficial agents. Antagonistic situations will often have exactly the opposite manners.

14 Intention and Expectation

14.1 Intention

As has been discussed previously [Mye90a], intention is quite different from expectation. Expecting another agent or yourself to achieve something depends upon your evaluation of the agent's capabilities. An agent can intend to achieve something, and yet you can not expect that agent to have a good chance of achieving that intention.

14.2 Expectation

Expectation is uncertain and consists of evaluation of the uncertain probabilities of reaching one of several goals. This can be represented in a plan tree that is completely feasible by the Expectation E() operator, [defined over possible universes], assuming either nonrepeatable actions or full expansion of all possible repetitions.

In the case that a feasible tree is not complete, e.g. indefinite repetitions are allowed but have not been expanded yet by the agent thinking ahead into the future, expectation is more nebulous.

In the case of a potential plan tree where the future is not known yet, the agent must worry about what it will COME TO KNOW later. The agent may Expect to Come To Know stuff and so expect to uncertainly reach some goals, without concrete justification

[4]Note that this is not always true for humans.

[5]This is often not true. Also, this assumes an objective observer, which is usually not realistic.

(abstract justification?). For instance, in our data an Office worker will commit to sending a form to a Caller without knowing the caller's address; the Office reasonably expects to be able to learn the address later. If the agent does not expect to come to know information but must act with currently known actions, the agent will expect to Dead End, as an actually-executed potential tree with no feasible branches is worth nothing.

14.3 Changing One's Mind

14.3.1 Intentions. How do Intentions get revoked?

Intentions can be *postponed* if something more important comes along and the agent decides and chooses to take care of that, instead. (Intentions Have Grades, which are taken from the importance of obtaining the goal.)

An intention can "unintentionally"[6] be postponed if something important comes up, and the agent (due to lack of computation power and attention) does not notice that taking care of the important action entails preventing performance of the unimportant action.

If the intention cannot be postponed, it must be dropped in this case.

15 Promising

Promising, like all communication, assumes a known common language able to semantically communicate degrees and vagueness properly. This is often NOT a good assumption in the real world, and breaks down between cultures and between individuals. The key here is, what does a particular word or phrase of words MEAN? "I will CERTAINLY do it tomorrow." can represent many different things to different people: 95%-100%, or 10%-40%. If one person says that "you said F and you meant G" and the other person says "Yes, I said F and I meant H", then communication has not been effective.

15.1 Direct Promises

Given agents Al and Beth and action X,

Al PROMISES Beth that Al will do X

i.e., Al says to Beth, "I PROMISE to do X".

Note that this does not specify the time of execution. When will Al do X? Immediately? At some time in the future? Before the end of the universe? Before or after Beth dies? Before Al dies?

A modified promise may contain a time clause, e.g. "at time T" or "before time T". "After time T" is a problem, as this could be unverifiable by Beth.

[6]accidently

16 Uncertain Communication

Now assume that the agents come from different cultures, and have different semantics for the same syntax.

Case 1: The agents are not aware of this fact. In this case, the agents will each proceed to use their established communication patterns, and will most probably get into trouble. The fact that they are using different communication patterns will only surface after their POSSIBLE EXPECTATIONS have been seriously violated, and am inconsistency is noticed in the reasoning. Since the reasoning normally depends on the assumption that the other person is honest, there will be a real conflict to determine whether the RULES are incorrect, or whether the ASSUMPTIONS are incorrect.

Case 2: The agents are aware of this fact. In this case, an agent will try to Estimate a distribution of the probable actual meaning of a syntactic utterance. Estimations will depend on (1) the own agent's opinion of what the syntax means and what "everyone" thinks it "should" mean; (2) the agent's model, however incorrect, of how the other agent communicates. There will probably be a mixture of these. Smart agents will build wide distributions around these estimates. Dumb agents will forget that they are dealing with different communications and will revert to Case 1, especially under strong stimuli.

As an example, in the Maori culture when a strange person snarls at you, this is a greeting and should not be treated with fear. However, when a large wild animal snarls at you, this is a warning and should be treated with fear.[7]

Case 3: One agent is aware, one agent is not. The aware agent may attempt to accommodate the other agent.

17 Bargaining

How does bargaining work? In general, the situation is as follows:

X may have a WANT. Y may have a WANT. X may be able to fulfil Y's WANT. Y may be able to fulfil X's WANT. Agents are in general not altruistic, that is, they do not like to give out things for FREE.

If agents are altruistic, they may or may not be busy. If agents are not busy, they will do whatever the asker requests (definition of extreme altruism). Semi-altruistic agents will evaluate the request vs. their own needs and reject the request if it violates their needs (please give me \$100,000). If agents are busy, i.e. the agent is not idle and perhaps more than one request comes at a time, then the agent must have a processing strategy. 1) FIFO; 2) Scheduling Priorities–Noninterruptable; 3) Scheduling Priorities–Interrupt current action if request is serious enough; 4) LIFO Interrupting. Scheduling priorities can be based a) mostly on the own agent's opinion of the need; b) mostly on the requesting agent's opinion of the need. Requesting agents can be honest or dishonest, and can use a

[7]There is a story about miscommunication between the American occupying soldiers in Vietnam and the Vietnamese natives. The soldiers would sometimes threaten the natives. The natives would respond with nervous fear by laughing. However, laughter did not communicate fear to the soldiers, it communicated disrespect. This caused an escalation of the threat, often leading to violence.

commensurable scale or incommensurable scales of comparison. Honest agents will state their need as they perceive it. Dishonest agents may decide to cheat and state their need as more, which will perhaps improve their chances of receiving aid (assuming a linear scale). Amount of aid vs. amount of need: "I REALLY REALLY need $10 to buy food!!" "I need $1000 to build a house!!". The other agent may reject the request if the amount of aid required is too high. The asking agent must CONVINCE the granting agent to honor his request.

