

Abstract

GARLICK, JOANNA SHANNON. The Inclusion of Reference Seeds in Generated Dialogue (Under the direction of Michael Young).

In the course of natural dialogue, humans are able to effectively use previous conversation. This is true for many types of dialogue, including instructional or task-related conversation. When human speakers effectively leverage past conversation for illustrative or comparative purposes, their hearers reap the benefits of increased dialogue coherency and are better able to assimilate knowledge. The practice of utilizing past utterances in current conversation is called referencing. It is believed that if computer agents could utilize referencing the way humans do, these agents would produce more natural-sounding and effective dialogue.

For this reason, the natural language community has conducted a fair amount of research into referencing. While most of the past and current research focuses on identifying opportunities to refer backward and how to do so, this research focuses on the novel concept of a reference seed. We introduce the term seed to refer to a piece of information opportunistically included in generated dialogue with the intent that it be referred back to in the future. We show how seeds and references can be implemented using discourse planning operators and identify some general rules for when to do so. We also discuss a test system implemented to showcase the potential of the concept and report the encouraging results of an informal user study we have conducted.

THE INCLUSION OF REFERENCE SEEDS IN GENERATED DIALOGUE

by

JOANNA SHANNON GARLICK

A thesis submitted to the graduate faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

COMPUTER SCIENCE

Raleigh

2001

APPROVED BY:

Chair of Advisory Committee

Dedication

This work is dedicated to my grandmother, Kathleen Woodson Garlick. She instilled in me a love of learning at an early age, and she taught me the virtues of curiosity, perseverance, and compassion. While she lived she was an example of how faith in God, a good sense of humor, and a genuine interest in humanity can positively affect the lives of others. She will always be one of my heroes.

Biography

Shannon was born in Lynchburg, VA in 1978 and lived there until she was 7. She spent the next 11 years in Raleigh, NC, with her mother, father, and sister. She had a great family life and enjoyed her years in North Carolina, so she decided to stay there for college. She attended North Carolina State University, originally double majoring in computer science and communications with a theatre concentration. However, she was forced to drop the theatre major after her sophomore year due to time and energy limitations.

She spent the next year trying to find a way to combine her two loves of theatre and computers, and eventually found a home in the Liquid Narrative Group, a research group that applied principles of narrative and traditional media to digital interactive media. She spent her senior year working for Dr. Michael Young on Liquid Narrative projects. She graduated in May of 2000 from NCSU with a degree in Computer Science. Shannon continued to work for the LNG in pursuit of a Master's degree through the summer of 2001. After obtaining her MS, she will move to Portland, Oregon to work for Intel Corp. She intends to go back to grad school for a Ph.D. eventually.

Acknowledgements

“There is no such thing as a 'self-made' man. We are made up of thousands of others. Everyone who has ever done a kind deed for us, or spoken one word of encouragement to us, has entered into the make-up of our character and of our thoughts, as well as our success.” --George Matthew Adams

I have learned that there is no such thing as a ‘self-made’ woman, either. God has not only guided my path, but he has blessed me with many people to help me along the way. I am eternally grateful to all of the people mentioned here, as well as many others that go unmentioned but not unremembered.

First, thanks to my family - mom, dad, and Amanda - for your undying support and unconditional love. All of the little things you guys do for me do not go unnoticed! I know I would never be able to manage life without your help and encouragement. Your continual faith in me is a constant affirmation when I am lacking confidence.

Next, thanks to my friends for being there for me in more ways than I can count. Besides helping me academically, transportationally, and financially, you make me laugh and help me to relax. Thanks especially to all of you who pestered me about my thesis this summer. It would be impossible to list everyone, but some friends deserve particular mention for tolerating me in excessive amounts: Jen, Mia, Kelli, Sharon, Blake, James, and Luis. To the rest of you, you know who you are.

I am also extremely lucky to have found such a wonderful advisor in Dr. Michael Young. You were right about my affinity for higher education. Thanks for being so patient and for helping me battle my academic insecurities. I also need to thank all of the Mimesis lab-dwellers for making the summer so much fun – without such a cool lab to work in, writing this thesis could have been a real drag.

I greatly appreciate the feedback of Dr. Young, mom and dad, and CJ, all of whom offered valuable comments during the writing of this thesis. I am very grateful for the cooperation and guidance of my committee members, Dr. Robert St. Amant and Dr. James Lester. I also offer many thanks to everyone who participated in the user study.

Last, I want to acknowledge some of my role models. (The people mentioned above, especially parents and advisors, are role models also.) These women have served as examples to me of how to live life with strength, sensitivity, courage, and character: Doug Haas, Carol Miller, Toni Thorpe, and Sarah Monast.

Table of Contents

1. INTRODUCTION.....	1
2. RELATED WORK	5
2.1 DISCOURSE PLANNING.....	5
2.2 THE INTENTIONAL STRUCTURE OF DISCOURSE	8
2.3 BACKWARDS REFERENCES	12
2.4 REDUNDANT UTTERANCES	13
3. IMPLEMENTATION	15
3.1 TERMINOLOGY.....	15
3.2 THE LONGBOW DISCOURSE PLANNER	19
3.3 THE ALGORITHM	23
3.4 THE TESTBED.....	29
3.5 AN EXAMPLE	31
5. APPLICATIONS	48
6. FUTURE WORK.....	51
7. CONCLUSIONS	56
8. BIBLIOGRAPHY	58
APPENDIX A – USER STUDY MATERIALS	60
APPENDIX B – USER STUDY RESULTS.....	63

1. Introduction

Communication is one of the most natural and also one of the most complex aspects of humanity. While many claim that we are born with an inherent ability to communicate, negotiation, debate, instruction, and persuasion, for example, are all skills that must be studied and learned. One of the many open research goals in the area of computer science and linguistics is the goal of making computers communicate with people in the same way that people communicate with each other. Computer interfaces designed to work in this manner would have many obvious advantages for the use of computers as an information medium – users would be able to obtain information quicker and understand it better and computer agents would be able to respond appropriately to users of different levels, for example. Computer agents might one day be able to carry on conversations with users as if they were friends. The goal of getting computers to generate a free-form dialogue that would sustain the long-term interest of a human is still a long way from being attained, but researchers have made progress in getting computers to practice some of the more structured and constrained forms of communication mentioned above.

Task-oriented dialogue is a special form of communication that necessitates skills like negotiation, instruction, and cooperation. Often when two or more agents (human or machine) need to work together to accomplish a goal, they will engage in task-oriented dialogue. One example of this type of dialogue can occur between a teacher and a student. When a student comes to a teacher with a question about a particular subject, the teacher and student together establish a mutual goal: for the student to understand the topic in question. The teacher may break the goal down into several sub-goals: in order

for the student to understand photosynthesis, he needs to understand cellular respiration, chlorophyll, cell walls, etc. An effective teacher will try to make the dialogue as easy for the student to understand as possible. This is true of task-oriented dialogue in general - humans typically try to achieve their conversational goals in the most efficient way.

One way that people try to make their communication as easy to understand as possible is by reducing the cognitive load of the listener. The more cognitive processing the listener has to do, the longer it will take for a mutual understanding of beliefs to be reached. A speaker's conversational choices can greatly affect the coherency of the dialogue and thus the comprehension of the listener. [16]

The research I describe here deals with one possible way to reduce the cognitive load of a conversational partner – by increasing the coherency of the delivered dialogue. In specific, an underlying assumption of this research is that embedded references to past utterances improve the overall coherency of a dialogue, [16] and that – up to a point – the more such references there are, the more coherent the dialogue will be.

A reference in the context of this paper is a linguistic element that serves to link a current utterance with a previous one. In normal conversation, references are statements like: “Remember when I said that I would be late today?” or “As per our phone conversation, our group will maintain this software.” Referencing is possible in normal conversation only when both the speaker and the hearer share a common conversational history. This conversational history is also necessary when one of the dialogue participants is a computer. Software systems that are designed to converse with users over a period of

time often store a record of what has been said by either party. This record is commonly known as the discourse history.

Many discourse systems in research today attempt to make use of the discourse history through referencing. It is believed that such references serve a discourse purpose – usually that of bringing relevant information into context – and are therefore desirable. While a number of researchers have identified the need for references in dialogue, the work done to date in this area has been focused on identifying opportunities to refer to previous discourse and how to do so. This research takes a more pro-active approach. We introduce the concept of reference seeds. A seed is an utterance deliberately embedded in a dialogue so that it can be referred back to later. Seeds will be discussed in greater detail throughout Section 4.

The problem that this research addresses is that of how to increase the overall coherency and effectiveness of generated discourse by making references. Specifically, how can a discourse planning system increase the number of references in the dialogue it produces? This paper outlines a method we have developed for actively increasing the number of possible referents in a discourse. The method does this by opportunistically planting seeds in generated dialogue and referring back to them in subsequent discourse.

It is not the case that increasing the number of references always increases the coherency of a dialogue – references that are ill-chosen or inconsistent would actually have the opposite effect. The relevance and timeliness of references are important considerations. In order to prevent inconsistent or superfluous utterances, the seeding process employed for this research uses a list of dialogue objectives as a filter for potential seeds. A seed

will only be introduced into generated dialogue if it directly relates to one of the system's current dialogue objectives. The objectives represent knowledge goals that the system will attempt to attain over the course of an interaction.

The next section of this paper provides some background into the field of discourse planning, discourse in general, and current pedagogical and discourse planning research. Section 3 contains information about the implementation, the testbed used for this research, and an example interaction with the testbed. The following section describes an informal user study we conducted to test the feasibility of the seeding concept, and the results of the study. Section 5 details potential applications of the research. The last two sections contain suggestions for future work and conclusions.

2. Related Work

2.1 *Discourse Planning*

Systems that generate natural discourse may divide the generation problem into two phases – the first phase determines the content and organization of the presentation, and the second phase determines the linguistic realization of the information. Planning is a field that involves reasoning about actions in order to accomplish a goal. In the case of discourse generation, many systems maintain goals that correspond to the information they want to present. Planning can be used during the first phase of the generation process, to organize the information to be presented according to the system's discourse goals. When planning is used for this purpose, it is called discourse planning.

Planners are computer systems that generate plans for solving a problem. Planning problems are situated in a particular domain. In order to create a plan, planners take three things as input: an initial state, a set of operators, and a goal state. The initial state is a description of the domain before any planning is begun, and the goal state is a description of the desired state of the domain after the plan is executed. The set of operators is a list of actions that are possible in the domain. The plan will consist of a sequence of operations (actions in the domain) that, when executed, will transform the state of the domain world from the initial state to the goal state. For example, for the domain of cooking, the initial state, goal state, and operators for a planning problem might resemble those in Figure 2.1.

