2006

Conversations with an intelligent agent: modeling and integrating patterns in communications among humans and agents

John Ray Lee
University of Iowa

CONVERSATIONS WITH AN INTELLIGENT AGENT: MODELING AND INTEGRATING PATTERNS IN COMMUNICATION AMONG HUMANS AND AGENTS

by

John Ray Lee

An Abstract

Of a thesis submitted in partial fulfillment
of the requirements for the Doctor of
Philosophy degree in Electrical and Computer Engineering
in the Graduate College of
The University of Iowa

July 2006

Thesis Supervisor: Adjunct Assistant Professor Andrew Williams

ABSTRACT

There is an overwhelming variation in the ways an intelligent agent can rationalize communication with a conversational partner. This variation presents many incompatibilities that lead to the specialization of conversational capabilities. This has produced a plethora of models and ideas on how an intelligent agent should understand, interact with, and incorporate communication from a human conversational participant. This dissertation approaches this problem with the thesis that there exists a language between that of human natural language and the behavioral reasoning of an intelligent agent, and that this language is capable of not only unifying the various models used in literature, but also provides the foundation for a theoretical framework for an engineering methodology for building such models.

A theory of practical communication language is developed, including the introduction of the meaning-action concept, an expressive and powerful representation based on speech-act and dialogue-act theories, but extended with notions of behavioral operators as well as signatures that allow the operators to incorporate structured and well-defined concepts. An engineering methodology is presented for the construction of concepts, operators and rules that create the language and model of a specific domain, including methodology for the verification and validation of that language and model.

The resultant practical communication language methodology, based on the combination of rational communication and meaning-action concepts, will introduce several major enhancements to dialogue management. These enhancements include the use of meaning-action concepts as a shared medium and the introduction of a shared concept graph. This methodology will be used along with various dialogue models from human-human, human-agent and agent-agent communication to construct a task-oriented language and model called the task communication language framework. This framework is then implemented within an intelligent agent in a real-time resource management simulation.

A sample output listing from actual human interaction with that implementation is used to demonstrate that the resulting framework does indeed incorporate many of the disparate models of communication and their corresponding capabilities including command and control, information seeking, notification and bother, clarification, explanation, discussion, negotiation, mutual planning, interruption, feedback, adjustable autonomy and corrective dialogues.

Abstract Approved: _____
Thesis Supervisor

_____
Title and Department

_____
Date

CONVERSATIONS WITH AN INTELLIGENT AGENT: MODELING AND INTEGRATING PATTERNS IN COMMUNICATION AMONG HUMANS AND AGENTS

by

John Ray Lee

A thesis submitted in partial fulfillment
of the requirements for the Doctor of
Philosophy degree in Electrical and Computer Engineering
in the Graduate College of
The University of Iowa

July 2006

Thesis Supervisor: Adjunct Assistant Professor Andrew Williams

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

_____

PH.D. THESIS

_____

This is to certify that the Ph.D. thesis of

John Ray Lee

has been approved by the Examining Committee
for the thesis requirement for the Doctor of Philosophy
degree in Electrical and Computer Engineering at the July 2006 graduation.

Thesis Committee: _____
Andrew Williams, Thesis Supervisor

_____
John P. Robinson

_____
Thomas L. Casavant

_____
Jim Cremer

_____
Dan Thedens

To Joni, my inspiration, my strength and my love.

With God, All Things Are Possible

Mathew 19:26
The Bible (NIV)

ACKNOWLEDGMENTS

I would like to thank my wife for standing by me, my advisor for advising me through the paths of life, my family for always supporting me and my friends for making it fun.

ABSTRACT

There is an overwhelming variation in the ways an intelligent agent can rationalize communication with a conversational partner. This variation presents many incompatibilities that lead to the specialization of conversational capabilities. This has produced a plethora of models and ideas on how an intelligent agent should understand, interact with, and incorporate communication from a human conversational participant. This dissertation approaches this problem with the thesis that there exists a language between that of human natural language and the behavioral reasoning of an intelligent agent, and that this language is capable of not only unifying the various models used in literature, but also provides the foundation for a theoretical framework for an engineering methodology for building such models.

A theory of practical communication language is developed, including the introduction of the meaning-action concept, an expressive and powerful representation based on speech-act and dialogue-act theories, but extended with notions of behavioral operators as well as signatures that allow the operators to incorporate structured and well-defined concepts. An engineering methodology is presented for the construction of concepts, operators and rules that create the language and model of a specific domain, including methodology for the verification and validation of that language and model.

The resultant practical communication language methodology, based on the combination of rational communication and meaning-action concepts, will introduce several major enhancements to dialogue management. These enhancements include the use of meaning-action concepts as a shared medium and the introduction of a shared concept graph. This methodology will be used along with various dialogue models from human-human, human-agent and agent-agent communication to construct a task-oriented language and model called the task communication language framework. This framework is then implemented within an intelligent agent in a real-time resource management simulation.

A sample output listing from actual human interaction with that implementation is used to demonstrate that the resulting framework does indeed incorporate many of the disparate models of communication and their corresponding capabilities including command and control, information seeking, notification and bother, clarification, explanation, discussion, negotiation, mutual planning, interruption, feedback, adjustable autonomy and corrective dialogues.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

This dissertation addresses the overwhelming variety of ways an intelligent agent can rationalize communication with a conversational partner, be it a human or an agent. Currently, this variety presents many incompatibilities that often lead to the specialization of conversational capabilities. However, this dissertation argues the thesis that there exists a language between that of human natural language and the behavioral reasoning of an intelligent agent, and that this language is capable of not only unifying the various models used in literature, but also provides the foundation for a theoretical framework for an engineering methodology for building such models. The theoretical aspects of this dissertation are demonstrated through a proof-of-concept model by incorporating a conversational intelligent agent inside a resource-management simulation.

Motivation

The rise of the personal computer has forever changed the way our society operates. Businesses are forced to adopt software processes in order to stay productive and competitive; and employees must have basic computer training and knowledge to be marketable. Moreover, the internet is changing the foundations upon which information and services are exchanged.

As technology in the information age continues to revolutionize our world, computers will become not only commonplace but also embedded in everything around us and our dependency on them will continue to strengthen. A separation of classes has already begun and those that do not embrace the technology will be left behind.

The intelligent agent is the one of the current leading paradigms, changing how we interact with computer systems. As computer-driven applications and services are becoming increasingly sophisticated, interface agents are developed to manage the overwhelming complexity while presenting a simple, intuitive and often collaborative interface. The assistive agent will carry out tasks delegated to it on the user's behalf.

Embodied agents will handle the behavior of humanoid robotics as they begin to be placed in the home to assist the elderly and disabled with activities of everyday life.

The ongoing pursuit of intelligent agents that can successfully operate in real life circumstances must be accompanied by advancing the means at which the average human can communicate with and utilize these agents. As intelligent agents are absorbed into everyday life, it is essential to make this human-agent interaction rich and complex but yet natural and commonplace. This dissertation focuses on one promising interaction paradigm, that of natural language.

Natural language understanding is in itself an extremely complex subject, inspired by linguistics, computer science, mathematics and psychology. Therefore, this dissertation further narrows its scope to that area that falls between language and behavior; and the understanding as expressed in both knowledge and reasoning inside an intelligent agent. This will be further defined in the next chapter. The remaining sections of this chapter will focus on conducted research and its importance and contribution to knowledge.

Problem

Human language and communication technologies play a key role in enabling humans to interact with agents in a natural and intuitive way. Although many research groups have great ambitions for human agent communication, there is a lack of complex multi-modal language skills in implementation. Currently, most communicative agent systems recognize specific keywords in incoming speech that then lead to the execution of predefined actions and procedures. Some allow the imposing of constraints, beliefs or planning; and others use multimodal interaction to allow gestures and pointing to accompany commands.

Research in task-oriented dialogue understanding and modeling has produced a plethora of models and ideas on how an intelligent agent should understand, interact with,

and incorporate communication from a human conversational participant. Examples of such models allow for action and plan recognition, procedure learning, intention recognition, collaborative planning, mixed initiative behavior, negotiation and more. Even though a couple of these models have been introduced into domain-specific research prototypes, the majority of them are based on either agent-agent communication, or are simply theoretic in nature, based only on a few examples of dialogue. Furthermore, these models have not been successfully integrated into a working human-centered, intelligent agent implementation. This is somewhat due to the lack of implementation platforms as well as current deficiencies of speech-act and dialogue-act recognition, which reflects upon the enormous complexities of the human language.