18 Areas for future research

- Causative and Third-Person Promises. X PROMISES Y to HAVE A DONE. X PROMISES Y that Z will do A.
- Negative Promises. X PROMISES Y that X WILL NOT DO A.
- Offers. Offers should have a time limit included if the offerer is wise. Note time is relative and may not include travel/communication time of the message.

  Offers can be one-way. "Anyone who reads this and sends in two box-tops will receive a prize." The company doesn't know who will read it.
- Contracts.
- Exchanges.
- Maintenance Plans and Goals.
- Convincing.

19 Multiple Agent Communication Conclusion

When multiple agents are communicating with each other in the real world, many problems must be considered. Many communication problems come from intentional agents, operating under limited resources, having to choose between conflicting possible courses of actions. An agent must thus choose whether to offer a commitment or not. Methods for determining how an agent should consider this have been detailed. The resulting theory allows a plan inference system to guess when an agent will accept a request and when an agent will reject a request.

In addition, there are many rules for working with commitments. These do not follow a strict logic but are based rather on "shoulds", and are therefore called *manners* rather than implications. A suggested set of manners for committing agents has been presented.

The results provide theory that will allow better understanding of human-to-human communication, and better design of human-to-machine and machine-to-machine communication.

20 Overview of Multiprocess Scheduling

In a parallel or distributed system with many processors (each of which can only run one task at a time), and many tasks to be run (where the number of tasks is greater than the number of processes), there is a need to select or *schedule* which tasks should be run on which processors. This need is especially important when the system is operating under conditions of limited resources–in other words, when the system needs to finish as soon as possible–and there is not enough time to run all of the tasks. Each task has a value, and the system must maximize the system's total value score before the system is terminated at the cutoff time. If the tasks are of uncertain duration or uncertain value, the problem becomes more complex. The scheduler must take responsibility for suspending tasks that are taking too long to complete, and for swapping them out and replacing them with new tasks.

There are many parameters that must be considered when designing a scheduling algorithm. These include: whether the tasks are disjoint single actions, disjoint sequences of actions, or overlapping trees of actions; whether the total cutoff duration of the system is fixed and known, unknown, or left up to the system; whether the tasks' values are non-significant or unit, known, or unknown; whether the tasks take a non-negligible amount of time to suspend, swap, and install or not; and finally, the scoring function of the system.

This portion of the paper discusses the theory behind scheduling tasks for multiple processes, and presents algorithms to deal with particular cases.

21 Introduction to Multiprocess Scheduling

This portion of the paper is concerned with the method of selecting tasks to be run on multiple processors. Why is such a problem important? In the ideal world, where there is an unlimited amount of time, the selection and ordering of tasks is not so important. A very simple algorithm can be followed: Start with the first task, and keep assigning tasks to open processors. When a processor finishes with a task, assign the next task on the list. Run all of the tasks until there are no more tasks left, and then finish. In this way one is guaranteed the maximum amount of information.

The problem with this approach is that in the real world, there is often not an unlimited amount of time. This means that some of the tasks will get finished, while other tasks will not be completed. It therefore becomes important to select which tasks will be run first, and which tasks may possibly not get to run at all, in order to maximize the quality or the quantity of the results.

This selection problem is called *scheduling.* The BEHOLDER[8] algorithms are designed particularly to schedule tasks that have durations that are uncertain—the scheduling process knows about how long each task takes to run (in a probabilistic manner), but cannot predict just exactly how much time it will cost.

[8]BEHOLDER stands for Beneficial Entity for Heuristically Ordering processes under Limited resources, and Decision-making for Execution in Real-time. A "beholder" is a mythical animal with seven eyes that can do multiple things in parallel. The algorithms presented here attempt to duplicate this capability.

22 Modeling the Problem

The problem is simulated by a model. The model has many parameters and assumptions. It is necessary to choose an algorithm for scheduling tasks, based on the model. The particular algorithm used for scheduling the tasks will depend strongly upon which parameter values are used to model the problem. Of course, the model is only an approximation; and if the model does not simulate the real process closely, the corresponding scheduling algorithm that is selected may not perform well for the real process. For this reason, it is important to explore the parameters that are used in modeling the problem.

The scheduling problem is defined by a number of components: the *processors*, the *scheduler*, the *jobs* composed of *tasks*, and the *deadline*. *Clients* submit jobs to the scheduler. Each job consists of one or more tasks to be performed. Depending on the model, these tasks may be unordered, or have to be done seqentially, etc. The scheduler examines the requested jobs and selects tasks that should be run. The scheduler assigns tasks to the processors, and waits until a processor has finished running a task. It then assigns another task to that processor. This continues until the deadline occurs, at which time all processing stops. At this point, the system is scored as to how well it performed, using a *scoring function*. The system attempts to score as high as possible.

Sometimes, while running tasks, the scheduler will decide that a task is taking too long to finish. In this case, the scheduler will *swap* the current task out of its processor, by *suspending* that task and *installing* a new task in the processor for the processor to run. Perhaps later on this new task will finish or take too long, and the old task will be swapped back in again and *resumed*.

It is assumed in this work that tasks can be suspended and resumed without serious penalty. If it is physically impossible to resume a task that has been suspended (due to the nature of the actual system), the scheduling algorithms must be significantly modified to take this factor into account.