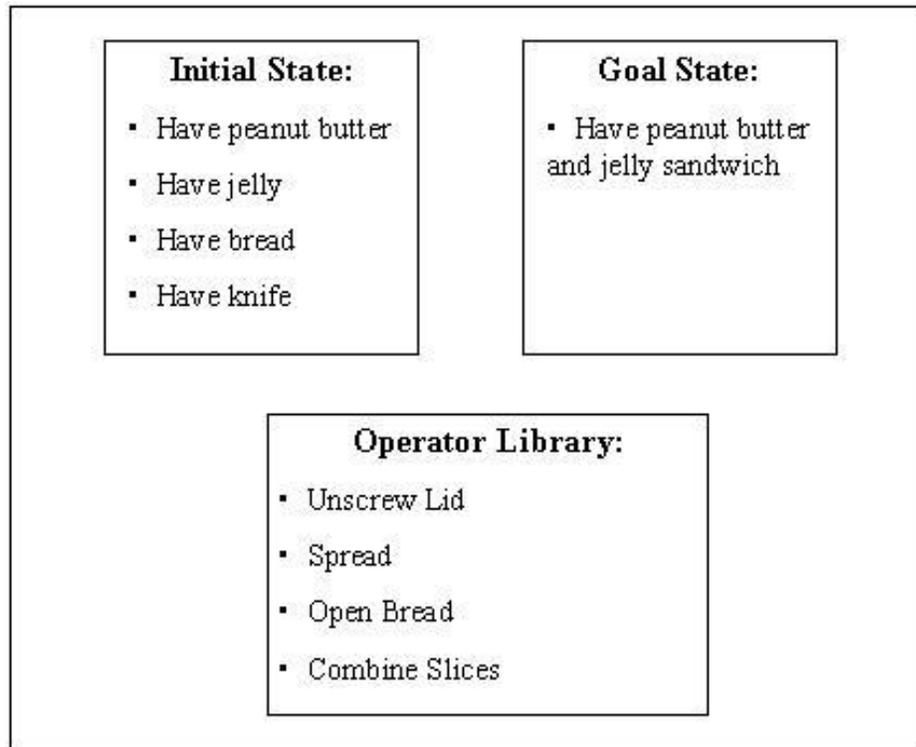


Figure 2.1 – Initial state, goal state, and planning operators for a problem from the cooking domain

The generated plan for the problem depicted above would consist of steps like open the bread, spread the jelly, spread the peanut butter, etc.

The field of discourse planning involves the use of a planner to generate, assemble, and/or control the flow of dialogue between two or more agents. Moore and Paris [12], Hovy [8], and Young [18, 19] have all implemented successful discourse planners. For the context of this research, a discourse is a multi-participant, multi-sentential sequence of utterances. Utterances are statements delivered by a discourse participant. The discourse itself will consist of a sequence of turns, which are groups of utterances

delivered by one participant at a time. In simple terms, a discourse is generated whenever two or more people conduct a conversation, either written or orally. The people involved in the conversation are discourse participants. They actively play roles in generating the discourse by delivering one or more utterances when it is their turn. Most humans act as discourse participants many times daily.

Natural Language Generation (NLG) systems are useful when a computer is to play the role of a discourse participant. A discourse planner is one part of such a system. The planner will cooperate with other components of the NLG system to create realistic dialogue on behalf of the computer discourse participant. A typical NLG system consists of many components. Some components are responsible for low-level tasks, like stringing words and phrases together to create a sentence, or resolving anaphora and noun phrases within a sentence. A discourse planner is a higher level component; instead of stringing together words to make a sentence, a discourse planner may group several sentences together to make a coherent argument.

As with all planning systems, discourse planners take as input a set of operators that characterize the set of actions available in the domain at hand. For discourse planners, these operators most typically specify a set of communicative actions (often at the level of speech acts that can be performed by the system [2]). Each operator is specified in a formal language in terms of its preconditions (things that must be true before the operation can take place) and effects (the results of the operation). These operators are typically conversational tasks, such as the introduction of a new topic, elaboration of an existing idea, the provision of support for a claim, or the acceptance of a proposal.

Discourse plans are created to attain conversational goals, for example, providing support

for a proposition, introducing a new subject, or rejecting a proposal. A plan to achieve the goal of rejecting a proposal may require operators such as reject (to explicitly reject the offer), one or more support operators (to give reasons for the rejection), and possibly even a suggest operator (to make a counter-offer).

Discourse planners can be especially beneficial when used to generate task-oriented dialogue. When two agents cooperate to achieve some goal, conversational or otherwise, they are generally required to share ideas about the goal(s) in question and the actions it necessitates [9, 4, 5]. The agents may be required to negotiate terms or actions, to remember key information, and to explain or justify decisions. Discourse planners are well suited to these types of conversational duties.

2.2 *The Intentional Structure of Discourse*

In 1986 Grosz and Sidner [4] introduced a new theory of discourse that is the basis of many of the current ideas on the subject. They assert that the structure of discourse is comprised of three elements: the physical structure (sequence) of the utterances, the intentional structure of the utterances, and the attentional state of the discourse. The first element, the linguistic structure, includes the utterances and their order, as well as the grouping of the utterances into logical groups called segments. Grosz and Sidner outlined several rules for assigning segment boundaries. In general, a segment is a group of utterances that all pertain to a particular topic, or all work together to achieve a particular discourse goal. A segment can consist of several turns, several utterances in one turn, or just one utterance.

Grosz and Sidner's second element of discourse, the intentional structure, includes the motivations behind each utterance and how they relate to the overall goal of the dialogue. The intentional structure is a record of each participant's intentions (or assumed intentions) for each utterance. Utterances are usually intended to further an overall discourse goal in some way. The third component, attentional state, is an on-going record of the focus of the discourse. A simplified definition of focus is that it is the current topic of conversation. It is the object of the participants' attention. Throughout the course of conversation, the focus can shift many times. Objects can go out of focus and return to it again during the flow of discourse.

Most discourse planner systems in use today attempt to identify and represent all three of Grosz and Sidner's components in the discourse they generate. The first component, the linguistic structure, is inherently addressed by a discourse plan – the plan itself contains at least a partial ordering of the utterances. The way steps in a plan are ordered can depend on the type of planner that was used to generate the plan. Plans can either be partially ordered or totally ordered. If a plan is partially-ordered, it means that there is more than one ordering of the steps that will satisfy the goal. A partially-ordered plan will contain constraints on the ordering of steps when it is necessary. Any ordering of steps that satisfies all of the ordering constraints imposed by the planner will succeed. A totally-ordered plan can only be executed in the order specified by the planner. The planner will provide a complete ordering for all of the steps. Step orderings (the sequence of utterances) are a component of linguistic structure, but they only apply to one turn, or one discourse plan at a time. The other component of discourse structure, segmentation, can be determined with some additional processing of the discourse

history. Grosz and Sidner touch on some rules for segmenting a discourse, and later work by Rich and Sidner [8] includes the implementation of a segmentation algorithm.

Determining a plan's intentional structure is not as easy. Work by Moore and Pollack [13] has shown that effects alone can not always be used to recreate the intentions behind a discourse. Young, Moore and Pollack [19] have developed rules and assumptions for determining the intentions of a discourse plan based on structures called causal links. Planners that annotate plans with causal links create a record of the plan's intentional structure. Causal links are described in detail in Section 3.2.

One of the ways that Grosz and Sidner determine the intentional structure of a discourse is by assigning Discourse Segment Purposes (DSPs) to each segment, and identifying relationships between them. The two primary relationships that exist between DSPs are dominance and satisfaction-precedence. If an action that satisfies DSP1 may also be intended to provide part of the satisfaction of DSP2, then it can be said that DSP2 dominates DSP1. In other words, DSP1 might be a sub-goal associated with the achieving DSP2. Satisfaction-precedence is defined more rigidly: if DSP1 must be satisfied before DSP2, then DSP1 satisfaction-precedes DSP2. Modeling these relationships across the segments in a discourse imposes a hierarchy on the segments that corresponds in part to the intentional structure of the discourse.

Decompositional planners (discussed in detail in Section 3.2) impose a hierarchy on the steps in the plans that they produce. This hierarchy of plan steps is similar to the segment hierarchy. Certain steps in decompositional plans can be decomposed into several sub-steps, each of which are either stand-alone steps or other decomposed steps. Figure 2.2

shows an example. Grosz and Sidner's dominance and satisfaction-precedence relationships are analogous to the relationships between sub-steps and super-steps in a hierarchical plan. In Figure 2.2, it can be said that the goal of step 1 dominates the goal of step 2, and the goal of step 4 satisfaction precedes the goals of steps 1 and 2. The hierarchy of a plan's steps, when it is available, can play a role in determining the plan's intentional structure as well as its segmentation.

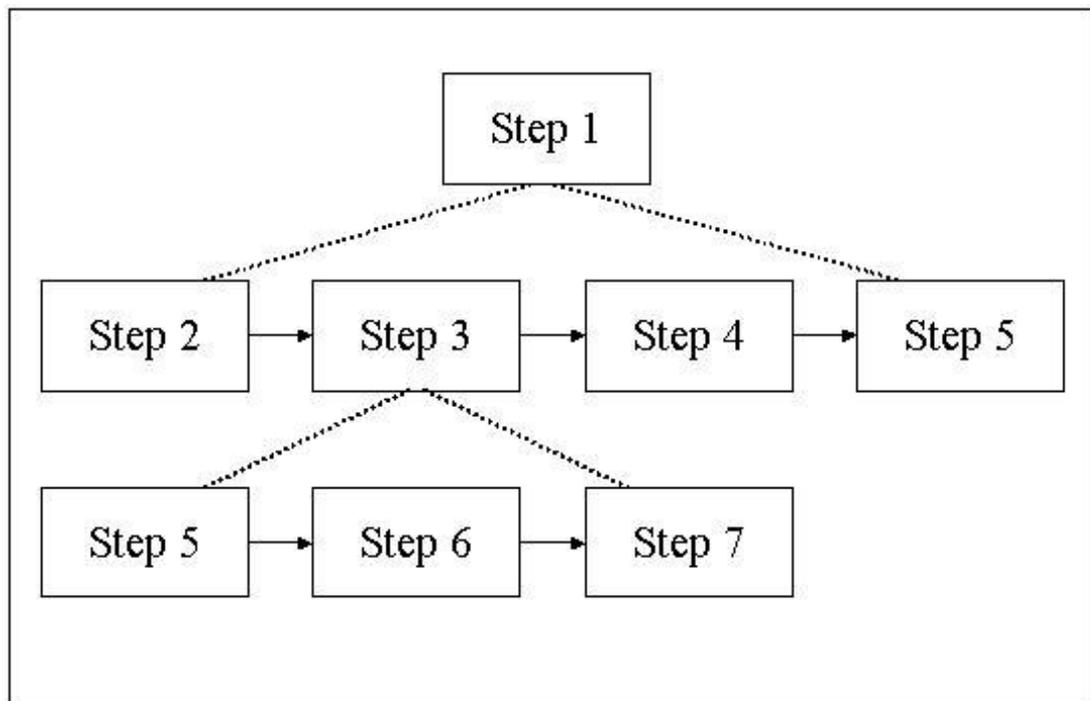


Figure 2.2 – Hierarchical steps in a plan

The attentional state of a discourse is the most difficult part to model, because it is a dynamic property of the dialogue, not just of a particular utterance or segment. Over the course of a long discourse, the focus of conversation may shift numerous times. Various systems have different needs and address the focus issue accordingly, but many use an

implementation based on Grosz and Sidner's focus stack concept. When a topic comes into focus, it is pushed onto a focus stack structure, and it remains there until all of the discourse goals associated with introducing that topic are achieved. As new topics are introduced, they are pushed onto the stack. When all of the discourse goals associated with a topic are resolved, the topic is popped from the stack.

2.3 Backwards References

Rosenblum and Moore [15] studied transcripts of tutorial dialogues as part of the research for an intelligent tutoring system, Sherlock. One of the things that they observed about their corpus of tutorial dialogues was that human tutors freely exploit past dialogue throughout a discourse. In the corpus of coherent tutorial sessions that they studied, the tutor made reference to previous discourse entities in order to make parallels between new and old material, to clarify examples, and to avoid repetition of explanations. The assertion that references to past dialogue improves the overall coherency of discourse is inferred from Rosenblum and Moore's research, and is an underlying assumption behind the research that is the subject of this paper.

The Sherlock system implementation is based on a discourse planner [12] that employs a stored dialogue history. Moore and Paris have demonstrated that, for explanation systems, it is necessary to maintain a history of past responses and to preserve the intentional structure of each response. The basis of the argument for preserving intentional information is that if a system is to be capable of responding properly to follow-up questions, it must be able to reason about its past responses. Imagine that, in the midst of a problem-solving session, a system gives a multi-sentential response to a

user's question that contains several important pieces of information. If the user asks for clarification about a particular utterance, the system may not be able to provide the clarification without knowing why it included the utterance in the response, or, what its intentions were when it originally included that utterance.