The success of a dialogue management system is dependent almost entirely on experience; where as in [34], experience is defined as the history of previous mistakes that one is not likely to make more than once. What is lacking here is a clear methodology and set of objective tools for the design and development of dialogue systems. By applying the engineering discipline to dialogue systems, a methodology and toolset can be developed so that the lack of experience is no longer a limiting factor. Furthermore, this methodology will allow for the extensibility, interoperability and maintenance of the dialogue system throughout the phases of design, validation, verification and testing.

The goal of this dissertation is not only to demonstrate a common foundation that unifies all dialogue models but also the construction of a well-founded and manageable yet extensible framework for conversational modeling inside the behavior of an intelligent agent. Certain fundamental questions will be addressed: How should an intelligent agent incorporate communication? How does communication and behavior integrate within an agent model? How can ideas from many different dialogue models and conversation examples be incorporated together? How can one validate the correctness of an agent conversational model?

Approach and Methodology

This dissertation represents the research behind developing a unifying architecture for the representation and incorporation of communication within the behavior of an intelligent agent, providing a foundation on which many of the behavioral aspects can be modeled and expanded. One of the keywords in the above sentence is unifying, not universal. The very nature of behavior is not known to be universal, and thus there can be no universal architecture. Therefore, this solution is not a cure-all, but rather is an approach to the development of engineering principles as well as a proof of concept of a single unifying language.

A theory of practical communication language, as well as the integration of communication and behavior has been outlined in chapter 3. This chapter also provides a strong and formal engineering methodology that is rooted in the fields of protocol engineering, as well as the mathematical foundations of rational agency, logics and speech-act theory. This theory is then followed in chapter 4 where the task communication language is defined as a subset of practical communication language. This chapter is extended by appendix A with a partial specification of the concepts and operators that make up the task communication language.

The developed methodology is followed in appendix B to produce a working conversational engine and an accompanying toolset, which is embedded into a simple intelligent agent. This agent is incorporated within Stratagus [55], a real-time strategy, resource management simulation. An example session with a human participant is provided in appendix C. Chapter 5 uses this session to demonstrate the many conversational capabilities of the task communication language as well as their unification, thus providing evidence supporting the thesis. Chapter 6 follows with a discussion of the implications of the thesis as well as its relation to various research fields.

The research in this dissertation focuses on how an agent understands and reacts to speech acts and related concepts, rather than how these acts are generated or interpreted from text or speech. Thus, this dissertation will not focus on any of the linguistics aspects of the interpretation mechanism. However, various insights will be offered on possible approaches.

Contribution to Knowledge

Practical communication language based techniques and methodologies provide a framework for the development of highly communicative and collaborative intelligent agents. These techniques enhance the capabilities of future agents. TCL techniques facilitate the development of complex dialogue models, focusing on how communication, behavior and task conception interconnect. TCL represents the elegant marriage of a task-based design approach to agent behavior modeling and a speech-act and dialogue-act based approach to communication and interaction design.

The developed mechanisms not only illustrate how communication affects reasoning and knowledge inside an agent; they also touch upon concepts such as autonomy, user modeling and preferences, learning, and more. TCL also illustrates how various agent-agent communication models can be transformed into human-agent models, thus allowing many known models to be reused and unified. The TCL vocabulary produces reusable classifications and algorithms, which can be applied to a number of dialogue systems.

The applied methodology extends on many interesting concepts of protocol engineering, such as partial specification, fault-tolerance, probabilistic sequencing, multi-threaded sequencing, interrupt-ability, self-correcting models, adaptive models and context-driven models. These extensions provide valuable insight into how protocol engineering methodologies might incorporate these concepts.

Results & Significance of Work

The successful integration of multiple dialogue models using meaning-action concepts will further expand applications that use natural language processing. The interface for meaning-action concepts will assist in the separation of language processing and other communication types (i.e. gesture recognition) from the implementation of the meanings of those utterances or gestures. This plays a key role in the theory of the separation between communication and behavior. This separation reduces the cohesion between dialogue interpreters and intelligent agents, thus allowing separate, yet parallel development of each. This separation provides for both interoperability, and a development process far above current techniques in bringing communication-enabled intelligent agents to market.

The application of the engineering methodologies developed an assistive agent with advanced behavior management and control, as well as other sophisticated communicative capabilities, through an intuitive human-machine interface. This demonstrates an agent that drastically improves the productivity and capabilities of novice users when interacting with computer systems and applications, as well as a proof-of-concept for improving upon the current features of intelligent agents and assistive robotics.

CHAPTER 2
BACKGROUND

Philosophy has been one of the major influences of artificial intelligence since its birth. Just as geometry, in which two points make a line, and three points make a plane; this chapter will begin by introducing several philosophical concepts that form a foundation of a particular research area. As these concepts are expanded upon, both historical context and technical details will continue to bring this research area into understanding. This chapter represents all the background material necessary to understand the research presented in this dissertation.

Following geometry, two philosophical concepts can be used to make a line. Even though this line need not be straight, as those concepts may not be readily connectable, that line can be turned into a spectrum. This chapter defines a spectrum that will help to organize and connect all of the material covered in this, as well as subsequent chapters.

The Communication – Behavior Spectrum

The communication-behavior spectrum begins with two concepts. The first concept was introduced by John Austin in 1962, when he wrote a book called *How to Do Things With Words*. This book had a simple idea that *utterances*, or atomic communicative phrases, not only carry meaning, but also perform an action. The investigation of this idea led to the development and subsequent categorization of the *speech-act*. This concept creates a specific point within the notion of communication. The next section will start with this point and follow it toward the notion of behavior. However, before the area of communication is explored, it is important to get a good idea of the destination.

The second concept is the notion of agency. The introduction of the *agent paradigm* has had a lasting impact on various areas within artificial intelligence, ranging from complex software systems to societies of intelligent beings. There are thousands of

definitions for the term agent; both formal, as supported by rich semantics and logics, to informal, as found in fields from sociology to virtualization. One important aspect of agency is its ability to encapsulate the behavior of an entity. It is used in many aspects of behavior modeling, as well as the primary paradigm to embody software-based behavior. A section on agency will not only define the notion of behavior using an intelligent agent, but will also pick up the section on communication, further connecting the notions of communication and behavior. Then a section on agents and communication will further expand the marriage of communication and behavior by discussing how agents use communication in and among themselves, as well as with human conversational participants.

Two critically important aspects create the distinction between a spectrum and a simple line. The first aspect defines the spectrum in such a way as to place the human on the communication end and the agent on the behavior end. This is because communication is the primary point of contact between a human and an agent. Although this is not necessarily true, as the agent can use the observation of human behavior as a point of contact, it is the role of this dissertation to focus on communication.

The second aspect of the spectrum builds upon the first. The human uses natural language to communicate. A language that is not only overwhelmingly complex, but also constantly fluctuating. On the other side, the agent is situated within a computer system, an environment that is precisely defined in mathematics using semantics and logics. Therefore, because abstraction is a key tool in simplification, it is used throughout the spectrum to take uncertainty to certainty, or incomprehensible to rational. A section on empiricism and rationalism further expands the understanding of this aspect of the spectrum by describing various scientific views and methodologies associated with the perspectives from each side.

The last section concludes the chapter by providing an overview of the field of protocol engineering. It describes various tools, models and methodologies pertaining to

the design, specification, verification and testing of communication protocols. The theory presented in this section will provide a necessary background to the understanding of the developed framework and methodology applied to the communication-behavior spectrum.

Communication: Language Theory

This section will highlight some of the important concepts of the language theory foundation of this research. The purpose of this section is to understand how the meaning-action concept comes from and fits within this theory.

Speech Acts

Speech act theory was attributed to Austin [5] when he wrote a book on the premise that when a person says a particular statement, that statement has an impact on the speaker or the hearer and thus changes or manipulates the environment in which the speaker is situated. Therefore, the speaker is able to carry out an action by merely speaking a particular utterance or sequence of utterances. What followed was a great number of taxonomies of speech-acts and their categorization as well as the linguistic analysis of many corpuses to inquire about the frequency and probability of speech acts. Speech acts have been used to recognize a particular author over another, categorize email, recognize spam and filter web pages as well as attempt to detect one's identify, ethnicity and cultural background.

Speech act taxonomies have three top-level categories:

- *Locutionary acts*, in which the utterance has particular meaning.
- *Illocutionary acts*, in which the speaker is committing, asking about or answering.
- *Perlocutionary acts*, in which the speaker intends to cause feelings or thoughts.

Generally, the study of speech acts with respect to practical language and agents has focused on illocutionary acts only. In fact, in those particular fields, the term speech act has been somewhat distorted to encapsulate only illocutionary acts.