It is assumed that there are a number of identical but separate processors. Each processor can run any onetask. A processor cannot run more than one task at a time. Different processors can run different tasks at the same time; this rules out SIMP machines such as Thinking Machine's CM-2. The scheduler knows the number of processors. Processors are accessible either by name or by location; the scheduler knows the names and locations of all processors. The processors may all be part of a multi-process machine, such as the Sequent Symmetry computer or Thinking Machine's CM-5, or they may be distributed on a network. The current work assumes that if distributed processors are used, then they are all the same "distance" away from the scheduler (measured by the amount of time necessary to effect a swap). This work does not explore the interesting cases in which the processors are different, e.g. have different processing speeds, different capabilities, or different distances from the scheduler requiring different loading times.

The BEHOLDER scheduler discussed in this paper is assumed to be separate from the processors and the tasks to be run. This is known as a Type 1 scheduler, and can be implemented by either building special hardware for the scheduler, or by providing the scheduler with a dedicated processor. In the implementation, the scheduler shares a dedicated processor with the `pformat` handler; this reduces the number of processors from 15 to 14, but results in a clean implementation.

Because the scheduler runs relatively quickly and can predict when it will need to wake up, it is also possible to run a different type of scheduler. Instead of having a separate dedicated processor, a Type 1.5 scheduler uses one of the same processors as the application in order to run the scheduler. The scheduler is run at a higher priority, however, and can interrupt an application task immediately at any point. The Type 1.5 scheduler runs, assigns tasks to the open processors plus one task left over for its own processor when the scheduler is finished, sets an alarm clock, and then puts itself to sleep and starts the substitute application task. When the alarm clock goes off, or if a new application job is submitted, or if one of the old tasks finishes, the scheduler wakes up, preempts the currently running application-task process (which should be running a task with the lowest priority), makes scheduling changes as necessary, and then goes back to sleep. Note that this cannot easily be implemented on the Sequent, because there is currently no facility to suspend a low-priority task in favor of a high-priority task that is not running yet.

It is theoretically much more difficult to build an optimal Type 2 scheduler, where the scheduler itself is one of the job tasks that the scheduler assigns to processors. To be done properly, this requires research into the value of running the scheduler and perhaps deciding to change the current tasks assignments, as opposed to the value of continuing with the current task assignments as they are. The expectation of such a value can be determined by guessing how much more useful changing the current assignments would be versus leaving them the same, which can be roughly estimated by subtracting the value of the highest new job from the value of the lowest currently running job. However, such research has not been performed yet.

It is assumed that the tasks being scheduled are relatively long in duration. Scheduling costs overhead time. If the tasks take much longer to run than the scheduler takes to decide on a task, then the overhead of time spent scheduling is a small percentage of the time actually spent running applications by the entire system. If, however, the tasks take a very short time to run, then the scheduler can fall behind and become a system bottleneck.

In the basic versions explored in this report, each task can be handled by only one processor.[9]

23 The MF and IMF Transforms

The scheduling algorithms are based on the Mean-time-to-Finish and the Inverse Mean-time-to-Finish transformations. Since these are central to the understanding of the scheduling algorithms, they are derived and explained separately here.

The basic input to the system consists of an approximate value estimate for each job and/or each task, and a probabilistic duration estimate for each task. Without these, the system does not have enough information to make informed decisions on the scheduling problem, and can only guess as to which job should go first.

The most simple duration estimate is in the form of a distribution plot of previously observed duration times vs. counts. See Figure 3. This is easy to obtain from actual

[9]It is also possible to provide algorithms for the cases in which multiple processors can work on aspects of one task in parallel. However, these will not be discussed here.

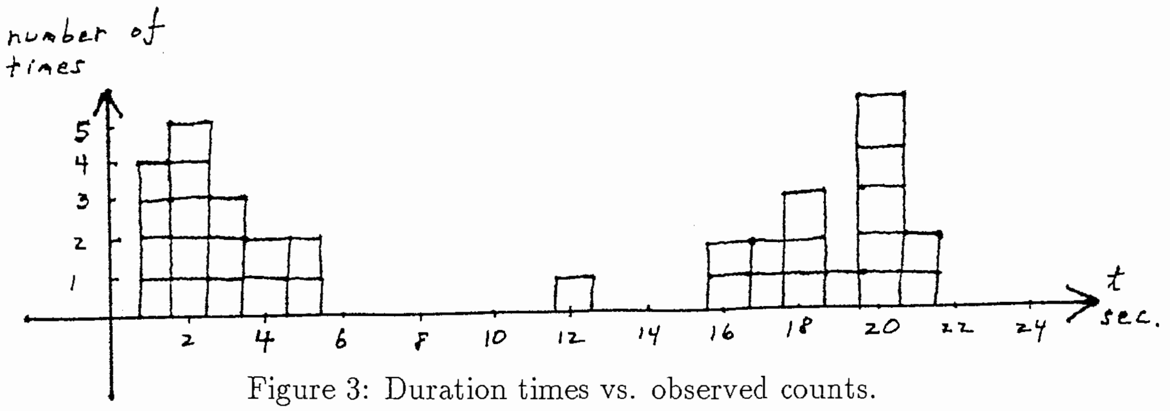

Figure 3: Duration times vs. observed counts.