The Sherlock system makes use of its stored dialogue history when handling follow-up questions, and it incorporates examples from its history when generating responses to new questions. When a student asks Sherlock a question, this discourse planner is used to determine the structure and content of Sherlock's multi-sentential answer. As a first step in constructing this response, the planning system consults its dialogue history to see if it has given a similar response before. If it finds a past response that is suitably similar, it uses this response as an "anchor" and generates references to it when constructing the answer to the student's current question. Rosenblum and Moore's anchors are analogous to our seeds, with the notable exception that seeds are deliberately inserted in a plan, and anchors are naturally occurring. Rosenblum and Moore identify a number of possible ways to reference the anchor, including comparing and contrasting the past and current situations, instantiating specific examples of general concepts, or producing alternate explanations when previous ones were not understood [15].

2.4 Redundant Utterances

Teachers and tutors have long employed methods like reinforcement learning. The basic concept of reinforcement learning is that a tutor can repeat certain important pieces of information in order to increase the likelihood that a student will remember them. While typical discourse structure is designed by speakers to be free of redundancy, redundancy

is allowed in this context because the information is especially valuable. In Walker's [16] work focusing on the use of redundant utterances in normal conversation, she found that, contrary to previous assumptions, informationally redundant utterances (IRUs) are quite prevalent in daily life. An utterance is deemed to be informationally redundant if it is something the hearer either already knew or could infer.

Certain types of IRUs are backward references, because they refer to previous information. Walker gives three communicative functions of IRUs: attitude IRUs provide positive evidence of a speaker or hearer's understanding of a concept, consequence IRUs explicitly state a logical consequence of prior utterances, and attention IRUs bring into working memory propositions that may not currently be salient. Walker acknowledges that IRUs could possibly also be shown to be used as a rehearsal mechanism: to help speakers or hearers remember information. The "rehearsal mechanism" concept is obviously analogous to the idea of reinforcement learning, which is one of the more obvious applications of our research into seeding references. Certain types of IRUs are backward references.

3. Implementation

Section 3 outlines some of the implementation details associated with the seeding and referring concept. The first part defines some necessary terminology. The following section describes in detail the planner used for the implementation. Section 3.3.4 contains a description of a testbed that implements the theory for a particular type of reference. Following that is an example generated with the testbed.

3.1 Terminology

A concept that is relatively unexplored in current research is that of reference seeds, or what we refer to simply as seeds. A reference seed is an utterance deliberately introduced in dialogue with the intention of referring back to it in the future. Introducing a reference seed into dialogue is like planting a seed, and we have coined the term “seed” due to this analogy. The seed concept is similar to Rosenblum and Moore’s anchors, except that an anchor is found by searching through the discourse history for an appropriate action, and a seed is an action that is deliberately included in a plan with the purpose of being referred to later. Anchors have discourse purposes that are directly related to the planning goals – they satisfy a precondition, help resolve a threat, etc. Seeds do not necessarily do any of these things, they are opportunistically added to a plan for no reason other than the fact that they are likely to be referenced at some point in the future. From this point forward in this paper, we will use the term “seed” to mean a reference seed, and the term reference will always mean a backward reference.

In a conversation, references are expressions that make note of things discussed previously. They can be a simple repetitions of former statements: “As I said before, I will be on vacation on that day”; or utterances that are designed to make certain information salient: “Remember when we were talking about Tom?”; or they can be informational in nature: “This is different from what we were talking about before because...”. For the purposes of this research, references are utterances that make use of (conversational) actions stored in the discourse history. References can refer to seeds that were deliberately placed in a previous plan, or to regular steps (anchors), but they always refer backward.

A speaker may plant a seed when she anticipates that her hearer will need certain information later, or when she wants to foreshadow a future event, for example. Because seeds enable references, seeds can potentially serve all of the conversational functions of references themselves. They can be used to reinforce important information (through repetition), to compare and contrast situations, or to provide background information or to foreshadow events as mentioned above. Based on Rosenblum and Moore’s [15] observations that coherent tutorial dialogues often include references, we assert that references increase the overall coherency of a dialogue, especially an explanatory or task-related dialogue.

While there are many potential benefits to introducing seeds, a speaker (or discourse planner) must take care to use them effectively. A dialogue filled with seeds, especially seeds that do not get referenced later, can appear to be inconsistent or redundant. We have identified three criteria for deciding whether or not to introduce a seed into dialogue being generated: relevance, timeliness, and referral probability. Relevance is a measure

of how important the object of the seed is to the overall discourse goal(s). Timeliness should also be taken into consideration: is this the best time to plant a particular seed, or will another (better) opportunity arise in the future? The last criterion, referral probability, measures the likelihood of the seed in consideration actually being referenced in the future. The threshold for each of the above metrics, as well as how they are measured, depends on the system. For an explanatory system where consistency and coherence are the most important concerns, relevance should be a highly ranked factor. For systems where the user's path is at least partially pre-scripted, the referral probability factor is not important.

Central to the concept of reference seeds are the notions of objectives, discourse history, and run-time constraint satisfaction. Any implementation of seeding and referring as it is defined here must incorporate these concepts. The term discourse history has been well defined above, and run-time constraint satisfaction is discussed in detail in Section 3.2. Objectives are defined here.

Objectives in this context are propositions that a planning system intends to make true at some point in the future. Since we are dealing in the domain of informational dialogue, they are generally knowledge goals: for example, one might be that the user understands what a certain term means. Objectives are analogous to a speaker's high-level discourse goals, or what a speaker intends for a hearer to gain from the conversation. Representing the speaker's intentions as knowledge goals for the hearer is a common occurrence in operator languages [10, 2]. This means that an intention like "State fact" from the point of view of the speaker would instead be written as "Know fact" or "Believe fact" from the point of view of the hearer. Objectives will generally take this form.

Objectives are formally specified. For the testbed used for this research, they are generated according to basic rules, but we envision the objectives being the natural output of a plan recognizing component for an explanatory dialogue system. Plan recognizers are programs that analyze a user's actions from a planning point of view and attempt to identify the user's high-level goals. Many conversational systems in research include a plan recognizer as one of the components [1, 14, 11, 3]. One of these systems, the University of Rochester's TRAINS, contains an intentional agent that forms goals based on a representation of the user's mental state. The agent's intentions are updated on a turn-by-turn basis and may range from basic discourse obligations (such as accept or reject the user's proposal) to high-level intentions that will further the conversation or make progress towards a task-oriented goal. The objectives of a conversational planning system, in the context of this research, are analogous to the TRAINS agent's high-level intentions in that they represent discourse acts that the system will try to instantiate in order to assist the user in her attempt to accomplish her task. Objectives should be globally-accessible, based on the user, and updated after every turn.

A system's objectives are similar in concept to learning objectives that a teacher might have for a student during a lesson. For an explanatory or task-related system, our assumption is that the objectives capture the most important pieces of information that should be conveyed during a particular navigational session. We assume that the more frequently these objectives are referenced, the more effective the session will be in terms of learning benefits for the user.

A final point on objectives is that they are the filter that must be used to measure the relevancy of introducing a seed. Seeds are only introduced if they correspond to one of

the systems current objectives. The exact degree of correlation depends again on the system. Using the objectives as a filter helps reduce the occurrence of un-referenced seeds in generated dialogue.

3.2 *The Longbow Discourse Planner*

The planner chosen to implement this idea is Longbow. Longbow is a decompositional, partial-order, causal-link planner based on the DPOCL algorithm [19]. The choice to use a general-purpose planner and not a special-purpose discourse planner was deliberate, as it has been shown that discourse planning does not necessarily require a customized planner [17]. Young and Moore identified two properties that must hold for discourse plans: they must be correct (which means they are sound and complete in traditional planning terms), and they must document the intentional and informational structure of the discourse. Longbow meets the first qualification by guaranteeing the plans it generates to be sound and complete. Regarding the second property, Longbow's plans capture the intentional structure of a discourse by storing causal links between plan steps, and the informational structure of the plan is stored as parameters associated with a particular step [19].

Causal links are annotations to a plan that indicate a particular causal relationship between pairs of steps in a plan. A causal link originates from an effect of one step and terminates at the precondition of a later step. The link indicates that the effect of the first step is used within the plan to establish the condition needed by the precondition of the second step. When reasoning about the causal structure of a plan, Longbow instantiates a step in a plan in order to achieve a precondition of another step. When Longbow adds a

step to a plan for this reason, it also adds a causal link indicating the relationship between the two steps. The link means that the first step makes (or helps make) the second step possible, by achieving one of its preconditions. In a discourse plan, the causal structure of the plan is analogous to the intentional structure of the discourse.

Research into multi-level discourse by Moore and Pollack [13] has shown that the intentional structure of a discourse must be preserved in order effectively utilize previous utterances. If a user does not understand a response given by the system, the system will need to refer to the intentions it had when it gave that response in order to clarify it. This is not a problem for the Longbow system since the intentional structure of generated plans is inherently stored as part of the discourse plan itself.

Even though it is a general-purpose planner, the Longbow system provides three ways for users to incorporate domain knowledge. The first way is through the operator language. All planning systems rely on operators that are designed by users. Longbow's action representation allows two types of operators: actions and decompositions. Longbow actions can be primitive or composite. A primitive action will correspond to one step in a plan. An example of a primitive discourse action might be an affirmation: a response of "yes" to a piece of information that is expected to be known by both a speaker and a hearer. Composite actions correspond to more than one step in a plan and map to the second type of operator, decompositions. Decompositions are used when there can be more than one way to accomplish a particular action. For example, a reject action may be accomplished by simply saying "no" and giving a reason, or by refusing and giving a counter-proposal. This would be modeled in Longbow's action language by a composite *Reject* action that had two corresponding decompositions: one that simply gives a reason,

and one that gives a counter-proposal. Decompositions can contain many steps, each of which can be primitive actions or other decompositions. Due to the inclusion of decompositional operators in Longbow's action libraries, the generated plans are often hierarchical in nature. This is beneficial because as Grosz and Sidner have shown [4], the linguistic structure of normal conversation is often hierarchical as well.

Another way that Longbow users can add domain intelligence to the system is through a knowledge base interface. It is not unreasonable for any real-time or interactive planning system to need to refer to a changing knowledge base of information. Longbow users can interface with such a knowledge base by writing two functions: a `precondition-kb-query` function and a `constraint-kb-query` function, which will be used to satisfy preconditions or constraints at run-time. In order to understand the utility of these functions, imagine a different domain: that of a simulated office-place. Suppose in the simulated office-place there exists a virtual copy-machine, and associated with it is a *Copy* action. The details of the *Copy* action are shown in Figure 3.2:

<p>Action COPY:</p> <p>Parameters:</p> <ul style="list-style-type: none"> ▪ ?user ▪ ?copier ▪ ?document <p>Preconditions:</p> <ul style="list-style-type: none"> ▪ has-toner ?copier <p>Constraints:</p> <ul style="list-style-type: none"> ▪ is-authorized ?user ?authorization <p>Effects:</p> <ul style="list-style-type: none"> ▪ copied ?document ▪ recorded ?authorization
--

Figure 3.2 – An example action and its associated parameters, preconditions, constraints, and effects.

The precondition of *Copy* is that the copy-machine has toner. In a sufficiently complex virtual world with many characters performing actions, the copier could feasibly run out of toner (if this example was more realistic instead of hypothetical, one of the effects of the *Copy* action would also be a decrease in toner, but this is non-trivial to represent).

The planner attempts to satisfy this precondition by calling the `precondition-kb-query` function, which consults a knowledge base and determines if the copier has toner.