One such categorization of illocutionary acts uses five core groups [50]:

- Assertives commit the speaker to a belief (proposing, concluding.)
- Directives attempt to provoke action in an addressee (asking, inviting.)
- Commissives commit the speaker to future events (planning, promising.)
- Expressives demonstrate psychological state of the speaker (apologizing.)
- Declarations bring about a change in the world (declaration.)

Conversation Acts [56] augment traditional speech acts with acts associated with turn taking, grounding and argumentation.

Dialogue Modeling

Dialogue modeling [20] is an attempt to understand sequences of utterances rather than individual utterances themselves. This pursuit led to the creation of dialogue acts and discourse structure. The general process for creating a dialogue model is as follows:

1. Conversation is generated or recorded and becomes a corpus.
2. Special discourse tags are generated and used to annotate the corpus.
3. A model is built which operates on the generated tags.
4. The original conversation is applied to validate and demonstrate the model.

The majority of dialogue models are created in theory, attempting to account for real-world problems and situations in an attempt to understand conversation. Only a few of these dialogue models are integrated into an agent implementation.

There are also generally two types of models: strong models, which are built through the process above; and weak models, which are built by applying dialogue sequences, often only made up simple examples, which break a given strong model and then extend that model to account for the exploited weakness.

Dialogue Acts

Research in building dialogue models has created multiple-function acts that are more complex than speech acts. These acts are referred to as dialogue acts [20].

Dialogue acts not only replicate the illocutionary power of speech-acts but also provide conditions on how they connect with one another, not only to specify constraints, but also to develop structure within a dialogue.

Many dialogue tag sets are used in current research from speech understanding systems to the generation of agent communication languages. These tag sets all attempt to describe the semantics of various utterances in a dialogue between two or more participants. The semantics are justified by tracking the utterances connected together to make up segment of a discourse. Models are associated with these sets in order to describe the process of conversing in a dialogue.

Creating Dialogue Tags

Early dialogue tag sets were created by an attempt to common sense the English language into obvious tags. The trouble with this approach is that there are thousands of tags, many of which are used only in very specific cases. Today, dialogue tag sets are generally constructed to focus on a particular type of discourse most applicable to a specific purpose. These tag sets are created by analyzing corpuses specifically pertaining to a domain. It is often the case that these corpuses will be collected through wizard-of-oz scenarios with users from a particular focus group.

The *wizard-of-oz* approach is to place an unseen human operator behind a figurative curtain. The human operator takes the place of a piece of software needed for the application to operate. A user interacts with this system thinking the system is working entirely on its own. The goal is to use the interaction context to model a computer system that will replace the human operator.

Dialogue tags are generally created by linguistically trained personnel that know about the application and what it is supposed to do. In order to understand these dialogue tag sets, some of the most widely used sets will now be discussed. They are presented in

an order that is not necessarily chronological, but so that each discussion may build off the previous.

LINLIN

LINLIN [22] is composed of three main types of utterances as seen in table 1. Initiatives begin a segment of conversation; responses finish a segment of conversation; in discourse management, the speaker performs conversational overhead.

Although extremely simple, this dialogue tag set already begins to form basic structures within a discourse. For example, updates assert knowledge; answers must follow questions. However, this tag set is only capable of modeling the most basic properties of simple discourse.

Table 1: LINLIN Dialogue Tag Set

- Initiative
  - Update
  - Question
- Response
  - Answer
- Discourse Management
  - Greeting
  - Farewell
  - Discourse Continuation

HCRC

Although HCRC [13] does not have the separate discourse overhead type of LINLIN, it still has the fundamental types of initiating and concluding a segment of discourse. HCRC is unique in that it treats the acts as a set of *dialogue moves* where the

conversation is likened to that of a game where each speaker is a player that is allowed to make certain moves during their turn at speaking.

The rules of this game are strict but simple. For example, a Query-yn, to ask a yes or no question, must be followed by a Reply-y, meaning yes, or a Reply-n, meaning no. The Query-w refers to a 'who, what, when, where, why' type question and should be responded with Reply-w. HCRC also demonstrates the further breakup of the tags found in LINLIN. Question is now two subtypes, and response is now three.

Table 2: HCRC Dialogue Tag Set

- Initiating Moves
  - Instruct
  - Explain
  - Check
  - Align
  - Query-yn
  - Query-w
- Response Moves
  - Acknowledge
  - Reply-y
  - Reply-n
  - Reply-w
  - Clarify
  - Ready

DAMSL

DAMSL [21] is an acronym for dialogue act markup in several layers. It begins to show the possible complexity of dialogue tag sets as seen in table 3. A hierarchy is used to break the acts down into more than one layer of groups, as is necessary to maintain order as the number of dialogue acts continue to expand.

Table 3: DAMSL Dialogue Tag Set

- Forward Looking Function
  - Statement
    - Assert
    - Reassert
    - Other-Statement
  - Influencing Addressee Future Action
    - Action-Directive
    - Open-option
    - Info-Request
    - Committing Speaker Future
    - Action
    - Offer
    - Commit
  - Conventional
    - Opening
    - Closing
    - Explicit-performative
    - Exclamation
    - Other Forward Function
- Backward Looking Function
  - Agreement
    - Accept
    - Accept-Part
    - Maybe
    - Reject-Part
    - Reject
    - Hold
  - Understanding
    - Signal-Non-Understanding
    - Signal-Understanding
    - Acknowledge
    - Repeat-Rephrase
    - Completion
    - Correct-Misspeaking
  - Answer
  - Information-Relation

In DAMSL, we still have the same basic principles of initiating and concluding a segment, but we extend much further beyond questions and answering to notions of proposals and commitment, as well as the acceptance or rejection of pieces of utterances as opposed to entire utterances. DAMSL is capable of modeling much more complex discourses and in further detail than previous tag sets.

TRAINS

Table 4: TRAINS Dialogue Tag Set

- Core speech acts
  - Inform
  - YNQ
  - WHQ
  - Request
  - Suggest
  - Offer
  - Promise
  - Eval
  - Accept
  - Reject
- Grounding acts
  - Initiate
  - Continue
  - Acknowledge
  - Repair
  - ReqRepair
  - ReqAck
  - Cancel
- Turn-taking acts
  - Take-turn
  - Keep-turn
  - Release-turn
  - Assign-turn

The valuable contribution of the TRAINS dialogue tag set [58], as seen in table 4, is the incorporation of the notions of grounding acts and turn taking. In grounding acts, participants of the conversation attempt to reach a mutual understanding within the discourse; thus accounting for sub-dialogues such as a clarification. In turn-taking acts, participants either keep or release their turn at speaking. This accounted for one of the first examples of a mixed-initiative dialogue, in which either participant can initiate utterances.

Dialogue Structure

All of the discourse tag sets above were created with an accompanying model. These models employ particular dialogue structure, whether it is based in protocols and sequences, such as the dialogue games of HCRC, or it is based on the understanding and interpretation of dialogue, as with TRAINS. The structure implies how individual utterances or segments are interconnected.

Countless researchers have analyzed discourse structure in a wide variety of domains. These have influenced many specialty models from negotiation to skepticism, founded agent communication languages and dialogue interpretation systems. It is well beyond the scope of this dissertation to provide any comprehensive background in this area. However, an extremely influential investigation of the structure of task-oriented discourse will be briefly presented.

Task-oriented discourse accounts for the dialogue pertaining to at least two participants. These participants focus only around a set of tasks, cooperating under a mutual objective. The majority of this type of discourse follows a unique layered structure in which individual segments are nested within one another. Almost all discourses have this type of layering property, however, it was the notion that each layer is associated with a particular purpose (task, objective, clarification, etc…) that helped to tie understanding components to the model [32].

According to the theory of discourse of [33], discourse structure is composed of three core components. The linguistic structure helps to explain how segments, sequences of utterances, occur naturally. The intentional structure captures purposes on the discourse level, expressed in the segments found within the linguistic structure. The attentional state abstracts the focus of attention; including what crosses the attention of the participants such as objects, properties and relations. These core components will be later addressed when several dialogue manager implementations are examined.

Meaning-Action Concepts

It is essential to separate the recognition and interpretation of communication as well as the discourse management from the behavioral aspects of an agent. Currently discourse managers perform this role, and dialogue-acts are used. However, the number of practical dialogue-acts is exploding.