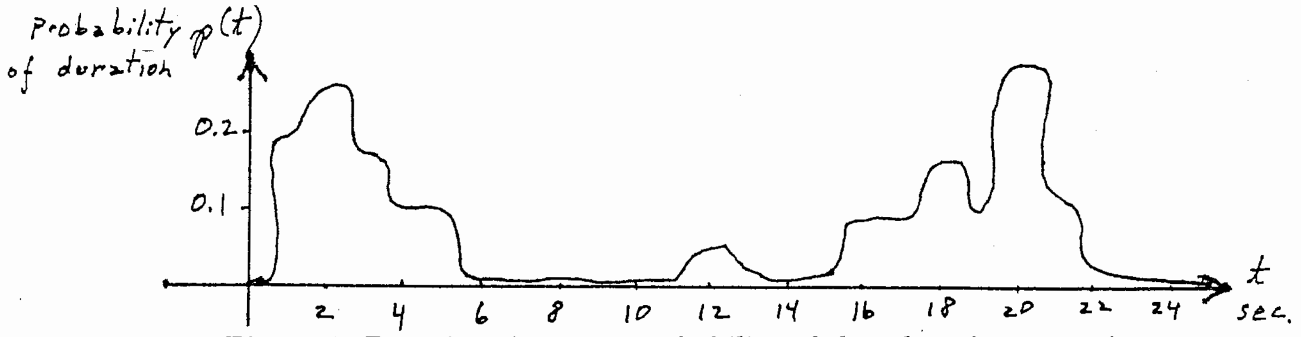

Figure 4: Duration times vs. probability of that duration occurring.

experiments, by timing how long the task takes to run, and then accumulating that result in a count array.

It is then necessary to construct a probabilistic estimate of the duration of the task. To be more specific, we need an estimate of the probability of each task duration occurring the next time that the task is run. The plot is duration time vs. probability. See Figure 4. This graph can be obtained by normalizing the previous graph (dividing by its area). It is important to add an extremely small probability of not terminating at all until the longest time possible, to acknowledge the possibility of real-world errors in executing the task. It is also important to realize that the probability on the Y-axis is the believed probability of the task taking a particular duration to run, given that the task *has not started yet.*

Now, examining this graph, it is possible to compute an estimate of the Mean (average, or expected) time for the task to Finish, given that the task has not started yet. This is done by the normal calculus formula for finding the center of mass of a given area. The integral is done starting from time zero, and ignoring everything before that (since it is impossible to finish in the past something that has not been started yet). The area under the curve is weighted by the duration and integrated; this is then divided by the total area under the curve. The formula is thus

$$MF(0) = \frac{\int_0^\infty t \cdot p(t)\, dt}{\int_0^\infty p(t)\, dt} \tag{1}$$

This finds the average time to finish, given that we have not started the task yet and are still looking at it. See Figure 5.

Now, what happens if the task has been going on for 10 seconds so far, but it is known to not be finished yet? How much more longer will the task be expected to run? In this case, we have to ignore the probability mass from 0 to 10 seconds, and perform the previous

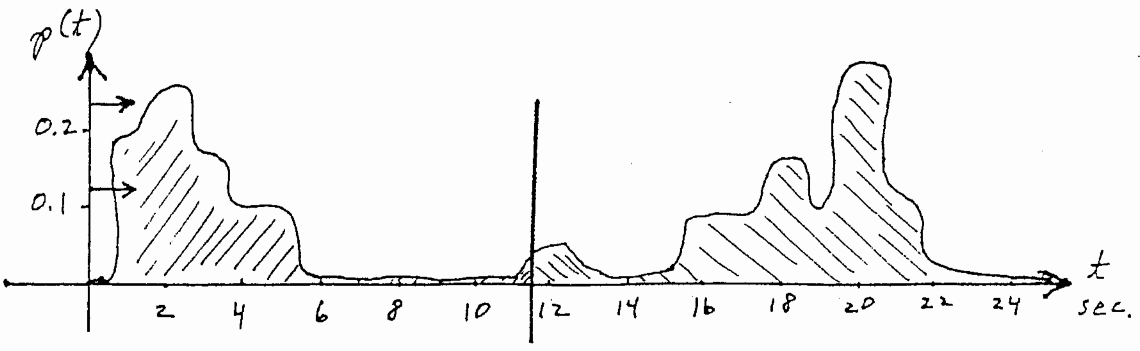

Figure 5: Mean Time for the Entire Task to Finish.

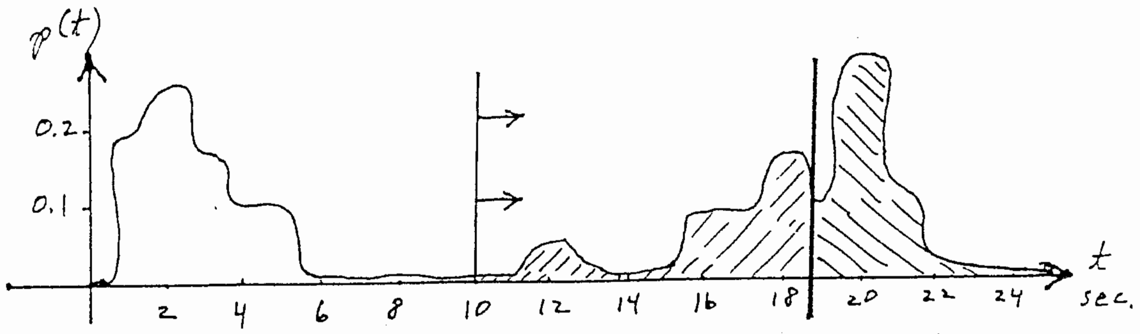

Figure 6: Mean Time Remaining to Finish, Given 10 Seconds of Run-Time

calculation again. But this time, the mass is integrated starting at 10 seconds, and we get a different answer. The equation looks like:

$$MF(10) = \frac{\int_{10}^{\infty} (t - 10) \cdot p(t)\, dt}{\int_{10}^{\infty} p(t)\, dt} \tag{2}$$

See Figure 6. This finds the average additional time required to finish, given that we have started the task and have already consumed 10 seconds running the task.