This precondition could only be satisfied at run-time, since there is no way for the writer of the *Copy* operator to know if toner will or will not exist each time the action is carried out in the office-place. The copy action also has an associated constraint, that the user of the copy-machine be authorized to use it. This information cannot be determined until

run-time either, so the planner will attempt to obtain it by calling the `constraint-kb-query` function. The `constraint-kb-query` function will consult its knowledge base and return a binding for the `?authorization` variable if the user is authorized. Both of these functions are useful when the planner is going to be used in a real-time interactive application with a complex domain. The `constraint-kb-query` function is heavily utilized in our implementation of the seeding and referring algorithm, which is discussed later.

The last way for Longbow users to influence the planning process is by writing a customized `rank-plan` function. Since Longbow is a plan-space planner, decisions are made with respect to whole plans. When considering how to satisfy a particular precondition, the planner will create a separate plan for each possible way to satisfy it. The possible plans that are generated make up the plan-space that Longbow searches. At each search interval, a ranking function is called that organizes the queue of potential plans. The plan at the front of the queue is expanded first and is thus more likely to be returned as the solution for a problem. The ranking function orders the queue according to the desirability of potential plans. If a Longbow user preferred for generated plans to be hierarchical as opposed to flat, then he could write the ranking function so that it gave preference to plans containing decompositions. Our testbed implementation also makes use of Longbow's plan ranking feature.

3.3 *The Algorithm*

Planning that supports sophisticated discourse functions like referring (both forward and backward) involves special-purpose intelligence. In general, there are two approaches to

how to program this intelligence: it can be added into the planner itself, or embedded in the action representation. Modifying the planner to perform special functions may seem like the most direct approach, and it might afford a performance increase over the other option. However, Longbow is a general-purpose planner, and we decided to keep it as such. Embedding the discourse intelligence in the action representation means that the system overall will be more portable and robust. Another reason to avoid modifying the planning algorithm is that the completeness or systematicity of the planner may be compromised.

Representing intelligence using the action representation means that the techniques used to perform functions like referrals can be tightly coupled to the domain, which can be a big advantage. Writing a function to search for possible ways to seed any objective might be very difficult, but when the types of objectives are known beforehand, the task is much simpler. The disadvantage of this is that much more action specification must be done at the design-time of a system.

Since the intelligence is modeled in the action library instead of coded in the planner, it is not really accurate to refer to a seeding and referring “algorithm.” Breaking the process into steps is a simplification of what actually happens, in the same way that plan-space planning in general can be simplified. A naïve way to think of how the planner operates is that it finds a flaw, resolves it, and repeats this until there are no more flaws. This might involve finding an open precondition and picking an action to satisfy it. In reality, the planner retrieves a plan from its queue, finds a flaw in that plan, generates one new plan for every possible way to resolve that flaw, adds the new plans to the queue, orders the queue, and starts again. A simplified, but not entirely accurate, way to characterize

the seeding and referral process is that the planner chooses a step to add, searches its discourse history to see if it can refer backwards to any actions, adds a reference if it can, adds the step, then searches through the objectives to see if it can introduce a seed that relates to that step in some way. All of the steps of this process are modeled by actions and decompositions, with searching accomplished using the `constraint-kb-query` function. In actuality, the planner executes these steps concurrently as it searches the space of possible plans.

The first part of the process - choosing a step to instantiate and add to the plan - is accomplished naturally as the planner resolves flaws. The possibility of including seeds or references in the plan is modeled using decompositions. The planner will have an open precondition to try to satisfy. One of the ways it can do this is by adding a step. Suppose there is only one action that will satisfy that particular precondition. We represented that action using four different decompositions: one that only satisfies the precondition, one that does that as well as generates a referral back to an appropriate action in the discourse history, one that satisfies the open precondition and includes one or more additional steps that attempt to seed a future reference, and one that satisfies the precondition, makes a backward reference, and includes a seed. Figure 3.3 shows four decompositions for an action called *Respond-Relation*, which gives a relation between two variables.

Decomposition 1 (no seed, no referral):

```
(define (decomposition RESPOND-RELATION)
:parameters (?OBJECT)
:constraints ((RELATION-P ?OBJECT)
              (DATA-SET ?OBJECT ?DATA-SET1 ?DATA-SET2)
              (RELATION-VALUE ?OBJECT ?VALUE)
              (POSSIBLE-REFERRAL ?OBJECT ?REFERENT)
              (ANOTHER-RELATION ?OBJECT ?SEED))
:links ((STEP3 (RELATION ?OBJECT ?VALUE) END))
:steps ((STEP3 (GIVE-RELATION ?OBJECT ?VALUE))))
:rewrites (((PRESENTED ?OBJECT)
            ((RELATION ?OBJECT ?VALUE))))))
```

Decomposition 2 (seed only):

```
(define (decomposition RESPOND-RELATION)
:parameters (?OBJECT)
:constraints ((RELATION-P ?OBJECT)
              (DATA-SET ?OBJECT ?DATA-SET1 ?DATA-SET2)
              (RELATION-VALUE ?OBJECT ?VALUE)
              (POSSIBLE-REFERRAL ?OBJECT ?REFERENT)
              (ANOTHER-RELATION ?OBJECT ?SEED))
:links ((STEP2 (RELATION ?OBJECT ?VALUE) END)
        (STEP3 (SEEDED ?SEED) END))
:steps ((STEP2 (GIVE-RELATION ?OBJECT ?VALUE))
        (STEP3 (SEED ?SEED)))
:rewrites (((PRESENTED ?OBJECT)
            ((RELATION ?OBJECT ?VALUE)
             (SEEDED ?SEED))))))
```

Decomposition 3 (referral only):

```
(define (decomposition RESPOND-RELATION)
:parameters (?OBJECT)
:constraints ((RELATION-P ?OBJECT)
              (DATA-SET ?OBJECT ?DATA-SET1 ?DATA-SET2)
              (RELATION-VALUE ?OBJECT ?VALUE)
              (POSSIBLE-REFERRAL ?OBJECT ?REFERENT)
              (ANOTHER-RELATION ?OBJECT ?SEED))
:links ((STEP2 (REFERRED-TO ?REFERENT) END)
        (STEP3 (RELATION ?OBJECT ?VALUE) END))
:steps ((STEP2 (REFER ?REFERENT))
        (STEP3 (GIVE-RELATION ?OBJECT ?VALUE)))
:rewrites (((PRESENTED ?OBJECT)
            ((RELATION ?OBJECT ?VALUE)
             (REFERRED-TO ?REFERENT))))))
```

Decomposition 4 (seed and referral):

```
(define (decomposition RESPOND-RELATION)
:parameters (?OBJECT)
:constraints ((RELATION-P ?OBJECT)
              (DATA-SET ?OBJECT ?DATA-SET1 ?DATA-SET2)
              (RELATION-VALUE ?OBJECT ?VALUE)
              (POSSIBLE-REFERRAL ?OBJECT ?REFERENT)
              (ANOTHER-RELATION ?OBJECT ?SEED))
:links ((STEP2 (REFERRED-TO ?REFERENT) END)
        (STEP3 (RELATION ?OBJECT ?VALUE) END)
        (STEP4 (SEEDED ?SEED) END))
:steps ((STEP2 (REFER ?REFERENT))
        (STEP3 (GIVE-RELATION ?OBJECT ?VALUE))
        (STEP4 (SEED ?SEED)))
:rewrites (((PRESENTED ?OBJECT)
            ((RELATION ?OBJECT ?VALUE)
             (REFERRED-TO ?REFERENT) (SEEDED ?SEED))))))
```

Figure 3.3 – 4 Different Decompositions for the Respond-Relation action.

In order to make a reference in a decomposition, the planner needs to find a step in the discourse history to refer to. In some cases this step will be a seed, in others, the planner may serendipitously find a naturally occurring anchor. The second decomposition for *Respond-Relation* contains a reference. If the reference is possible at run-time, the constraint *Possible-Referral* will return a binding for the ?REFERENT variable. If no binding is returned, that action cannot be instantiated and that possible plan (that uses the referral decomposition to satisfy a precondition) will fail. The search algorithm compares the open precondition in question to the effects of steps in past discourse plans. If the precondition to be satisfied correlates sufficiently with a previous effect, then a referral is made. For our implementation, the matching algorithm is deliberately simplistic: it looks for a direct match between the precondition in question (including parameter bindings) and a past effect (and its parameter bindings). In general, the function used for matching is system-dependent and should be written with consideration for the types of references the system will employ, the various actions in the library, the design goals of the system, and other relevant issues.

Seeds are included in the plan in a similar way. The third decomposition in Figure 3.3 contains a seed. If the constraint *Another-Relation* is satisfied (meaning that the variable ?SEED is bound to a proposition, then the seed is possible. Again, the matching algorithm used here should be system-dependent. For our testbed purposes, it is less generic and takes into account the possible ways in which seeds can be introduced in our system. In general, the open precondition that the step satisfies is compared to the system's current objectives (which are obtained from a file or from another module). This comparison is not looking for an exact match, however, because that would mean

that the seed is not necessary – the step being added already meets one of the objectives – either it is itself a seed, or, hopefully, it is an instance of the objective that can refer back to a previous seed. This comparison is looking for a partial match of parameters or effects.

In higher-level terms, if the planner is about to include information about a certain subject, and some of that information is the same as one of the objectives, it can opportunistically mention that objective. For example, suppose that the system is about to inform the user that the variable x is related to the variable y . Suppose also that one of the system's current objectives is that the user knows that the variable y is related to z . The system can include a seed for this objective by saying informing the user of both relationships at the same time: "X is related to Y, which is related to Z." Later in the interaction, when the user asks about variable Z , the system can refer back to the information that Y and Z are related.

Representing the seeds and referrals as actions means that more potential plans are possible for a given problem. For a planning system not employing seeding and referring, if a given problem could be solved in one step with the *Respond-Relation* action, only one complete plan was possible. With the seeding and referring implementation, four plans would be possible. Due to the non-deterministic nature of planning algorithms, any one of these plans could be returned as the solution to the problem. For a system implementing seeding and referring, we want to favor plans that include seeds and references. The `rank-plan` function is used to ensure that plans including seeds or references are moved to the front of the planner's queue and are thus more likely to be returned.

3.4 The Testbed

In order to conduct a small-scale user study and to illustrate the potential of the seeding and referring concept, we have written a small test system called “Garage.” Garage is a car-maintenance expert system, able to answer general questions about cars and car maintenance. Using the Garage system for testing allows us to fully control the domain and the user’s experience, without being forced to implement a fully-functional, robust problem analysis system. We have chosen to implement only one style of reference, which is a direct repetition of a past utterance. This is arguably the most simple type of reference, and the most easily implemented. Implementing referencing through repetition affords very basic matching algorithms for satisfying run-time constraints (as described above). While we acknowledge the limitations associated with our implementation choice, we feel that a system that can benefit from using the seeding and referring algorithm for simple repetition can showcase the potential of implementing the concept for more complicated references.

Repetition in an explanatory system can be a benefit in many ways. Walker [16] suggests that repetition can be used as a “rehearsal mechanism” to assist conversational participants in remembering key facts. Reinforcement learning is an obvious potential extension of this concept to tutorial systems. Rosenblum and Moore [15] show that referencing occurs naturally in tutorial dialogues, and Walker [16] shows that IRUs occur frequently in conversation. As stated in the introduction to this research, we believe that one of the potential benefits of any type of referencing is increased coherency of dialogue.

The Garage system accepts queries in restricted natural language and provides responses in natural language with no graphics. The system is comprised of three components: Longbow, an IM, and a database of car maintenance information. Instead of objectives being set by a plan recognizer, they are based on simple rules encoded in the IM. The system is able to answer five types of questions: basic questions (“What is a catalytic converter?”), difference questions (“What is the difference between shocks and struts?”), process questions (“How does the engine work?”), relationship questions (“How does fuel-efficiency affect a car’s emissions?”), and problem-solving questions (“Why won’t my car start?”). In addition, the system is also able to provide domain information by responding to commands like “List the components of the exhaust system.”