A meaning action concept is the semantic and pragmatic meaning of utterances in a language-independent idiom free representation, based on the theory of speech and dialogue acts. A given utterance is translatable directly into a single or a set of meaning action concepts. However, unlike dialogue-acts, meaning action concepts can also relate directly to behavioral changes and actions to be taken within the system. Meaning action concepts are also associated with a signature, which assists in this endeavor, and they have the ability to be nested within one another. In addition, meaning action concepts also correlate directly to notions of shared interaction models, as will be explained later. In order to understand the notion behind a meaning action concept, an examination of the following utterances is provided.

"I want to go on a vacation." In LINLIN, this would be modeled simply as an update. In DAMSL, it would be considered a statement or perhaps even an exclamation and in TRAINS, it would be considered an inform. The critical aspect of this statement is that it conveys one of the speaker's desires. This may still be modeled as an inform of a

desire, but the meaning action concept will be directly tied and associated with the desire, rather than the general statement.

"Let's plan a trip to Germany." In LINLIN, this would be an update and in DAMSL a statement. TRAINS is more aligned with the true nature of the statement by modeling it as a suggestion. However, the meaning action concept will not only pick up the fact that the speaker is proposing some joint activity, but also that the activity is to generate a plan.

"What are you thinking?!?" As before, LINLIN would see this as an update and DAMSL a statement or perhaps an exclamation. TRAINS may see this as an inform. However, the true meaning behind this utterance is the assertion that the speaker is displeased with the choice or behavior of the other participant. The meaning action concept will model this as a scolding, which represents negative reinforcement learning, associated with displeased action on the participant's part.

Some dialogue managers, which will be discussed later, perform plan interpretation in an effort to understand why the utterance was stated, rather than the meaning of the utterance itself. This would yield interpretations, such as desires or reinforcement that were given above. However, this added information is trapped within the dialogue manager and its use of a specific API, rather than provided within a message-based medium. The introduction of meaning action concepts to include this added information is essential to the engineering based approach to formal modeling techniques.

Synopsis - Communication Perspective

The goal of this section was to present the background theories of communication necessary for an understanding of this dissertation. The path started with the incredible complexity of natural language, and provided a series of abstractions in order to bring

language closer to rationality. Concepts were presented that will later tie communicative aspects to behavioral counterparts.

The section began by presenting the idea of the speech-act behind an utterance. The speech-act turned the utterance into an action, carrying with it an illocutionary force. The illocutionary force implies various ways in which an utterance could be used or interpreted. Dialogue tag sets then explored how sequences of utterances might be interrelated, as well as a brief look into some dialogue structures including the structure of task-oriented dialogue. Finally, the idea of meaning action concepts was presented as a means for stating more expressively not only the illocutionary force of, but also the meaning inside an utterance.

Behavior: The Theory of Agency

Agency has been extremely successful in a number of areas and is one of the most widely popularized concepts of artificial intelligence. Because of both its simplicity and widespread use, the agent can be used as a universal paradigm, able to incorporate ideas from many interesting and unique research areas.

This section attempts to provide a necessary background in the various aspects of agency as it relates to the dissertation. The section will begin with several definitions of what an agent is, and then provide some of the important types and properties of an agent in relation to this research. Then, related agent standards and well-known architectures will be discussed.

The Agent Paradigm

An agent is a term used to conceptualize the boundary of a particular entity, often referring to a software or control system. One of the more popular definitions [47] defines an agent as anything that *perceives* an *environment* through *sensors*, and *acts* on that environment through *actuators*. This definition, although somewhat wide in scope,

is shared by all agents. All agents should have notions of percepts and actions. The agent paradigm is illustrated in figure 1.

The *internal perspective* of an agent defines how its precepts are mapped to its actions, and what internal components are used in this connection. The *external perspective* defines the various outward properties of an agent as demonstrated by their sensors and actuators.

From a software engineering standpoint, an agent can be considered an encapsulation mechanism much like object oriented programming. The agent encapsulates all of its state information, internal mechanisms for learning and reasoning, and threads of control. The cohesion between the agent and its environment are reduced to that of sensors and effectors, and the agent then demonstrates its behavior within the environment.

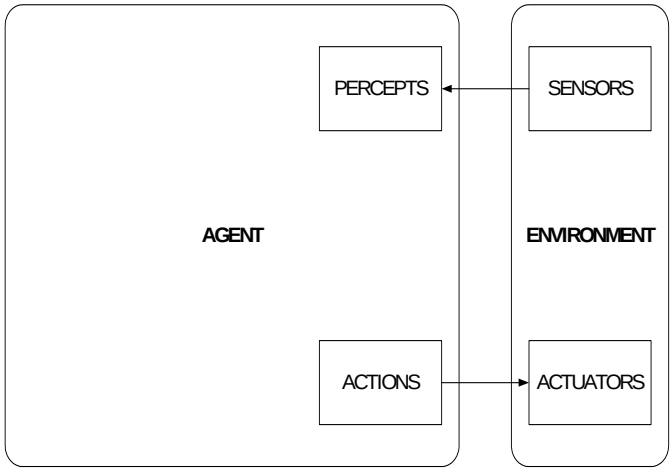

Figure 1: The Agent Paradigm

Agent Embodiment

The agent paradigm as illustrated in figure 1 shows an agent that is separated from its environment. However, most often but not always, the agent will be a part of its environment, and the environment will react to it, or interact with it accordingly.

In order for an agent to exist, it must be *situated* in some form of environment, whether physically or through indirect encapsulation. The exact placement of its sensors and actuators define the boundaries of the agent within its environment. Often the term *embodied agent* will refer to an agent that completely fills a particular system or device, most often physical. This means that the bounds of the agent are the same bounds as the entity that it embodies.

For example, if the agent is to be embodied inside a humanoid robot, then the eyes, location and acoustic sensors would feed into the agent's percepts, and the actuators would feed directly to the motors, speaker and manipulation devices of the robot. Since the robot is situated within the robot's environment, then the agent would then also be situated within the robot's environment. The importance of an embodied agent is that when embodied in an entity, the agent can psychologically believe that it is the entity, rather than the software inside of it. Likewise, the human consciousness can believe that it is the entire human body, rather than something that may or may not exist within the brain. This belief is not only held by the agent, but how the environment perceives the agent, as the environment will often perceive it by the entity that it embodies. In the above example, the environment will perceive the agent as a humanoid robot. Unfortunately, the term, embodied agent has been somewhat restricted to refer only to agents that are placed into a personified entity.

Types and classifications of Agents

Agents span out into thousands of classifications, based particularly on either what environment they are situated within, what properties or capabilities they include, what purpose they serve or how their internal structure maps percepts to actions.

An agent may fit within multiple classifications or types. For example, a thermostat device would fit into the reactive agent classification, in which the percepts, temperature gauge, are directly mapped to actions, heat or cool. It would also fit a mechanical agent classification, in which its mappings are entirely mechanical. Several agent classifications relative to the research topic are defined below along with their association to the agent developed throughout this dissertation.

Intelligent Agent

An *intelligent agent* is an agent that demonstrates intelligent behavior. The definition is somewhat vague and circular, as it is incredibly difficult to define. Often intelligent agents will have a history with which it can analyze all past percepts and actions it has experienced. In addition, it will most likely have a complex behavior structure allowing it to reason and have notions of knowledge.

Sometimes, intelligent behavior can be derived by programming a very simple agent with specific rules that make it appear intelligent within a specific environment; however, this definition often proves a trap when the environment is slightly modified, or a conflict in the provided rules break down the intelligent facade. Therefore, this dissertation will focus on endowing an agent with specific powerful constructs to build its intelligence rather than designing a simple intelligent appearance.

Conversational Agent

A *conversational agent* is an agent endowed with the necessary language capabilities to allow it to hold a conversation with a conversational partner. Conversational agents range from the simple chat-bot systems such as [61] to more

sophisticated question answering agents for information retrieval or agent-assisted user planning [8].

The agent implementation built in this dissertation is capable of carrying out sophisticated conversations with humans over complex command and control and the management of resources in real-time situations.

Interface Agent

The *interface agent* is situated between an interface and another system, which may be a human, an agent or something else. Such an agent will translate the information or requests from one entity to another, in an effort to make the communication between entities more successful.

The agent implementation built in this dissertation is situated between the Stratagus gaming interface and the player. Rather than the player directly controlling the resources in the game, the interface agent controls them on behalf of the player. This drastically changes the interface paradigm the player is using to play the game.

Assistive Agent

The *assistive agent* is one that is designed to assist its user with performing actions. The goal of the assistive agent is to make the user's actions more productive by understanding the user's desires and performing as much work as possible to alleviate the task load of the user. This sometimes elevates the user to the role of manager or director, often just telling the assistive agent goals or desires which are then carried out.