It is hopefully obvious at this point that we can perform this calculation for any arbitrary time T, and come up with the expected time left to finish at any one point in time, given that we know that the task has been running and has not yet finished at that point. This formula is

$$MF(T) = \frac{\int_{T}^{\infty} (t - T) \cdot p(t)\, dt}{\int_{T}^{\infty} p(t)\, dt} \tag{3}$$

See Figure 7.

This formula in fact defines a *transform*, from $p(t)$ into $MF(T)$. We can now compute this formula for all T's from 0 to ∞, and then graph the results. The new plot is the *Mean-time-to-Finish Distribution* for this task. The axes are duration taken so far vs. expected

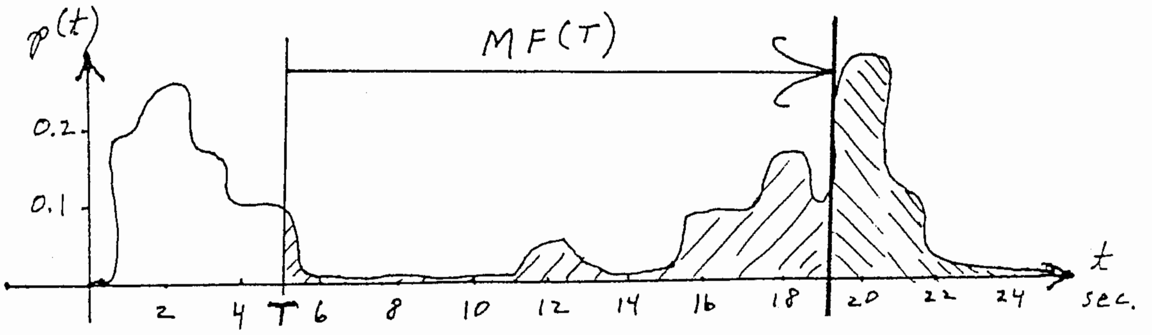

Figure 7: Mean Time Remaining to Finish, Given T Seconds of Run-Time

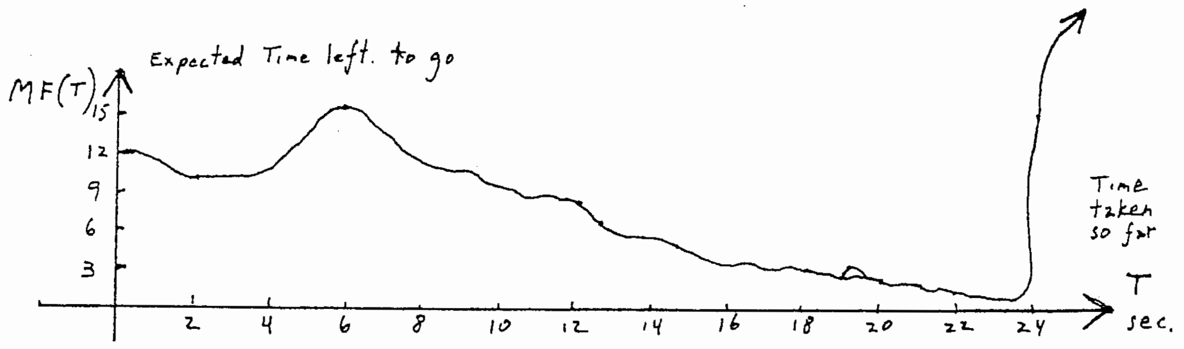

Figure 8: Mean Time Remaining to Finish, as a Function of Time Consumed So Far

(average) additional duration required to finish the task, given that we know that the task has not finished running yet. See Figure 8. This transform is used as is in some algorithms.

Finally, we can take the inverse of this function, using equation:

$$IMF(T) = \frac{\int_T^\infty p(t)\, dt}{\int_T^\infty (t - T) \cdot p(t)\, dt} \tag{4}$$

This defines the *Inverse Mean-time-to-Finish.* This function is used in most of the scheduling algorithms.

Sometimes this function will be multiplied by the expected value of the finished task, V. In this case, we call the resulting distribution the *Valued Inverse Mean-time-to-Finish VIMF.*

23.1 Comments on the MF Transform

If a particular subtask is known with certainty to have a duration of x seconds, then the MF(t) transform graph will be a straight line from (0,x) to (x,0).

It is not necessary when computing the MF transform that the $p(t)$ function be normalized. In fact, in the implementation on the Sequent, $p(t)$ is left as an accumulation of the duration counts. It is also not necessary that $p(t)$ be stored with linear quantization, as long as the actual times (and not the quantized index) are used to multiply with $p(t)$ in the integral. The system uses a duration vector that is stored with logarithmic quantization, which covers time from 100 microseconds through 12 days using 101 entries.

There is a problem in that the MR distribution is undefined after large t entries, when the $p(t)$ distribution is zero from the current given t to infinity. This mathematics corresponds to answering the question, "How long do you expect the routine to continue running if enough time has elapsed so that the routine could not possibly be running any more, and yet the routine is still running?". In order to avoid singularities in the mathematics, it is necessary to introduce an ϵ possibility of a duration of $t = \infty$. In practical terms, this translates into introducing an adjustment count of 1 observation at the largest duration quantization (e.g., $t = 12$ days in the system).

As long as the duration distribution is unimodal, the MF distribution will basically be monotonically decreasing. However, when the duration distribution is multi-modal, the MF distribution can increase. For instance, say that you have a parsing routine which takes either about two seconds or about twenty seconds to execute (perhaps because it detects parse failures in an early manner). At the beginning, you expect it to take an average of 11 seconds to complete. If, after four seconds, the routine has still not completed, you now expect it take an average of 16 more seconds to complete, longer than before.