The IM is responsible for converting the user’s question into a problem structure that Longbow can interpret, and for converting discourse plans into responses that are displayed to the user. Both of these conversions are template-based. The IM also regulates data flow between components, and is responsible for retrieving information from the database when it is needed as a part of a response. When the IM receives a completed discourse plan from Longbow, it will store the plan in a discourse history structure, convert it to text, obtain any needed information from the database, and display the response to the user. The IM has the additional responsibility of setting the objectives for each turn, which it does according to a set of hard-coded rules.

The database contains information that the system will need in order to answer user’s questions. Each question that the user asks can be broken into several sub-questions whose answers are stored as text strings in a hash table. For example, the question “How often should I service my car?” might be broken into three sub-questions: how often

should the oil be changed, how often should the transmission be serviced, and how often should the tires be rotated. The database will in turn contain text string answers to each of those questions. This method of generating multi-sentential text is not as robust as the methods employed by today's NLG systems, but it is adequate for our testbed.

It was only necessary to create a small number of operators for use with the Garage system. The action library used for user testing contains a global action called answer with many decompositions corresponding to each type of question and command that the system responds to. Other actions introduced in decompositions include conversational acts such as listing components, delivering a supporting statement, stating a fact, and of course, seeding a future reference or referring to a seed.

3.5 *An Example*

We have generated an example dialogue showing that shows some of the potential of the seeding and referring concept. This example is generated using the Garage system. For simplicity, assume that at the start of this sample session, there were no previous plans stored in the discourse history, and there were no initial objectives. Figure 3.5.1 shows the example.

User: "List the systems in the car."

System: "The car has 5 systems. They are:

electrical system - The electrical system generates, stores, and distributes the electrical current required to start and run a car.

lubrication system - The lubrication system controls the amount of friction and heat generated in the engine.

exhaust system - The exhaust system manages the flow of exhaust gases from the engine out of the vehicle.

drive train - The drive train is the system that is responsible for moving the car.

fuel system - The fuel system delivers gasoline to the engine."

User: "List the components of the drive train."

System: "The drive train has 6 components. They are:

differential

gears

crankshaft

transmission

tire

engine

The engine, transmission, and gears are key components of the drive train."

User: "How does the engine work?"

System: "Most cars in operation today have 4-stroke internal combustion engines.

The engine operates in 4 phases:

1. The intake stroke brings air and gas into the main cylinder.
2. The compression stroke compresses the air and gas at one end of the cylinder.
3. The combustion stroke is when the spark plugs fire, causing an explosion.
4. The exhaust stroke forces the exhaust out of the cylinder through the exhaust valve.

The engine runs through this cycle hundreds of times each minute.

The transmission's function is necessary to keep the engine from exploding."

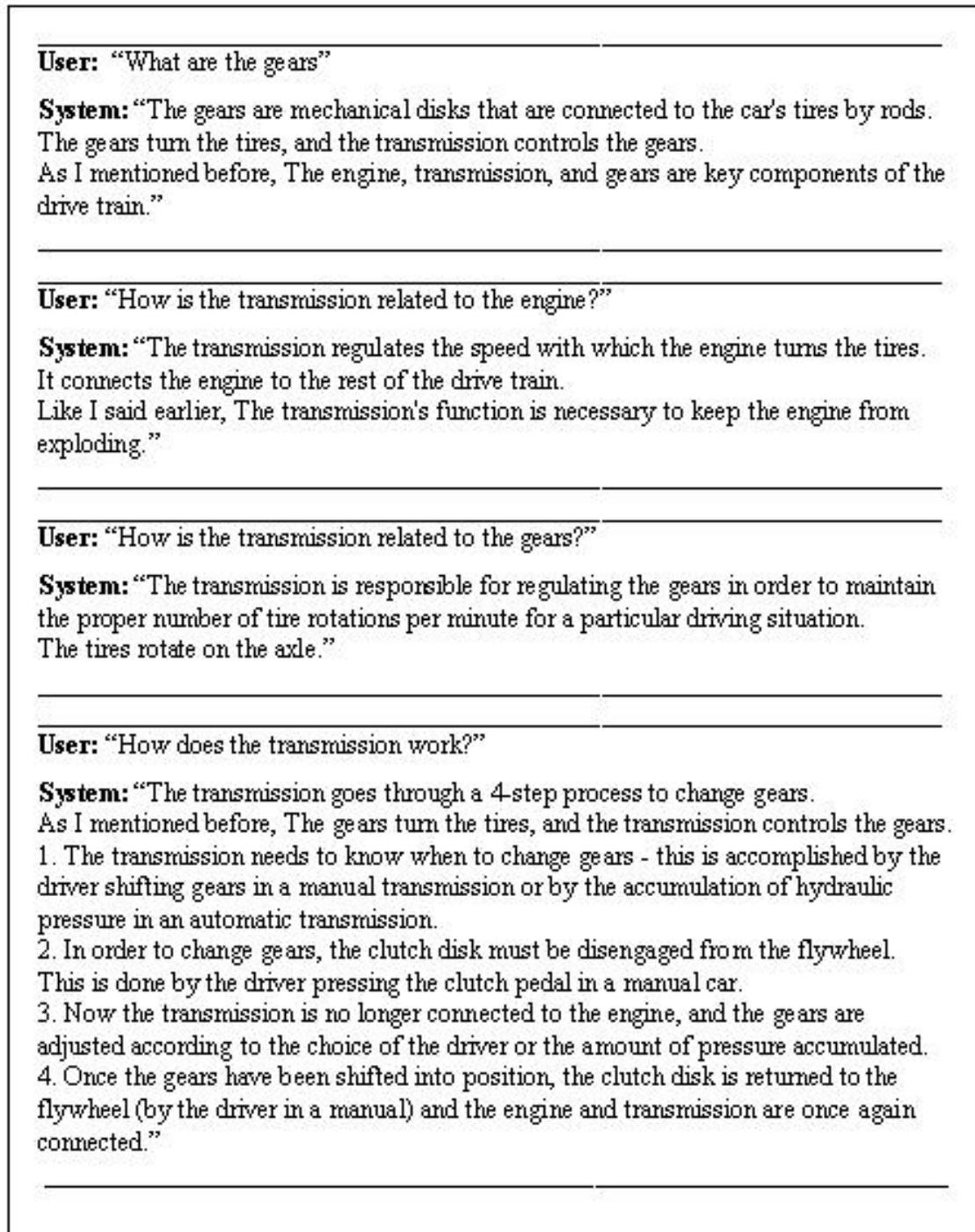


Figure 3.5.1 – An example dialogue with the Garage system.

For each of the interactions above, the information flows as follows: first, the user's question is translated into a problem and passed to Longbow. Figure 3.5.2 shows an example of the problem data structure resulting from the user's third question. The global objectives are also updated as a part of this first step, based on the user's question. Second, Longbow generates a complete discourse plan, taking the IM's globally accessible objectives into account. An example of a generated discourse plan is shown in Figure 3.5.3. Third, the plan is passed back to the IM and it is stored in the globally accessible discourse history structure. The fourth and last step in the process is the conversion of the discourse plan to multi-sentential text. This conversion is done by the IM according to pre-defined templates. Any queries to the database for text strings are also performed in this step.

```
Problem Three:  
Initial State:  
    <no initial conditions>  
Goal State:  
▪ Know-user (process engine)
```

Figure 3.5.2 – An example of a problem data structure.

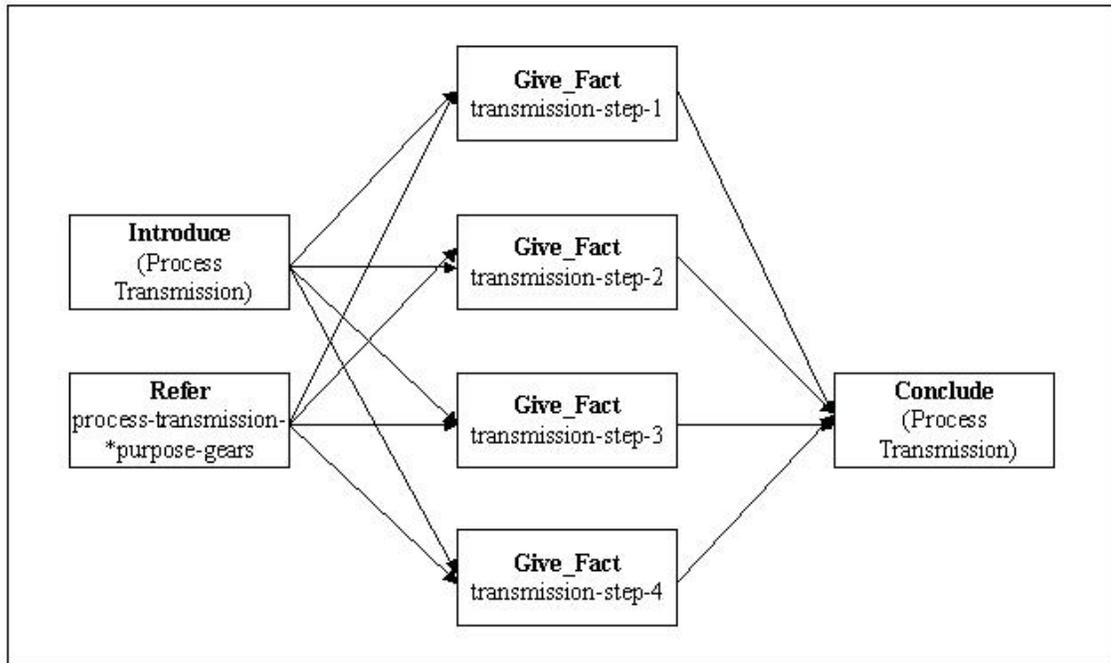


Figure 3.5.3 – An example discourse plan.

In the example above, the user first directs Garage to list the systems in a car. This command is processed and the IM and does not result in the creation of any objectives. Longbow receives the problem structure and generates a plan, which is passed back to the IM and processed, and the response is displayed. This first turn is an example of a very basic interaction with the system. The second turn is also a command, this time to list the systems in the drive-train. It is handled by the IM in much the same way, except that this time three objectives are created. Since the user inquired about the drive-train, Garage assumes that at some point the user might want to know about what it considers to be three main components of the drive-train: the engine, the transmission, and the gears. These three objectives (stored in a propositional language) are: (know-user (purpose

engine)), (know-user (purpose transmission)), and (know-user (purpose gears)). Figure 3.5.4 shows the objectives that are created in response to each of the user's questions from the sample session above. The problem structure is passed to Longbow, and a plan is generated. This time, the completed plan includes a seed. The IM displays the response, which contains the manifestation of the seed, the statement "The engine, transmission, and gears are key components of the drive-train." This seed is included because the system has reason to believe (due to the objectives) that the user will inquire about either the transmission, the gears, or the engine in the future. If the user does inquire about any of those parts, the system can use that opportunity to refer to the seed and reinforce the concept that the engine, transmission, and gears are key components of the drive train.

User: "List the systems in the car."

System: "The car has 5 systems. They are:

electrical system - The electrical system generates, stores, and distributes the electrical current required to start and run a car.

lubrication system - The lubrication system controls the amount of friction and heat generated in the engine.

exhaust system - The exhaust system manages the flow of exhaust gases from the engine out of the vehicle.

drive train - The drive train is the system that is responsible for moving the car.

fuel system - The fuel system delivers gasoline to the engine."

User: "List the components of the drive train."

System: "The drive train has 6 components. They are:

differential

gears

crankshaft

transmission

tire

engine

The engine, transmission, and gears are key components of the drive train."