The agent implementation built in this dissertation is an assistive agent in that it performs much of the low-level tasks in Stratagus allowing the user to focus more on the high-level tasks of the game. The user directs the agent by specifying high-level goals and sometimes provides low-level details in how its objectives should be carried out. When there is any kind of conflict in the low-level details that the agent cannot solve or is not allowed to solve autonomously, then the user will be addressed at a higher level to

help resolve the conflict. In the sense of Stratagus, the agent micro-manages the resources and objects within the game while the player orchestrates top-level strategies and objectives.

Social Agents

There has been much research into making agents more realistic and life-like, in order to help their acceptance into society. They have been often endowed with scripted personalities and given expressive qualities, which develop their personification. Many assistive agents have been given these features, such as [44], in order to facilitate their integration into the workplace and their acceptance. One particular researcher did a study on programming computer systems to show signs of distress and lack of confidence noting that their users often felt compassionate and rated the program with better scores than in normal operation.

The dissertation focuses on social constraints and obligations to help define the behavior an agent should or should not take when interacting with the user. For example, if the user asks a question, the agent is obligated to provide a response. The response may not be an answer, but should address the fact that the question was asked. These simple social conventions help conversations to go more smoothly and thus are critical to the adoption of communicative agents.

Agent Standards

As with many scientific and engineering communities, the agent community provides many standards and standards organizations. Standards are an important concept in developing engineering methods as well as the adoption of those methods into practice. Of the many standards of artificial intelligence and intelligent agents, a few have been selected and are described below.

FIPA: Foundations for Intelligent Physical Agents

FIPA [27] is a non-profit organization whose purpose is to develop and produce sets of standards for the interoperation of software agents. Their specifications fall roughly into five categories. 1) Applications demonstrate the integration of agents and their standards. 2) Abstract Architecture discusses the architecture of agents. 3) Agent Communication groups the interaction protocols, communicative acts and content languages. 4) Agent Management includes agent discovery systems. 5) Agent Message Transport discusses the Agent Communication Language (ACL) [28] standard among others.

These groups of standards provide all the necessary building blocks to construct fully complaint intelligent agents as well as their inner components, frameworks and toolsets. Agent communication and agent communication language will be discussed in more detail in the next section, which focuses on agents and communication.

OAA: Open Agent Architecture

The open agent architecture [45] is focused on creating a system where agents, when conforming to OOA standards, can use the interagent communication language to register services it can provide, or find services of other agents. Users of CORBA may be familiar with the basic concepts. OAA is similar to many of the web service description and discovery services and can be used with the semantic web.

Although an important standard, it does not directly contribute to the dissertation research, other than providing an idea on which other communication services may be built. The dissertation focuses on human-agent communication; however, the theory of practical communication of the next chapter is also suitable to agent-agent communication and thus the OAA standards may be applicable.

DARPA Communicator

The DARPA Communicator, now under the galaxy communicator [29], was a project designed to integrate many speech and language components into an integrated system. This included such components as speech interpretation and generation, utterance parsing, and dialogue management. The goal of the project was to allow for the creation of spoken language interfaces utilizing various components of the system. It is now used as a possible architecture for the construction of dialogue systems and is found to be a rival of OAA in that purpose.

This dissertation describes an agent implementation in appendix B. This agent utilizes the galaxy communicator to separate the components of the system. This allows various compatible speech-processing components to be used and their interaction can be logged for both debugging and gathering empirical data.

Agent Architectures

There is a vast array of agent architecture styles. These range from simple reactive systems to systems that incorporate all kinds of design patterns. However, the belief desire intention (BDI) architecture stands above the rest, due to its uniqueness, understandability and widespread adoption.

Belief Desire Intention

The *belief desire intention* (BDI) architecture [30] consists of a model of three specific components, which interact with each other. This model is founded in cognitive and psychological theory, and is one of the major leading models of an intelligent agent.

Beliefs represent knowledge the agent holds to be true, about both itself, and the environment. They are developed from the agent's percepts and previous experience, as well as domain principles provided to it. Although some claim that beliefs not only hold facts but also hold rules, beliefs in the BDI architecture are generally based on prepositions. Beliefs are located within the agent's knowledge. If they are present, then

the agent holds them to be true. If the negations of them are present, then the agent holds them to be false. If they are not present in either form, then the agent simply does not know. Reasoning may update the agent's knowledge to add, negate or remove various propositions of belief.

Desires represent the agent's goals and objectives that are modifiable and selectable. They are developed through reasoning over beliefs. The agent will reason about its beliefs and create a number of possible courses of action based on its overall purpose. In task-oriented BDI architecture, desires reflect these courses of action.

Intentions represent the current desires that the agent is attempting to accomplish. While desire is often high-level, the intention is often lower, such as an instantiated plan that ends in the desire being achieved or maintained. Intentions reflect which desire or set of desires is currently being pursued. Generally, the agent will deliberate on all possible desires and select a single one to pursue, and then generate the plan to fulfill that desire.

If an agent constantly switches intentions based on which desire it is pursuing, then it is viewed to be flakey or scattered. If the agent does not adopt a new desire except when the intention it is pursuing is found incapable, then the agent is viewed to be super-focused or stubborn.

The overall BDI cycle is as follows. An agent performs actions that change the environment. Changes in the environment are reflected in the agent's percepts, which cause changes in its beliefs. These changes in beliefs then may cause changes in the desires of the agent, which may or may not change its intentions, such as making it unachievable or not worth achieving. When the agent no longer wants to pursue its intended plan of action, or comes with a better plan of action based on desires, it will generate new intentions. The intentions will directly lead to the execution of actions and thus the cycle will continue.

The BDI architecture is extremely popular with the notion of rational agency, or agents that are extremely specified by semantics and logic. Many agent-agent dialogue models have been built using rational agency and thus incorporate ideas of BDI within their conversational models. Therefore, an understanding of BDI will enable the understanding of many of these models.

Adjustable Autonomy

One important aspect of an intelligent agent is its autonomy. An agent is autonomous if it can act on its own under its own experience. An important concept for assistive agents is the concept of adjustable autonomy in which the human can control the depth of the autonomy of an agent. A typical agent can act on behalf of a user making decisions as is required. However, there may be a certain decision that the agent has to ask the user to make, or perhaps a constraint the agent asks the user for permission to override.

Many users will want little autonomy in the agent concerning choices with important consequences such as spending money or making commitments; but users will want the agent to have more autonomy when performing less important tasks, thus reducing the amount of bother the user will have to face.

Synopsis - Agency Perspective

The goal of this section was to present the background theory of agency necessary for an understanding of this dissertation. The path started with the agent paradigm, and provided further detail about the models, properties and classifications. Standards were briefly mentioned and the BDI architecture was described.

The BDI architecture represents a starting behavioral notion in the communication-behavior spectrum. The various agent properties outlined, and their discussed relevancy helps to define the goal of this dissertation; not only the engineering

methodology for conversational modeling inside an intelligent agent, but also the essence of what the resultant agent is to become.

Agents and Communication

This section continues both the notions of agency and communication by providing the background material pertaining to their integration. The section will begin with communication among agents. It will discuss the broad transport level between agents, and then narrow down to the specifics over what they are conversing. The section will then provide a path through human-agent communication. This path will start with extremely rational communication in which the human is severely limited on expression, and move the agent into uncertainty by allowing the human more and more expressive power, and incorporating more and more modeling techniques. This section will finish with a discussion over the current language-based behavioral development techniques.

Communication among Agents

Agent-agent communication has come a long way from proprietary messages or protocols to many open-standards for interoperability. One of the most important developments of agent communication is the breakup of communication into multiple layers.

Agent Communication Languages are high-level languages based on the primitives of speech-acts, and utilize structure to express negotiation, information exchange, collaboration and more, that are required for inter-agent interaction. The primitives of speech-acts reflect the illocutionary theory in which the agent desires to perform an action on or with another agent or group of agents, and thus generates a message.

The term 'agent communication language' (ACL) has become polluted to mean any communication between agents rather than at any specific level. Therefore, this section will introduce two additional terms to make a distinction. *ACL transport layer*

reflects the message exchange between agents, rather than the contents of each message, and *ACL conversational modeling* will reflect the contents of each message and neglect how they are exchanged. The ACL transport layer reflects the ability of humans to form, speak and hear utterances. ACL conversational modeling reflects the ability of humans to interpret utterances and understand one another. The further layering of communication itself will be discussed later when communication is related to protocol theory.