24 The BEHOLDER system scheduler

Given N processors and an expected duration distribution for each task, schedule the tasks.

24.1 The Beholder-1 Scheduling Algorithm

The Beholder-1 algorithm assumes that there are a number of separate tasks to be scheduled. Each task has the *same* expected completion score, or, equivalently, all the tasks' expected completion scores are completely unknown. The system evaluation function is the *sum* of the scores of all of the fully completed tasks. The duration time allowed to the system is *completely unknown.*

The Beholder-1 Scheduling Algorithm uses the raw MF transforms for scheduling. The tasks are sorted by their MF(0) score. The top N tasks are selected and run; the (N+1)th task is "on deck" and will be the next task run. The MF(0) of the "on deck" task is called the "trigger level". The trigger level is compared (ahead of time) against the MF distributions of the N running tasks. When one of the distributions becomes higher than the trigger level, the slow task must be swapped out and the on-deck task swapped in; the evaluation process is then repeated, and a new on-deck task chosen.[10]

24.2 The Beholder-2 Scheduling Algorithm

The Beholder-2 algorithm deals with tasks with differing expected completion-scores. The MF transform is modified to be equal to the score divided by the mean-time-to-finish. This new valued inverted MF (VIMF) is then maximized to obtain the ranking.

The IMF transform has the quality of making very short tasks of little value equivalent to very long tasks of great value. It again assumes that the system duration is completely unknown.

It is necessary for the IMF that all tasks have a non-zero duration (otherwise they have been completed already).

It does not matter whether the expected score is exact or a distribution. Since the VIMF only deals with expected value, and does not represent the spread of uncertainty, there is no need to represent the distribution of possible scores.

[10]Since the MF transformation only deals with the expected time to finish, no second-order uncertainty needs to be represented. It does not matter whether the duration distribution is represented by a simple probabilistic distribution or by a second-order uncertain distribution; both are entirely equivalent.

24.3 The Beholder-3 Scheduling Algorithm

The Beholder-3 algorithm deals with sequential tasks consisting of disjoint subtasks that must be completed in a given order. In this case, the duration distributions of the tasks are convolved, and the resulting total duration distribution is used for the IMF transform of the whole sequence. The expected value of the entire sequence's completion is used. When the first subtask is finished, the IMF of the sequence switches over to the IMF of the second subtask, etc.

For example, say that subtasks A and B make up the sequence A;B. The ranking distribution is established by taking the duration of A, convolving it (using addition) with the duration of B, and taking the VIMF of the resulting convolution, using the value of the entire sequence A;B's completion as the scaling value.

Note that, although the initial value (MF(0)) of the transform of the convolution is equal to the sum of the transforms of the addends, unfortunately the MF distribution of the convolution is not equal to the sum of the MF distributions of the subsequences. It is easy to see this by noting that the MF distribution for the total sequence will in general be defined for a distance twice as long as either of the two subsequence's MF distributions. It is thus necessary to compute the entire convolution before taking the MF transform.

24.4 The Beholder-4 Scheduling Algorithm

The Beholder-4 algorithm deals with forking tasks consisting of sequences of subtasks that must be completed in a given order, where some of the initial subtasks are identical between sequences. In this case, the duration distributions of all of the task sequences must be computed. However, the highest scoring subtask sequence in a fork *shadows* task sequences with similar initial subtasks; these are put on a list behind the shadowing task sequence and not allowed to compete for system resources. When the initial subtask is finished, the shadowed tasks are then expanded and allowed to compete. As long as the final sequence value does not change and the initial subtask has not yet been completed, the VIMF transform that is highest at time $t = 0$ will continue to be the highest transform of the set of shadowed sequences.

24.5 The Beholder-10 Scheduling Algorithm

The Beholder-10 algorithm assumes that there are a number of separate tasks to be scheduled, with unit scores. The system evaluation function is the *sum* of the scores of all of the fully completed tasks. The system duration cutoff time is a known constant.

This problem is isomorphic to the bin-packing problem, with probabilistically-sized objects. The following is believed to be an exact solution to this problem.

The expected value of executing a task is equal to the expected value of the completed task (a unit value in this case) divided by the expected duration of the task (the current MF transform value) times the probability that the task will be completed before cutoff (unit if the cutoff is "far away"). The probability of completion can be computed by integrating the duration distribution between the current time and the cutoff time, and dividing by the integral of the duration distribution between the current time and positive infinity. If the current time is zero, this can also be found by subtracting from one the integral of the duration distribution from the cutoff time to infinity, divided by the mass of the entire distribution (which should be unit in a normalized distribution). Note that in general the integral of the duration from zero to the cutoff may *not* be used if the current time is not zero.

24.6 The Beholder-11 Scheduling Algorithm

The Beholder-11 algorithm assumes that there are a number of separate tasks to be scheduled, with various known scores. The system evaluation function is the *sum* of the scores of all of the fully completed tasks. The system duration cutoff time is a known constant.

This problem is isomorphic to the bin-packing problem, with probabilistically-sized objects with known values. The following is probably an exact solution to this problem, although it may be approximately optimal.

The expected value of executing a task is equal to the expected value of the completed task divided by the expected duration of the task (the current MF transform value) times the probability that the task will be completed before cutoff (unit if the cutoff is "far away"). The probability of completion can be computed by integrating the duration distribution between the current time and the cutoff time, and dividing by the integral of the duration distribution between the current time and positive infinity. If the current time is zero, this can also be found by subtracting from one the integral of the duration distribution from the cutoff time to infinity, divided by the mass of the entire distribution (which should be unit in a normalized distribution). Note that in general the integral of the duration from zero to the cutoff may *not* be used if the current time is not zero.