•(know-user (purpose engine))

•(know-user (purpose transmission))

•(know-user (purpose gears))

User: "How does the engine work?"

System: "Most cars in operation today have 4-stroke internal combustion engines.

The engine operates in 4 phases:

1. The intake stroke brings air and gas into the main cylinder.

2. The compression stroke compresses the air and gas at one end of the cylinder.

3. The combustion stroke is when the spark plugs fire, causing an explosion.

4. The exhaust stroke forces the exhaust out of the cylinder through the exhaust

valve.

The engine runs through this cycle hundreds

The transmission's function is necessary to k

•(know-user (purpose transmission))

•(know-user (process transmission))

•(know-user (purpose gears))

•(know-user (purpose cylinder))

•(know-user (relationship engine transmission))

<p>User: "What are the gears?"</p> <p>System: "The gears are mechanical disks that are connected to the car's tires by rods. The gears turn the tires, and the transmission controls the gears. As I mentioned before, The engine, transmission, and gears are key components of the drive train."</p>	<ul style="list-style-type: none"> •(know-user (purpose tire)) •(know-user (relationship engine transmission)) •(know-user (purpose transmission)) •(know-user (process transmission)) •(know-user (purpose cylinder))
<p>User: "How is the transmission related to the engine?"</p> <p>System: "The transmission regulates the speed with which the engine turns the tires. It connects the engine to the rest of the drive train. Like I said earlier, The transmission's function is necessary to keep the engine from exploding."</p>	<ul style="list-style-type: none"> •(know-user (purpose tire)) •(know-user (purpose cylinder)) •(know-user (process transmission)) •(know-user (relationship engine transmission))
<p>User: "How is the transmission related to the gears?"</p> <p>System: "The transmission is responsible for regulating the gears in order to maintain the proper number of tire rotations per minute for a particular driving situation. The tires rotate on the axle."</p>	<ul style="list-style-type: none"> •(know-user (purpose tire)) •(know-user (relationship engine exhaust-system)) •(know-user (purpose axle)) •(know-user (process transmission)) •(know-user (purpose cylinder))
<p>User: "How does the transmission work?"</p> <p>System: "The transmission goes through a 4-step process. As I mentioned before, The gears turn the tires, and the transmission controls the gears.</p> <ol style="list-style-type: none"> 1. The transmission needs to know when to change gears - this is accomplished by the driver shifting gears in a manual transmission or by the accumulation of hydraulic pressure in an automatic transmission. 2. In order to change gears, the clutch disk must be disengaged from the flywheel. This is done by the driver pressing the clutch pedal in a manual car. 3. Now the transmission is no longer connected to the engine, and the gears are adjusted according to the choice of the driver or the amount of pressure accumulated. 4. Once the gears have been shifted into position, the clutch disk is returned to the flywheel (by the driver in a manual) and the engine and transmission are once again connected." 	<ul style="list-style-type: none"> •(know-user (purpose tire)) •(know-user (purpose cylinder))

Figure 3.5.4 – An example transcript with objectives. The objectives that are formed in response to each turn are shown beneath the turn.

4. Empirical Evaluation

The concepts of seeding and referring hold much promise for discourse systems of the future. While there is a significant amount of current research into the notion of referring, there is no known research on the subject of seeding referrals. In order to demonstrate the potential of this approach and to gauge its feasibility, we have performed a preliminary user evaluation with the Garage system. While this evaluation was preliminary in nature, the results were encouraging and we feel that further research as well as a more extensive study is warranted.

We had three primary aims in designing the user study:

1. To see if Garage users characterized the system's behavior as repetitive, and if so, if it had a negative impact on their views of the experience.
2. To see if Garage users characterized the text containing seeds as being inconsistent or extraneous, especially text containing seeds that were not referred to, and if so, if that had a negative impact on users' views of the experience.
3. To see if the inclusion of the references as direct repeated statements aided in the user's learning process by reinforcing important information.

Questions 1 and 2 are fairly subjective, and could also apply to any implementation of the seeding and referring concept (meaning, not just through repetition). Question 3 applies specifically to the use of repetition as a learning tool and is more empirical in nature.

For the study, we wanted to be able to control the user's knowledge goals so that we could obtain accurate information about what was learned, and we also needed to be able

to control the number of seeds and/or references generated in each session. In addition, we needed to be able to ensure that different users would have the same type of experience with the system. We could address those concerns by pre-scripting the user's session with the system, but that approach comes with its own set of concerns, for example, we wondered if users would really pay attention to the system's responses if they did not have a choice about the question that was asked. We settled on an approach that allowed the user to have some control over the system but was pre-scripted enough to allow us to obtain usable data.

We created a modest Graphical User Interface (GUI) for the system to enable this type of interaction. Figure 4.0 shows the GUI. For each sample interaction with the system, the user is given a knowledge goal that appears in a text-box in the top-right corner of the GUI. This is the user's high-level goal and it will remain constant throughout the session. Underneath that goal is a sub-goal pertaining to a specific subject within the domain of the overall goal. This goal changes as the user obtains information about various topics. To the left of the Goals text-box is a drop-down list labeled "Query." This list contains a selection of questions that the user can ask at any time. The list changes each time a sub-goal from the Goals box is satisfied. One or more specific questions from the Query list will be necessary to satisfy each sub-goal. Figure 4.0 shows an example. Suppose the user is working on attaining the sub-goal "Understand what emissions are." She can choose from the list of questions shown in the figure. To attain this sub-goal, she will need to receive the answers to the first two queries ("What is an emission?" and "What is the relationship between emissions and fuel efficiency?"). It is not necessary to know the answers to the last two questions in order to achieve the

goal. This method of querying the system allows the user some choice while still constraining her possible path through the system. Once the user chooses a query from the list, the system's response is contained in the Response text box in the center of the window. If the current sub-goal has been satisfied, it changes and the Query list is re-loaded with a new set of questions.

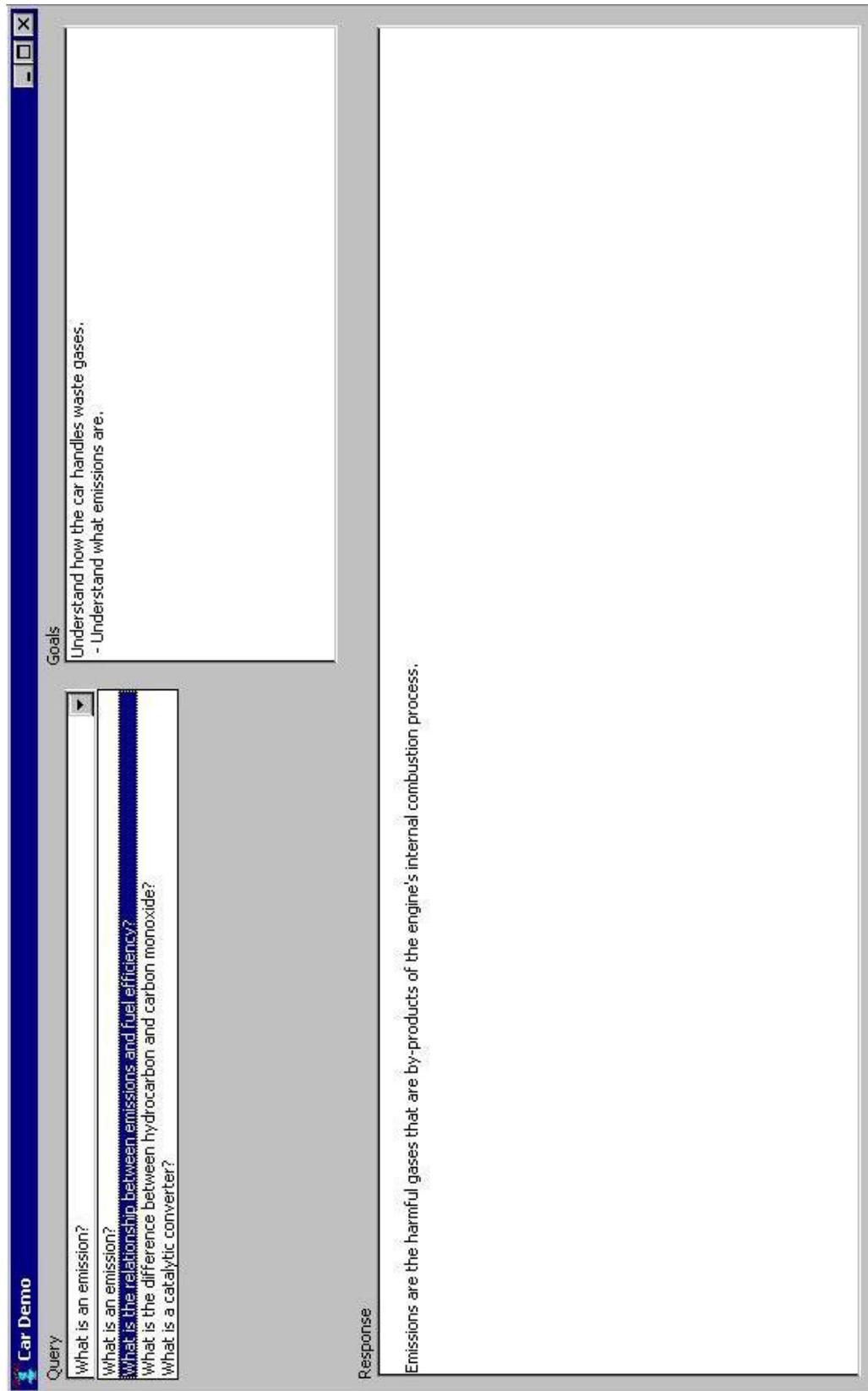


Figure 4.0 – The user interface for the informal evaluation.

For the evaluation, each user participated in two sessions with the Garage system, one that contained seeds and references and one that did not. The sessions were about different topics and contained no overlapping information. Before the evaluation began, the user was instructed on how to use the system and was advised that she would be tested on the information learned (this was intended to ensure that the user would read each response). After each session, the user was asked to complete a survey. The surveys contained two sections: an opinion section designed to address concerns 1 and 2 (repetitiveness and consistency) identified above, and a test section containing specific questions on the topics presented to the user, designed to address concern 3. We attempted to consider the user's previous knowledge about cars with a question on the survey and to control for ordering effects by randomly varying which session (seeded or unseeded) the user participated in first. Each session consisted of five sub-goals and took eight conversational turns. Appendix A contains the surveys used (one for each session, seeded and unseeded).

The subject pool consisted of 11 volunteers, approximately half male and half female, ranging in age from 18-52, with varied levels of computer experience. Based on his or her reported knowledge of cars and car maintenance, each user was categorized as either a beginning, average, or advanced user. This sampling contained five average subjects, five beginners, and one advanced user. The complete results of the evaluation are detailed in Appendix B. The following paragraphs of this section contain key results as they pertained to each of our three user study goals.

Questions 9 and 10 addressed the first aim of the user study – question 9 asked users to rank the system's repetitiveness and question 10 asked users if they found repetitiveness

in systems like Garage to be annoying. For question 9, a scale of 1-5 was used, with 1 being that the system was always repetitive and 5 being never repetitive. For the seeded version, the mean ranking was a 2.73, and for the un-seeded version, the mean was a 3.27. This suggests that the seeded version was perceived as more repetitive than the unseeded version. When asked if repetitiveness was annoying, 6/11 of the users thought so for the seeded version, and 4/11 did for the unseeded version. These results suggest that repetitiveness can have some negative effects on the user's perception of the system. Question 3 addresses a possible benefit of repetitiveness or of referring in general, which is increased coherency across responses. Users did rank the seeded version of Garage as more coherent (mean = 2.27, with 1 being most coherent) than the unseeded version (mean = 2.45).