ACL Transport Layers

Languages classified in the ACL transport layer act as a transmission and receiving mechanism or underlying protocol of the exchange of messages between interacting agents. The protocol generally handles transmitting a message to another agent or broadcasting a message to a group of agents. All messages are generally associated with an identifier, and contain headers, or meta-data, that associates the message with an identifier acknowledging a reply, an identifier for future replies, as well as an intended receiver and the originator of the message. Some messages also include not only the language of the message contents, but also include the ontology the message content uses. The two major agent communication languages in the transport layer are FIPA-ACL, or the agent communication language, ACL, of the foundation for intelligent physical agents, FIPA, and KQML, the knowledge query and manipulation language. Both of these also provide accompanying agent-frameworks in which one can create, name and organize multiple intelligent agents, and allow them to discover each other and communicate as a society.

Because the ACL transport layer reflects on the message overhead, it is separated entirely from the behavior of an intelligent agent. Therefore, no more detail will be provided because it does not directly influence the communication-behavior spectrum.

However, the transport layer developed for TCL, as seen in chapter 4, is based in part on these transport layers.

ACL Conversational Modeling

Several researchers were unsatisfied with the level of detail found in agent communication languages, such as FIPA-ACL and KQML, as well as the various speech-act taxonomies and dialogue understanding models available. They were rationalists and needed a more formal specification of what each speech-act meant in terms of agents and their behaviors. This was both to reduce the ambiguity presented in the mere name alone and to help formalize a standard to lead to a more deterministic interoperability. While this work was still an agent communication language, much detail was also given to the entire conversation, not only the messages being passed, but the state of the participants as well. Therefore, this dissertation refers to such languages as ACL conversational modeling. In the case of restricting which messages may follow other messages, the model is referred to as an ACL communication protocol. These conversational models typically deal less with the discovery and transport of messages and more with their meaning as pertaining to agent knowledge and reasoning.

One of the major contributors to the rationalization of speech-acts was lead by Cohen and his colleagues. A widely used formalization of standard speech-acts can be found in [54] and formal semantics of KQML based on joint intention theory can be found in [17]. In order to understand the rationalization and specifications, the next paragraph looks at the specifications in relation to teamwork.

In [18] a communicative act for the attempt to achieve a goal is defined as below.

$$(ATT\ x\ e\ p\ q) =<>= (BEL\ x\ \neg p) \& (GOAL\ x\ (HAPPENS\ x\ e;\ p?)) \& (INT\ x\ e;\ q?)$$

This act *ATT* includes which agent is attempting the act *x*, the act the agent will attempt to perform *e*, the goal the agent hopes to accomplish *p* and the result that the agent has committed to performing *q*. Not only has the act itself been defined, but what it

means semantically has also been strictly defined. According to their theory, if the agent communicates this act, it means that the agent *x* believes *BEL* that the goal *p* has not been accomplished. Furthermore this agent *x* states that it has the goal *GOAL* that it will take a course of action *e* towards the goal *p* and that it has the intention *INT* to perform an action *e* to at least bring about *q*.

In this manner, rational agency can be derived through the composition of elements of the underlying theory, or in this case, goals, beliefs and intentions. This idea of using composition of elements and treating a message as a set of underlying or implied elements is an important idea into the subsequent behavior changes a message can bring as well as insight into why a message may have been transmitted. This is critically important in conversations between or among agents, especially when considering conversational policies and obligations, defining properties like expected behavior. However, because of the flexibility of the natural language, these semantics should not be so concretely defined, although they should indeed be well described.

ACL conversational models are generally well formed in first-order logic and transmitted through KQML or FIPA-ACL to the other agents who can understand the components that make up the message. In general, models of this type are proof-theoretic and are built up through logical reasoning with speech-acts as a design guide.

It is clear that with the level of detail in these types of models, if they can be incorporated into human-agent communication, the types of conversations and the understanding power of both participants will be greatly improved over the current dialogue managers. For example, [53] defines both semantics and conversational policies for the concept of a standing offer and its subsequent acceptance or rejection, including the commitments on behalf of the participants. Because such a model is both inspired by human-human communication and founded in agent-agent communication, they should be applicable to human-agent communication.

ACL Specification and Notational Schemes

A variety of notational schemes has been developed in order to represent and describe various agent communication languages and protocols. This section will briefly review the specifications because they apply directly to engineering methodologies of agents and societies of multiple agents.

The majority of these specifications provide only a notational framework for the description of various agent properties. Protocols are often abstracted into just simple message exchanges. The testing and verification methodology to accompany these tools is somewhat lacking, however the specifications provide integration between communication and the behavior modeling of the agent.

UML

UML, or the unified markup language, attempts to be the graphical design representation for all of software engineering. [39] demonstrates how unmodified UML can be used to represent agent communication languages. Specifically, the author incorporates activity diagrams, macros and swim lanes. It is their contention that by using unmodified UML, the concepts of intelligent agents and communication can be understood by the wide range of computer scientists and software engineers capable of understanding UML. In addition, existing tools that can fully model UML can be utilized without modification. Although noble, the added complexity of the structure that is required may not be worth the trade off of standards. Simple modification can yield a more elegant specification.

AUML

Agent UML [4] is an initiative by the foundation for intelligent physical agents. Its goal is to be the all-encompassing agent-oriented software representational and notational language. AUML uses UML sequence diagrams to express the interaction between agents and agent interaction protocols. Unlike the others listed here, AUML is

extremely complex and defines specific standards for the protocol frame, lifelines, messages, constraints, timing constraints, splitting and merging paths, protocol interactions, interaction termination, protocol combination, actions and protocol templates. Although young, AUML has the potential to become a practical agent standard, especially when backed by FIPA.

Agent Communication Specifications in Practice

There gives a plethora of specifications and methodologies for developing agents and societies of multiple agents. In incorporating agent-interaction protocols, Tropos [43] uses UML while PASSI [14] and Prometheus [46] use AUML. Traditionally those that use UML leverage user interaction diagrams while those that use AUML prefer using sequence diagrams. Others such as GAIA [63] abstract protocols down to purpose, initiator, responder, inputs, outputs and processing; and do not specify the underlying protocol itself. Other systems simply abstract the interaction into the act of exchanging a message and do not specify it further.

BRIC [26] is the block-like representation of interactive components. It consists of a high-level language for a modular approach to multi-agent systems. Components are based on UML style syntax and the interaction and communication between components are modeled as Petri Nets.

Discourse Conventions

A *conversation* is nothing more than a sequence of exchanged messages among interacting participants. However, in order for conversations to be coherent and meaningful, agents should follow commonly known rules limiting the types of utterances that can be asserted at any point in the conversation. Following such rules is often referred to as adherence to *discourse conventions*.

A simple example of a discourse convention is answering a question. If one participant asks a question to a second participant, the second participant is socially

obligated to address the question, even if it refuses or is unable to answer. To account for this phenomenon, the notion of conversational policies and conversational obligations has been introduced to the agent.

Conversational Policies

A *conversational policy* represents a communication or dialogue convention that restricts what messages or performatives can be sent and in which order they must follow. Conversational policies can also specify notions of turn taking and abandonment, such as when a message response took too long or there are changes in the environment. The majority of research on conversational policies has been on communication among agents, especially with heavy rational semantics to define speech-acts and model types such as negotiation.

In [10] finite state models are used to construct a set of conversation schemas, a form of conversational policy. They specify the form of a conversation, but leave open the content. The use of finite state models will be later addressed during the discussion of protocol specification techniques to model conversation.

Conversational policy validation must prove the correctness of a given policy. In particular, it should be able to prove that all transitions in the policy are possible, and that no necessary transitions are missing, and the final states are actually endpoints of a conversation. In addition, a conversational policy is often applied to a particular purpose, such as setting up a greeting, or negotiating an item. In order to prove that a policy is suitable for a particular purpose, the expected behavior of the conversation's participants must be specified. These expected behaviors may be things like 'commitment to a goal', or 'giving up on a point' or 'believing what was said to be true'.

Joint intention theory is used in [53] to specify communication acts semantics. The authors analyze two conversational policies and show that these semantics provide meaning behind the policies, analyze the policies for consistency and combine the

conversational policies into more complex dialogues. Their methods for performing verification is merely just showing how the specifications change throughout the policy and reflecting conceptually on what happens. There are no real verification techniques, just merely human inspection.

While agents can follow communication protocols effectively, a human participant may not know of such protocols, nor would they be inclined to learn them in order to communicate effectively with agents. The conversational policies provide severe restrictions in agent systems and control what can be said, and when it can be said. In application to actual human conversation, this restriction must be dropped on the part of the human participant, but not on the part of the agent. This adaptation would allow the human to break discourse conventions when necessary but provide the agent with coherence. Rather than abandoning many of these conversational policies, they can be relaxed into either conversational obligations or layered and used with the focus stack.