24.7 The Beholder-15 Scheduling Algorithm

The Beholder-15 algorithm assumes that there are a number of separate tasks to be scheduled, with unit value. The system evaluation function is the *sum* of the scores of all of the fully completed tasks. The system duration cutoff time is a random variable with a known probabilistic distribution.

In this case, the system has to evaluate the probability that a given scheduled task will finish and will contribute towards the scoring. If a task waiting to be scheduled may take longer than the cutoff, then it can't be counted. The practical effect is to change the MF integral from an upper-unbounded integral into an upper-bounded one with a probabilistic weighting.

24.8 The Beholder-20 Scheduling Algorithm

The Beholder-20 algorithm assumes that there are a number of separate tasks to be scheduled. The system evaluation function is the *sum* of the scores of all of the fully completed tasks. The system duration cutoff time is self-determined by the system, using a decreasing utility curve.

The system must look at the utility of doing further processing, versus the utility of stopping now with all of the tasks that have been performed so far. This is a decision that is similar to a meta-decision, and is based on the expected value of performing further tasks.

24.9 The Beholder-30 Scheduling Algorithm

The Beholder-30 algorithm assumes that there are a number of separate, sequential, or forked tasks to be scheduled. The system evaluation function is the *maximum* of the scores of all of the fully completed tasks. The duration time allowed to the system is *completely unknown*, and is assumed to be "large". The task values are known precisely.

The expected value of a task is equal to the probability that the task is finished by the unknown cutoff, multiplied by the (known) value of the task. In this case, there is no need to compute the probability of completion, since the cutoff time is assumed to be "large". Thus, the best scheduling algorithm is to use only the values for ranking, and assume that any one sequence that is started will be completed before cutoff time.

If the tasks are forked, the best initial subtask shadows execution of the other sequences having the same initial subtask. When the initial subtask is finished executing, the shadowed sequences are expanded and the ranking must be evaluated again.

If the task values do not change, it is sufficient to execute only one sequence in this case, as the task values are known precisely from the beginning.

24.10 The Beholder-31 Scheduling Algorithm

The Beholder-31 algorithm assumes that there are a number of separate, sequential, or forked tasks to be scheduled. The system evaluation function is the *maximum* of the scores of all of the fully completed tasks. The system cutoff duration is known precisely.

In this case, the expected value of starting a task is equal to the expected value of completing the task, times the probability that the task will in fact be completed by the system cutoff point. This can be found by integrating the duration distribution from the current time (0?) to the cutoff time, divided by the integral of the distribution from the current time to infinite time (which will be 1 for normalized distributions, if the current time is 0). The tasks are ranked using this value. For sequential tasks, the convoluted total-duration distribution is used to rank the sequence. The value of a sequence must be recomputed after a subtask is finished.

All of the sequences of a fork are computed and used; the highest value then shadows the others, but the shadowed sequences must be expanded and the system must be reranked when the initial subsequence is finished. These values must be recomputed for the remainders of the sequences, since the actual initial subtask finishing time has become definite, not a stochastic range.

24.11 The Beholder-32 Scheduling Algorithm

The Beholder-32 algorithm assumes that there are a number of separate, sequential, or forked tasks to be scheduled. The system evaluation function is the *maximum* of the scores of all of the fully completed tasks. The system duration is known approximately, with a probability distribution on an anticipated hard cutoff.

The probability that a given task will finish is found by working with the cutoff duration distribution. The proportional area behind each cutoff is determined, and then multiplied by the probability of cutoff at that point. The results are summed, to determine the weighted probability of finishing before cutoff. This is then multiplied by the value to determine the ranking index.

A sequence uses the convoluted total-duration distribution in this calculation, and must be recomputed after a subtask has been finished.

A forked sequence requires evaluation of all of the separate sequences. The best sequence in the fork shadows the others. The system's ranking must be evaluated again when the initial subtask is finished.

24.12 The Beholder-33 Scheduling Algorithm

The Beholder-33 algorithm assumes that there are a number of separate, sequential, or forked tasks to be scheduled. The system evaluation function is the *maximum* of the scores of all of the fully completed tasks. The system duration is self-determined by the system.

It is important both to obtain good results and to work quickly. The task ranking is thus determined by the expected value squared, times the probability that this task will finish in time, divided by the current mean time to finish. Since the system should not start tasks that it does not believe it will not finish, the probability that a given task will finish is approximated by one, and this term is ignored.

A sequence uses the convoluted total-duration distribution in this calculation, and must be recomputed after a subtask has been finished.

A forked sequence requires evaluation of all of the separate sequences. The best sequence in the fork shadows the others. The system's ranking must be evaluated again when the initial subtask is finished.

25 Discussion

In running a current-technology utterance speech-to-speech translation job, there is usually only one job. However, this job has a number of tasks, in the form of a forked tree. Each branch in the tree represents one thread of translation, from speech recognition, through parsing, understanding, transfer, and language generation. The results are sent to speech generation. The tree forks because each process is capable of generating more than one possible output for each input. However, many fork threads will not be viable. The system wants to come up with the one thread that has a maximum score, and wants to ignore all the others, within a reasonable amount of time. If there are adequate parallel system resources, it makes sense to explore many different fork threads, in the hope of finding the best one.

Often the true value of a task will not be determined until after the task has been finished. This requires the scheduler to keep trying to do the best that it can.