From these results we conclude that the seeded version of Garage was indeed characterized as more repetitive than the unseeded version, and that approximately half of our subjects found this to be annoying. One possible explanation for this result is that the interactions subjects participated in contained too many seeds and referrals in too few turns. In our user study, each test scenario contained eight necessary turns. For the seeded version of Garage, four of those contained seeds and three contained referrals. It is possible that the frequency of seeding and referring was too high for this interaction, which caused some subjects to view the system as repetitive and to be annoyed by this quality. Seeding frequency might be another parameter that should be taken into consideration by systems employing this technique. This is addressed in the Future Work section.

The second aim of the user study was to determine if subjects characterized seeds (within responses) as extraneous or unnecessary. Questions 11 and 12 asked users about particular responses from the seeded interaction. Question 11 inquired about a particular seed that did not get referred to during the interaction. Five out of eleven subjects responded that they found the text containing the seed to be extraneous. All of the respondents who did not find the seed to be extraneous were in the beginner or average category of car knowledge. Of the five users who did find the seed to be extraneous, only one was a beginner. These results suggest that seeds might be less noticeable (and potentially more effective) to novice learners. In fact several of the subjects commented without provocation that the repetitiveness of the system was beneficial to beginning users or could be used as a learning tool. A possibility for future research is to measure the negative and positive effects of seeding and referring for different levels of users, from novices to experts.

Question 12 asked subjects if they found a particular referral (e.g. “As I mentioned before...”) to be out of place or unnecessary. For this question, eight out of eleven subjects responded that they did. Again, all three of the users who did not find the referral to be out of place were in the beginning or average categories. We believe that this result might be different for different types of referrals. While a direct repetition of previous information might be viewed as unnecessary, a comparison between a current situation and a previous one might be considered useful.

The last aim of the user study was to find out if the direct repetition of certain pieces of information aided in the subject’s learning process by reinforcing important facts.

Questions 4, 5, and 6 attempted to discover this by questioning the subject about specific

information contained in the system's responses. For the unseeded version of Garage, the questions concerned the car's electrical system, and for the seeded version, they concerned the drive train. For questions 4 and 5, subjects appeared to retain the same amount or more information from the unseeded system than from the seeded system. This result is contrary to our hypothesis, and we feel it can be explained by the difference in the knowledge tested. All users received the same questions, but the questions were different for each version of the system (each subject answered the questions twice, once for each version). We attribute the difference in retention across versions for questions 4 and 5 to the fact that the questions for the unseeded version (about the electrical system) may have simply been easier to answer than the questions for the seeded version (about the drive train). This hypothesis could be tested in a future study by having subjects answer questions based on only one interaction with either a seeded or unseeded system. The interactions would contain the same queries and users would be tested on the same information. Then the retention of subjects who used the seeded version could be fairly compared to the retention of subjects using the unseeded version.

While a simple comparison of responses for questions 4 and 5 did not directly support our hypothesis, there are some positive indicators that seeds and referrals can be effective learning tools. For example, for the seeded version, four out of five of the beginning users got at least two (out of three) of the knowledge questions correct. All of these questions asked about seeds that were referred to over the course of the interaction. For question 5, four subjects gave an answer that contained exactly the same phrasing and content as the seed repeated in the test interaction. For question 6, eight out of eleven subjects (including four out of five beginners) responded correctly for the seeded version,

and only five out of eleven subjects answered correctly for the unseeded version. This again supports our theory that seeding and referring can be a learning tool.

Section 6 (Future Work) reiterates some of the suggestions made here for improving the user study process. We feel that even with the relatively small number of subjects tested for this particular study, the results justify further implementation and analysis of the seeding and referring concept.

5. Applications

The ability to dynamically seed future references in a discourse has many potential applications. It can be particularly useful in pedagogical systems. Many times systems that are designed for teaching purposes maintain lists of objectives for an interaction. The teaching agent may have three pieces of information to impart to the user, for example, that make up the “key points” of a lesson. Knowing these pieces of information are going to be imparted at least once over the course of the lesson, the agent could include as many reference seeds to these topics as possible. When the time comes to introduce each piece of information, there will be anchors that can be referred back to, making the overall dialogue more coherent. The references might also serve to reiterate certain important facts, which helps the learner remember them. This concept is commonly known as reinforcement learning.

The seeding and referring methods outlined have only been tested with one kind of reference, which is a repeat of a past utterance. However, these methods could be used to facilitate many of the types of references commonly desired in task-oriented dialogue, such as comparing and contrasting situations or varying past explanations. Expanding this research to include these reference types would be a matter of creating new decompositions and more robust methods of searching a list of objectives (or dialogue history) for seeds or referents.

In order to illustrate this, Figure 5.0 contains an (admittedly contrived) example.

Suppose that an expert system is used to aid students in a beginning programming class. Students are able to ask the system questions about their lessons in one-on-one problem-

solving sessions. The system can obtain its list of objectives from the topics the student is currently studying in class. In this example, the system knows that the student has learned about top-down programming and might ask about it. The student asks a general question about debugging. If the algorithm to identify seeds was sufficiently intelligent, it might know that debugging is associated with bugs, which is associated with bad program design, which is associated with top-down-design. It might mention design methodologies, knowing that when the student later inquires about top-down design, it can refer to that seed and say that top-down design is a specific example of design methodology in general.

User: How do you debug a program?

System: There are several methods. If a debugger is available, it can be especially useful, because it allows you to step through code line-by-line and see the contents of memory. If no debugger is available, common methods include outputting the status of certain variables at various points to see when they change, and isolating functions to ensure they work correctly. Of course, putting a good deal of thought into the design of a program can significantly reduce the occurrence of bugs down the line.

...elapsed time...

User: What is top-down design?

System: Remember when we were talking about the importance of designing a program before hand? Top-down design is a specific example of a design methodology that involves decomposing a program hierarchically starting from the highest level.

Figure 5.0 – An example dialogue.

While this example is very domain-specific and clearly doctored, it is intended to illustrate the point that seeding can be used for many types of references. Relying again on the assumption that references make a discourse more coherent, it is reasonable to think that the seeding concept would have utility in any system that will generate extended task-related discourse.

It is not inconceivable that these methods could be employed outside of the domain of discourse planning altogether. The seeding concept is not that dissimilar from the idea of foreshadowing in narrative. A narrative planner that is a component in an interactive virtual world might be able to employ seeding as a form of foreshadowing. Interactive narrative is a relatively new medium that involves a user playing the role of a character in a dynamically-generated story. Most interactive narrative systems are still in the research stage, but they generally include a planner that is specifically designed to craft a story from the user's interactions with the environment. In this scenario, the narrative planner may know that certain events will need to occur in the world in order to create an aesthetically pleasing story. Knowing that certain actions are going to be included in the plan later, the planner can use seeding in order to subtly influence the user to perform these actions.

6. Future Work

Many performance and usability issues have surfaced over the course of this research.

While this research makes no attempt to resolve them, what follows is a list of concerns that are likely candidates for future work in this area.

The first is the issue of overall repetitiveness of the discourse generated using the seeding and referring method. Under ideal conditions, with an appropriately robust action specification and a competent plan-to-natural-language translator, repetitiveness of the generated discourse would probably not be an issue. However, the seeding and referring methods outlined above show no regard for the number of times a particular seed is introduced, which could in many circumstances lead to very redundant dialogue. User modeling is one way to circumvent this potential problem. One system in research today that addresses this issue is the University of Edinburgh's Intelligent Labeling Explorer (ILEX) system [7].

ILEX is a system designed to create captions (labels) for dynamically generated web pages containing text and graphics for an online museum gallery. The labels are tailored to each user based on their experience, interests, and the course of their visit to the gallery. ILEX stores a history of what labels a user has seen and uses it to make its presentation less repetitive and more context-sensitive, to reinforce, compare, or contrast concepts, and to maintain a model of the user's knowledge. ILEX annotates its database of museum facts with a number called an "assimilation quotient" that represents how salient each fact is for the user. Saliency is a concept related to a user's short-term memory. If a fact is salient for a user, it is readily available in memory. Saliency is

directly related to how many times a user has seen a piece of information, and for that reason ILEX's assimilation quotient is based on the number of times a fact has been presented. The assimilation quotient directly addresses the concern of presenting information too many times.

A system implementing seeding and referring could greatly benefit from an approach similar to the one taken by the ILEX designers that incorporates a user model. User modeling is an attempt to maintain an on-going representation of the user's knowledge. One way to increase the intelligence (and decrease the redundancy) of the seeding process is to take into consideration the user's current (assumed) knowledge when deciding whether or not to introduce a seed into the dialogue.

Another related issue is that of inconsistent dialogue. In theory, all of the seeds "planted" by the methods above will eventually be referred to in subsequent responses. This theory follows from the assumption that the objectives for a session will be well-chosen and will indeed reflect the most important pieces of information to be related. In practice, the objectives for a session might be less accurate. If a plan-recognition module generates them, the objectives are especially likely to be incorrect at the beginning of the discourse. Unreliable or under-specified objectives could lead to inconsistent dialogue. Seeds that are never referenced later (due to changing or misplaced objectives) may appear to the user to be inappropriate or out of context.

There are many possible avenues of exploration into this problem. In order to determine how much of a problem it really is, research should be conducted to see how much inconsistency a user notices and at what point she begins to notice it. An IM-centric

approach to solving the problem might involve monitoring the effectiveness of the seeding process over the course of a dialogue and performing adjustments. Another approach would involve improving the module that produces the objectives or compensating for misplaced objectives in some way.

The use of seeding and referring methods like those in this paper requires a plan-to-dialogue translator that is aware of these concepts. Identifying when such a construct can be introduced into a plan is just the first step. Next the seed or reference must be artfully embedded in the system's response. Doing this in a natural way can present a very complex problem. The output for our system is based on pre-specified templates that include seeds or references. For a more dynamic system, a grammar or translator would have to be able to manipulate seeds and references in a natural way. This is a problem that involves HCI concerns such as overall dialogue coherency and maximizing the information-processing abilities of a user, as well as linguistics issues such as optimal groupings of tokens in a string of utterances.

The goals of this research involved the quality of produced discourse plans rather than optimizing the performance of the planner. As a result, the effect of these methods on the planner's performance has not been quantified. A shallow worst-case analysis follows: if every action in the planner's library had four associated decompositions to handle seeding and referring, then the average branching factor of the search through the plan-space would be multiplied by 4. If only one action could be used to resolve a particular open precondition, then a planner using a standard library would create only one plan. A planner using a seeding and referring library would generate 4 plans in this case

(assuming the seeds and references are possible). If two actions could be used to resolve the precondition, 8 plans would be possible.

A four-fold increase in the branching factor would surely affect the performance of the planner, however, this is not going to happen in the average case. First, it is unreasonable to assume that every action in the planner's library would have all four types of decompositions associated with it. For any library incorporating sufficient domain knowledge, seeds and references would not be possible for many actions. Certainly some actions would be primitive and have no decompositions associated with them at all. Second, throughout the planning process the planner will attempt to resolve three types of flaws: open preconditions, unexpanded composite steps, and threatened causal links. Not every flaw will require the planner to add a new step to the plan. In some cases, the planner will attempt to satisfy a precondition using a step that is already included in the plan rather than by adding a new one. This possibility will occur jointly as well as the possibility of adding a new step, and both will be explored, but in this case the branching factor will not be as heavily impacted. Resolving the third type of flaw, a threatened causal link, does not involve adding any new steps to the plan and thus should not be affected by the changed planning library.

The inclusion of seeding and referring methods in a planning library warrants further research into the effect on the planner's performance. If it is found that the speed of the planner is significantly decreased, future work in this area might be able to mitigate the negative effect.