Conversational Obligations

Obligations represent what a participant should do or should not do, according to the norms of the medium in which they are based. Social obligations [57] are derived from rules of social convention, often using Deontic logic. Similarly, *conversational obligations* are a form of social obligations that specify what a participant should or should not communicate during a conversation.

Most dialogue systems attempt to connect dialogue acts directly to an agent's beliefs, desires or intentions. However, [57] connects them to obligations which then fit alongside beliefs, desires and intentions within an agent framework. They discuss how this is a better alternative than relying on goal-adoption or intention recognition, especially when dealing with non-cooperative agents.

Dialogue Systems

A *dialogue system* attempts to use dialogue models in implementations to control the interface to a program or set of services. Because of the simplicity of dialogue models, most dialogue systems in use today take on a single role pertaining to the application domain. Applications that help you book a flight, or movie tickets follow an *information seeking* role. *Command and control* pertains to giving orders to a robot or agent. Other roles may involve *believable agents* also known as chat-bots, or even *tutorial systems*, which are designed toward teaching skills to a user.

Dialogue Task Complexity

Dialogue models do need not to have conversational capabilities, nor support natural language; however, they do share two important properties. The first property defines how the state of the dialogue model transitions; and the second property defines the possible contents of the message, or message vocabulary. This section presents a brief history [1] of the types of dialogue systems along with a discussion of these two properties. Given both the history and the context of this dissertation, the systems will be placed within a task-oriented context.

The first and simplest dialogue systems are based on finite-state scripts, allowing the user to input certain symbols in specific states. An example of this type of system could be a touch-tone menu one used to encounter on phone calls with automated systems. Typically, each state represented a question and the user's input would represent their response. This type of dialogue system uses strong conversational protocols, clearly defining how states transition, along with a set of known, finite message symbols. There is no uncertainty in this type of system and the entire model, along with all possible sequences of interaction can be clearly defined.

The second type of dialogue system, a frame-based system, represents early information seeking dialogues. The user typically asks questions, to which the system

responds. The flexibility of the questions and the possibility of clarification dialogues are up to the dialogue model. The more flexible models will walk the user through filling in information for each field in a given frame and provide the query result once the frame is complete. An example of this type of system would be getting airplane arrival, departure and gate information. In the walking fields approach the system would ask first for the airline, and then the flight number. Models that are even more flexible will allow this information to be filled in out of order. The vocabulary is generally known and finite, in this case the system would only need to know particular airlines and flight numbers, as well as how to recognize them. In strict conversational policies, each field of the frame must be filled in when it is asked. In flexible conversational policies, the fields may be provided in any order. Even though the transitions may be flexible, the vocabulary is well defined, and thus there is little uncertainty in this type of system.

An extension to the frame-based dialogue system allows multiple frames to be accessed in a set of contexts. An example of such a dialogue system is a travel agent, where the first context is in finding a flight, and the last context is in finding a hotel or car. The arrival, departure and location information are carried from previous frames. Even the first context can include multiple frames, such as finding departure and arrival cities and then finding seat preferences. Similar to single frame dialogue models, the vocabulary for each context is typically finite and known. The transitions between the sets of contexts are generally also finite and well defined. Therefore, there is also little uncertainty in this type of system.

Dialogue models are often extended through the attachment of a plan library; such models are referred to as plan-based models. Typically, these plan libraries are collections of pre-programmed domain-specific procedures. This type of model represents an elegant marriage between the state transition capabilities of finite-state models with the information aggregation capabilities of frame-based models. A given step in the plan may request information from the user, or formulate responses from the

system. The exact path the procedure will take is based in part on the knowledge of the environment, and in part on the interaction with the user. This approach is expressive enough to emulate all of the previous types. Most current commercial dialogue systems are based on the plan-based approach. Although it is reasonably certain how a given dialogue may play out, it is difficult to prove the non-existence of incorrect behavior. In addition, the vocabulary can be somewhat unknown.

Leading edge dialogue systems utilize an intelligent agent in place of a plan library. This allows the dialogue system all of the expressiveness and flexibility of a plan-based model along with the additional abilities of reasoning, adapting and learning. The vocabulary in this type of model is considered mostly unknown and the transitions between modes of operation and contexts are hard to predict.

The earlier types of dialogue systems represented strict conversational policies and thus forced the user to a limited vocabulary and specific transitions. This greatly reduced the expressive power of user interaction and made the system feasible by shifting the interaction from uncertainty to certainty. However, in the latter types of dialogue systems, the expressive power of the underlying model has been improved; and as a result, the system is brought from certainty into uncertainty to accommodate the naturalness and flexibility of user interaction. This often causes the system to abandon the abilities of explicit deterministic behavior and system verifiability. It is the goal of this dissertation to provide fundamental engineering methodologies to allow for this increased expressive power without sacrificing the integrity or verifiability of the system.

Dialogue Managers

A dialogue manager is the core component of a dialogue system, which incorporates the dialogue model and conversational capabilities. This section will discuss the composition and operation of the manager and its underlying models. Dialogue managers can reason about a task being discussed, track the context of the conversation,

understand partial information and clarification, can utilize turn-taking strategies for mixed-initiative interaction, and more. These capabilities are based in part on the underlying dialogue model.

In a typical conversational dialogue manager, the user will speak or type text. Speech and language understanding components will then translate this into an utterance. Part of speech tagging and parsing is then used to create a frame. The frame represents a semantic representation of what was said by the user. Most dialogue managers deal only with frames and leave the creation of frames up to other language components. Frames are processed in the dialogue manager by applying them to the underlying dialogue model and generating results. Other components, such as utterance and speech generation then use these resultant frames to generate a response that the user can understand.

The underlying dialogue model utilizes several important components [41]. A *dialogue history* is a record of the dialogue in terms of what was uttered by both participants. The history also might include meta-data on how each utterance was interpreted as well as the previous implied connections and clustering of utterances. This provides a basis for not only anaphoric resolution, but also conceptual coherence. The *context*, often referred to as the focus stack represents important utterances, shared objects and concepts currently under the attention of the conversation. The context is used to derive important structural relationships in the ongoing conversation.

Dialogue managers may also use the following types of knowledge. A *world knowledge model* includes the core commonsense reasoning required for operation. A *domain model* includes specific information about the domain. Generic *models of conversational competence* represent knowledge of 'principles of conversation'. Such principles might include turn taking and discourse obligations. A *user model* represents information about the user relevant to the dialogue. A dialogue manager uses all of these types of knowledge to fully understand and resolve a given frame.

Dialogue managers often utilize knowledge to perform *intention recognition*, or the attempt to understand why a user uttered a particular phrase. In addition, they utilize the intentions of the system to figure out what to say next, when to say it and how to say it, also referred to as *content planning*.

Dialogue Control

The control mechanism behind dialogue management can take a variety of approaches. The simplest approach is to base the control directly on the system's beliefs and intention states. This causes the system to treat interaction as reactionary and does not allow for comprehensive conversational competence.

Another type of approach is the *theorem proving approach*, in which the dialogue control attempts to acquire axioms that are missing but required to complete a given step in the theorem. Only the user has the required knowledge to build these axioms, and therefore interaction must occur. Any added information that the user provides will directly be placed into the reasoning of the system. This type of dialogue control is driven by a set of logics and a desire to obtain a result.

Plan based approach

The dialogue control mechanism of many popular systems use a plan-based approach, in which utterances themselves are treated analogously to actions in a planning system. Just as a planner selects specific actions to carry out a goal, utterances are selected in order to achieve a goal. The benefit of understanding this relationship leads to *intention recognition*. In this approach, intention recognition is performed by tracking plan paths to determine which plan or end goal the user is trying to accomplish. Two popular models of planning used in many dialogue systems are that of *joint intentions* and that of *shared plans*.

The problem with systems that rely on plans and intention recognition is that they often impose cooperation, especially in which the agent must understand the plans of the

participant in order to adopt their goals. This makes it impossible to reason when an agent does not need to cooperate or understand the participant, especially where cooperation may conflict with the agents internal reasoning or personal goals.

Rational Interaction

Another method of dialogue control views communication as *intelligent behavior* also referred to as *rational interaction*. This is based on the premise that an intelligent system is required for intelligent dialogue. The theoretical framework behind rational agency is built on the work of [16], later extended by [49]. They introduced a set of logical axioms that formalize simple principles of rational action and cooperative communication.