26 Multiprocess Scheduling Conclusion

This paper has presented the theory behind the BEHOLDER family of algorithms for scheduling tasks of uncertain duration under limited time and computational resources. The particular algorithm that should be used depends upon the particular class of scheduling problem to be solved. The user may have to extend these algorithms to solve new kinds of tasks. Hopefully the theory behind the scheduling method has been explained enough so that this can be done in a straighforward manner.

References

[AI89] Hidekazu Arita and Hitoshi Iida. Tri-layered plan recognition model for dialogue machine translation. Technical Report TR-1-0067, ATR Interpreting Telephony Research Laboratories, Kyoto, Japan, 1989. (in Japanese).

[BP83] Jon Barwise and John Perry. *Situations and Attitudes.* The MIT Press, Cambridge, Mass., 1983.

[Bra87] Michael E. Bratman. *Intention, Plans, and Practical Reason.* Harvard Univ. Press, Cambridge, MA, 1987.

[CL86] Philip R. Cohen and Hector J. Levesque. Persistence, intention, and commitment. In Michael P. Georgeff and Amy L. Lansky, editors, *Reasoning about Actions & Plans: Proceedings of the 1986 Workshop*, Los Altos, CA, 1986. Morgan Kaufmann Publishers, Inc. Also report CSLI-87-88.

[CL87] Philip R. Cohen and Hector J. Levesque. Intention = choice + commitment. In *AAAI'87*, pages 410–415, Seattle, WA, 1987.

[Den87] Daniel C. Dennett. *The Intentional Stance.* The MIT Press, Cambridge, Mass., 1987.

[dK86] Johan de Kleer. An assumption-based tms. *Artificial Intelligence*, 28(2):127–162, March 1986.

[Gol70] Alvin I. Goldman. *A Theory of Human Action.* Prentice-Hall Inc., Englewood Cliffs, NJ, 1970.

[MT90] John K. Myers and Takashi Toyoshima. Known current problems in automatic interpretation: Challenges for language understanding. Technical Report TR-I-0128, ATR Interpreting Telephony Research Laboratories, Kyoto, Japan, January 1990.

[Mye88a] John K. Myers. Action parameter determination under fallible execution. ATR International, Kyoto, Japan, December 1988.

[Mye88b] John K. Myers. Fallible execution. ATR International, Kyoto, Japan, December 1988.

[Mye88c] John K. Myers. The necessity of intentions under fallible execution. ATR International, Kyoto, Japan, December 1988.

[Mye89a] John K. Myers. An assumption-based plan inference system for conversation understanding. In *WGNL Meeting of the IPSJ*, pages 73–80, Okinawa, Japan, June 1989.

[Mye89b] John K. Myers. The atms manual (version 1.1). Technical Report TR-1-0074, ATR Interpreting Telephony Research Laboratories, Kyoto, Japan, February 1989.

[Mye89c] John K. Myers. Np: An assumption-based feature-structure plan inference system. Technical report, ATR Interpreting Telephony Research Laboratories, Kyoto, Japan, December 1989.

[Mye90a] John K. Myers. A design for a disambiguation-based dialog understanding system. Technical Report TR-I-0189, ATR Interpreting Telephony Research Laboratories, Kyoto, Japan, November 1990.

[Mye90b] John K. Myers. The fs-lf manual, version 1.2. Technical Report TR-I-0162, ATR Interpreting Telephony Research Laboratories, Kyoto, Japan, April 1990.

[Mye90c] John K. Myers. The meanings of ability utterances with applications to dialog understanding. Technical report, ATR Interpreting Telephony Research Laboratories, Kyoto, Japan, January 1990.

[Mye90d] John K. Myers. Methods for handling spoken interruptions for an interpreting telephone. In *IEICE Technical Report NLC-90*, pages 17–24, Tokyo, Japan, May 1990.

[Mye90e] John K. Myers. Np: An atms-based plan inference system that uses feature structures. In *Meeting of the IPSJ*, pages 438–439, Tokyo, Japan, March 1990.

[Mye90f] John K. Myers. A project report on np: An assumption-based nl plan inference system that uses feature structures. In *COLING-90*, volume 3, pages 428–430, Helsinki, Finland, August 1990.

[Mye90g] John K. Myers. Research on understanding at atr, with recommendations for multran (or, how to build intentional autonomous agents). Lecture notes., September 1990.

[Mye91] John K. Myers. Plan inference with probabilistic-outcome actions. In *Conf. Proc. Information Processing Society of Japan*, volume 3, pages 168–169, Tokyo, Japan, March 1991.

[Mye92a] John K. Myers. Bsure: A believed situation and uncertain-action representation environment. In *COLING-92*, Nantes, France, July 1992.

[Mye92b] John K. Myers. An introduction to planning and meta-decision-making with uncertain nondeterministic actions using 2nd-order probabilities. In *First International Conference on AI Planning Systems*, College Park, Maryland, June 1992.

[Mye92c] John K. Myers. A theory of intentional action under uncertainty as applied to physical and communicative actions. In Hisato Kobayashi, editor, *Proceedings of the IEEE Workshop on Robot and Human Communication: RO-MAN '92*, pages 305–310, Hosei University, Tokyo, September 1992.

[OCP90] Sharon L. Oviatt, Philip R. Cohen, and Ann Podlozny. Spoken language in interpreted telephone dialogues. Technical Report AIC-496, SRI International, Menlo Park, CA, 1990.

[Ovi88] Sharon L. Oviatt. Management of miscommunications: Toward a system for automatic telephone interpretation of japanese-english dialogues. Technical Report Tech note 438, SRI International AI Center, Menlo Park, CA, 1988.

[Suc87] Lucy A. Suchman. *Plans and Situated Actions.* Cambridge University Press, Cambridge, Mass., 1987.