Finally, Section 4 identifies the need for a more extensive user study that tests knowledge retention and the possible negative effects of including seeds and referrals in dialogue. A particular issue that should be addressed is the frequency of seeds and referrals in dialogue and how it affects the subject's perception of the system and the effectiveness of the references. Other avenues of exploration should include a more calculated study of knowledge retention taking into account variables such as the quality of the text generation, the different types of references (repetition, comparison, clarification, etc) utilized, and the skill level of the user. In particular, our user study results indicated users who are less knowledgeable about a particular subject may favor seeding and referring more than advanced users. This is appropriate given that tutorial systems are generally designed to be used by students who are novices in a particular area.

7. Conclusions

This research introduces the idea of seeding and referring - an idea that has been relatively unexplored in current research up to this point. We have demonstrated its feasibility and potential by implementing a modest test system, and we have obtained encouraging results from an informal user study. We feel that seeding and referring is a concept that merits further exploration and implementation. The idea holds promise as a learning tool, and also simply as one part of a repertoire of conversational techniques for increasing the naturalness of computer-generated dialogue.

The methods outlined here are specifically designed to take advantage of Longbow's planning capabilities. They have not been implemented using any other planner, however, it is reasonable to think that the idea of coding seeds and references into an action library can be adapted to work with any planner. Obviously, the process is better suited to a decompositional planner. The planner should also be capable of interfacing with a knowledge base in order to satisfy constraints and preconditions at run-time.

These methods have also been described and implemented with particular attention given to tutorial and task-related dialogue systems. These systems could benefit from using seeds and referrals effectively because they typically contain conversational agents with the goal of producing dialogue that is both natural and instructive. However, it is not infeasible to think that other types of systems (like those mentioned in the Applications section) might also be able to employ this concept.

In the future we can expect the quality of text generation to improve and its computational expensiveness to decrease as computer technology advances. As more potential uses of natural language generation are discovered, we anticipate more emphasis being placed on the naturalness and usefulness of conversational agents rather than the speed and practicality of generation methods. This research into seeding and referring is one example of the types of research we expect will become more prominent with time.

8. Bibliography

1. J. F. Allen, L. Schubert, G. Ferguson, P. Heeman, C. Hwang, T. Kato, M. Light, N. Martin, B. Miller, M. Poesio, and D. Traum. *The Trains project: A case study in building a conversational planning agent*. Journal of Experimental and Theoretical AI 7:7—48, 1995.
2. D.E. Appelt. *Planning English Sentences*. Cambridge University Press, Cambridge, England, 1985.
3. Ardissono, L. and D. Sestero. *Using Dynamic User Models in the Recognition of the Plans of the User*. User Modeling and User-Adapted Interaction. 5(2):157-190, 1996.
4. B.J. Grosz and C.L. Sidner. *Attention, Intentions and the Structure of Discourse*. Computational Linguistics, 12(3), 175-204, 1986.
5. B.J. Grosz and C.L. Sidner. *Plans for Discourse*. P.R. Cohen, J.L. Morgan, and M.E. Pollack (eds.): Intentions and Communication. Cambridge, MA: MIT Press, Chapter 20, pp. 417-444, 1990.
6. C.G. Healey, R. St. Amant, and M. Elhaddad. ViA: A Perceptual Visualization Assistant. In *Proceedings 28th Applied Imagery Pattern Recognition Workshop*, (Washington, D.C., 1999), pp. 1-11.
7. J. Hitzeman, C. Mellish and J. Oberlander. *Dynamic Generation of Museum Web Pages: The Intelligent Labelling Explorer*. Archives and Museum Informatics, 11:107--115, 1997. Also presented at the Museums and the Web Conference, Los Angeles, March 1997.
8. E.H. Hovy. Approaches to the Planning of Coherent Text. In Paris, Cecile L., Swartout, William R., and Mann, William C., Eds., *Natural Language Generation in Artificial Intelligence and Computational Linguistics*. Kluwer Academic Publishers, Boston, 83-102. 1991.
9. N. Lesh, C. Rich, and C. Sidner. *Using Plan Recognition in Human-Computer Collaboration*. Technical Report TR98-23, MERL, 1998.
10. D.M. Levy. Communicative Goals and Strategies: Between Discourse and Syntax. In Cole, P. and Morgan, J.L., Eds., *Syntax and Semantics, Volume 12: Discourse and Syntax*. 183 – 210. Academic Press, New York, NY. 1979.
11. E. Maier. *Context Construction as Subtask of Dialogue Processing: The Verbmobil Case*. Proceedings of the Eleventh Twente Workshop on Language Technology,

- TWLT 11, 1996.
12. J. Moore and C.L. Paris. *Planning text for advisory dialogues: Capturing intentional and rhetorical information*. Computational Linguistics, 19(4):651--695, 1993.
 13. J. Moore and M. Pollack. *A problem for RST: The need for multi-level discourse analysis*. Computational Linguistics, 18(4):537-544, 1992.
 14. C. Rich and C. Sidner *COLLAGEN: A collaboration manager for software interface agents*. User Modeling and User Adapted Interaction, 8(3-4), 315-350, 1998.
 15. J.A. Rosenblum and J.D. Moore. *Participating in instructional dialogues: finding and exploiting relevant prior explanations*. Proceedings of the World Conference on AI in Education, (Edinburgh), 1993.
 16. M.A. Walker. *The effect of resources limits and task complexity on collaborative planning dialogue*. Artificial Intelligence, 85(1), 1996.
 17. R.M. Young and J. Moore. *Does discourse planning require a special-purpose planner?* Proceedings of the AAAI Workshop on Planning for Inter-Agent Communication, 1994.
 18. R.M. Young and J. Moore. *DPOCL: A Principled Approach to Discourse Planning*. Proceedings of the Seventh International Workshop on Natural Language Generation, Kennebunkport, 1994
 19. R.M. Young, J.D. Moore, and M.E. Pollack. *Towards a Principled Representation for Discourse Plans*. In Proc 16th Annual Meeting of the Cognitive Science Society, p. 946-951, 1994.

Appendix A – User study materials

Car-demo system user evaluation –Drive Train example

1. Rate your prior knowledge about cars:
 Very knowledgeable
 More than average
 Average
 Less than average
 Very unknowledgeable
2. How effective (in terms of conveying information) would you say the responses were at answering the queries?
 Always effective
 Usually effective
 Sometimes effective
 Rarely effective
 Never effective
3. How coherent was the dialogue over the course of the whole interaction (from one question to the next)?
 Always coherent
 Usually coherent
 Sometimes coherent
 Rarely coherent
 Never coherent
4. What are some of the key components of the drive train?
5. Why is the transmission's function necessary?
6. What part does the transmission control?
7. How consistent would you say the dialogue was within individual responses?
 Always consistent
 Usually consistent
 Sometimes consistent
 Rarely consistent
 Never consistent
8. How often did you feel that a response contained extraneous information?
 Always
 Usually
 Sometimes
 Rarely
 Never
9. How repetitive would you say the system was?
 Always repetitive
 Usually repetitive
 Sometimes repetitive

- Rarely repetitive
- Never repetitive

10. Do you feel that repetitiveness in this type of system is annoying?
11. For the query “How are the gears related to the transmission?” the response was: “The transmission is responsible for regulating the gears in order to maintain the proper number of tire rotations per minute for a particular driving situation. The tires rotate on the axle.” Did you find the last statement, “The tires rotate on the axle”, to be extraneous?

12. In response to the query “What are the gears?” the system generated the dialogue “The gears are mechanical disks that are connected to the car’s tires by rods. The gears turn the tires, and the transmission controls the gears. As I mentioned before, the engine, transmission, and gears are key components of the drive train.” At the time, did you notice the last statement (“As I mentioned before...”) as being out of place or unnecessary?

Car-demo system user evaluation –Electrical system example

7. Rate your prior knowledge about cars:
- Very knowledgeable
 - More than average
 - Average
 - Less than average
 - Very unknowledgeable
8. How effective (in terms of conveying information) would you say the responses were at answering the queries?
- Always effective
 - Usually effective
 - Sometimes effective
 - Rarely effective
 - Never effective
9. How coherent was the dialogue over the course of the whole interaction (from one question to the next)?
- Always coherent
 - Usually coherent
 - Sometimes coherent
 - Rarely coherent
 - Never coherent
10. What are some of the key components of the electrical system?
11. What do the spark plugs do?
12. How does the alternator produce current?
7. How consistent would you say the dialogue was within individual responses?
- Always consistent
 - Usually consistent
 - Sometimes consistent
 - Rarely consistent
 - Never consistent
8. How often did you feel that a response contained extraneous information?
- Always
 - Usually
 - Sometimes
 - Rarely
 - Never
10. How repetitive would you say the system was?
- Always repetitive
 - Usually repetitive
 - Sometimes repetitive
 - Rarely repetitive
 - Never repetitive
10. Do you feel that repetitiveness in this type of system is annoying?

Appendix B – User Study Results

Drive Train Example

	Q1*	Q2*	Q3*	Q4**	Q5**	Q6**	Q7*	Q8*	Q9*	Q10***	Q11****	Q12****
User 1	3	4	3	Y	Y	Y	2	1	2	Y	Y	Y
User 2	3	2	1	N	N	N	2	3	4	Y	Y	N
User 3	3	2	3	N	N	Y	3	3	2	N	N	Y
User 4	4	2	2	N	Y	Y	2	3	3	Y	N	Y
User 5	4	1	2	Y	N	Y	1	4	3	N	N	N
User 6	5	3	2	N	N	Y	2	2	2	Y	Y	Y
User 7	4	1	1	Y	Y	Y	1	4	4	N	N	Y
User 8	2	2	2	Y	N	N	2	3	3	N	Y	Y
User 9	3	3	4	N	N	Y	3	3	4	Y	N	Y
User 10	3	4	4	Y	Y	Y	2	4	2	N	Y	Y
User 11	5	1	1	N	N	N	1	5	1	Y	N	N
Average	3.5	2.27	2.3				1.9	3.18	2.73			

Electrical System Example

	Q1*	Q2*	Q3*	Q4**	Q5**	Q6**	Q7*	Q8*	Q9*	Q10***	Q11****	Q12****
User 1	3	3	4	Y	Y	Y	2	1	2	Y	N/A	N/A
User 2	3	2	2	Y	Y	N	1	3	3	N	N/A	N/A
User 3	3	3	3	Y	Y	Y	2	4	3	N	N/A	N/A
User 4	4	2	4	N	Y	N	3	3	3	Y	N/A	N/A
User 5	4	2	2	Y	Y	N	1	4	4	N	N/A	N/A
User 6	5	2	2	N	Y	N	2	5	4	N	N/A	N/A
User 7	4	1	1	Y	Y	Y	1	5	5	N	N/A	N/A
User 8	2	2	2	Y	Y	Y	2	4	3	Y	N/A	N/A
User 9	2	3	2	Y	Y	N	2	3	4	N	N/A	N/A
User 10	3	3	4	Y	Y	Y	4	3	4	N	N/A	N/A
User 11	5	1	1	Y	N	N	1	5	1	Y	N/A	N/A
Average	3.5	2.18	2.5				1.9	3.64	3.27			

* Questions 1, 2, 3, 7, 8, & 9 - were measured on a scale of 1-5, with 1 receiving the greatest quantification (very knowledgeable, always repetitive, etc), and 5 receiving the least quantification (very unknowledgeable, never consistent, etc)

** Questions 4, 5, & 6 - For the drive train example, the responses were marked with a Y if subject gave the same answer that was delivered to them by the Garage system as a second answer was technically correct but not what was delivered by the system, or if it was in the answer was marked as N. For the electrical system, answers that were correct according to the information presented by the Garage system were marked as Y, and all others as N.

*** Question 10 - If the subject responded that he or she either found repetitiveness to be annoying, or stated that it "could be" annoying, it counted as a Y, otherwise, an N.

**** Questions 11 & 12 - If the subject responded with a yes or a possible yes ("maybe," "possibly"), the answers were marked with a Y, otherwise, they were marked with an N.