In [48], the user's utterance can result in a chain of reasoning. For example, the utterance "what is X?" is interpreted into an intention or desire for the user to know X. The system then adopts the intention that the user will know X. In order for the user to know it, the system adopts the intention of informing the user of X. The chain may follow into certain reasoning stages such as, if the user needs to know more or less than X, or if user should not know X due to security restrictions. The chain of reasoning is not predefined, but instead is rationally deduced from principles of communication.

Example Dialogue Systems

In order to understand the approaches and methods of dialogue modeling and control, as well as their contexts of use, two dialogue systems will now be introduced.

TRAINS

TRAINS [57], is a spoken language dialogue system capable of mixed-initiative, cooperative planning in the domain of scheduling trains and shipping. The goal of the system has been to build a spoken language system with specific capabilities, and the underlying theory has been added and refined to make this possible.

The TRAINS architecture augments the plan-based model by using discourse obligations to account for discourse behavior. The *context* component uses a discourse obligation stack where obligations are derived directly from communicative acts. Each incoming conversational act is mapped directly to a discourse obligation, which represents the obligation type and content. That obligation is put onto a stack. In practice, a participant must respond to the most recently imposed obligation, therefore a stack is appropriate for this model. The control mechanism is programmed to respond to any pending obligations before considering other parts of the context. The control mechanism removes an obligation from the stack and translates it into an intention to communicate. If the agent is able to communicate, it produces an outgoing communicative act. If the agent decides to abandon the intention, the source obligation that created it will be returned to the stack.

This method of translating communicative acts into discourse obligations, adopting intentions and responding with communicative acts allows the system to reject proposals and refuse to answer questions, but it still does not have higher conversational abilities such as persuasion or negotiation, nor does it have basic ad hoc competencies.

Collagen

Another successful dialogue system is Collagen [52], which is a truncation of the words ‘collaborative agent’. Collagen is used as an agent that can observe a user interacting with a shared interface and offer all forms of assistance including training the user on how to use the interface, correcting problems with the way the user is using the interface, and suggesting what to do next. Collagen uses an abstract, hierarchical representation of the environment to build a task-model. This allows it quite a degree of application or interface independence.

Collagen relies heavily on a plan tree that contains various procedures on how to use the interface task model. This plan tree is tracked by the actions of the user or agent.

The plan tree gives Collagen the necessary knowledge to train and suggest the next steps in a given procedure. Collagen also maintains a simple focus stack that tracks the current focus of attention.

Collagen uses a universal discourse language [51] to specify the interpretable form of incoming statements. However, most implementations of collagen use a choice-selection template-filling approach that allows the user to select predefined messages to communicate to the system. This greatly dampens the user's expressiveness and restricts the interaction to merely a handful of possible messages.

Conversational Bots

The conversational bot or *chat bot* is another type of dialogue system. The purpose of this type of system has nothing to do with control or behavior. Rather it is intended to attempt to hold a convincing conversation with a human user. Most conversational bots are a clear example of a system that does not try to implement intelligent behavior, but merely mimics intelligent behavior.

Alan Turing proposed a simple game to deal with the question of whether or not a machine could think. Although he referred to the game as *The Imitation Game*, it is known today as the *Turing Test*. In this game, there is a person, a machine and an interrogator. The interrogator is separated physically from both the machine and the person, and is only allowed to pose questions to two unknown entities. Upon receiving responses to the question, the interrogator is to identify which entity is the machine and which entity is the person. The premise behind the test is that if the machine were intelligent enough, than the interrogator would not be able to tell the difference.

The *Loebner competition* is a modern day Turing test, holding an annual competition in which judges, or interrogators, communicate with unknown entities across a computer terminal. The computer terminal only allows simple text to be typed back and forth, much like an instant messaging or chat application. Although most judges can

clearly identify the computer from the person, they are asked to distribute a fixed number of points between two entities. Thus, the intelligent appearance of an entity can be assigned a numerical value. The computer with the highest value wins the competition.

Most modern day conversational bots are based on achieving a result similar to the Turing test, where the goal is to try to convince the user that the system is a human. Often they are developed by creating a set of *rewriting rules* or triggered responses. These responses can be atomic, such as "Hello" responds with "Hello"; translation, in which "Hi" is mapped to "Hello" so that the system can respond to the atomic "Hello"; or pattern recursive translating "Do you know what X is?" to "What is X?" to be properly handled by the system. The input is often *normalized* by expanding all contradictions and removing ambiguous punctuation; and splitting, thus allowing for multiple triggers from an utterance. For example, "Hello, my name is X" would split into "Hello" and "My name is X" as separate statements. The artificial intelligence markup language, AIML [59], is an XML based language consisting of such triggered response rules.

Although it is important for a conversational agent to converse naturally to most effectively communicate with a human user, it is the goal of this research to focus on the behavioral components and the language after natural language processing has taken place.

Engineering Methodologies

Several engineering methodologies have been developed for dialogue systems. A series of projects are outlined in [41] that specify best-practice methodologies for the development and evaluation of dialogue systems. The consensus of these methodologies involves the following.

- Selecting tasks to perform when interacting with a human participant.
- Developing specifications for dialogue structure that will support selected tasks.
- Gather vocabularies and language structures used in recognition. (corpus)

- Constructing a system that meets these criteria.

Shortcomings

There are a few noticeable shortcomings of current dialogue systems. First, there is a lack of corrective dialogue models, or models which are capable of reinterpreting past input. For example, if the user states, "That's not what I meant" the system should be able to review past ambiguities and identify a potential misinterpretation, then reinterpret the origin and resolve the context.

There is also a lack of learning in dialogue models. Current dialogue models are not able to pick up idioms or colloquiums, nor adapt user models to specific reference nomenclature. For example, if the user were to use the expression "That's just the tip of the iceberg" and later the system was able to understand that the expression was equivalent to "That's just the beginning", then it should be able to map all future uses of the iceberg expression to its intended meaning. This is critically important because idioms and colloquial expressions include a significant amount of intentions and other implied meaning.

Behavior Development

Thus far, this section has covered the communication among agents as well as human-agent communication; including how the intelligent agent interprets, reasons about and responds to interaction with a conversational participant. Dialogue systems capable of information query and cooperative planning have been discussed but not systems responsible for command and control. The remainder of this section is devoted to how communication can be applied to developing and managing the behavior of an agent.

Behavior Implementation

There is a variety of ways to design the behavior aspects of an intelligent agent within a computer system. Some of these have been listed in table 5. Although the list is not exhaustive, it is comprehensive and presents accompanied characteristics for discussion.

Table 5: Types of Behavior Implementations

Type	**Manipulation**	**Modifiable**	**Speed**	**Flexibility**	**Properties**
Hardware	Never	Never	Fastest	None	Constant, Reliable
Hardcode	On Release	Never	Fast	Little	Predictable, Reliable
Modules	On Release	Selection	Fast	Modular	Predictable, Reliable
Scripted	Online	Trained	Slow	Much	Semi-Predictable
Graphical	Online	Intuitive	Slow	Much	Semi-Predictable
Rule Set	Online, Self	Intuitive	Slowest	Much	Semi-Unpredictable
Model	Online, Self	Natural	Varies	Most	Semi-Unpredictable

It is possible to design the behavior of an agent in hardware. This was popular with many of the early intelligent robots, especially at a time when hardware was readily modified. A direct hardware implementation can be extremely fast and reliable, however it is hard to modify. Instead, many intelligent agents are compiled in software as either part of a program or an extensible module. This method also yields predictable and reliable systems that can operate in real time. However, these systems are also not readily modifiable unless they were programmed with various options or parameters. Often, these types of systems only represent the mimicking of intelligent behavior.

In scripted and graphical approaches, the behavior is specified through a series of readily modifiable scripts written in an interpreted language. Although the interpretation mechanism can yield inefficiencies in processing time, the flexibility of this approach

allows even the user to reprogram the behavior at run time. However, special training or knowledge of the scripts and the scripting language may be required. Similarly, in a rule set based approach, the behavior is specified by a collection of rules that govern how the behavior operates. Rules may be added, removed or modified as well as the rule selection and execution algorithm. Again, this approach can yield inefficiencies in processing time, and the user may be required to have special knowledge or training in the rules or the rule engine. In addition, this approach can yield unpredictable results. As with more conventional approaches, interpreted and rule set approaches often only mimic intelligent behavior, with the exception that there can be some reasoning and learning in the rule engine.

The model-based approach is the most powerful and the most expressive approach. Behaviors are represented by a set of abstract concepts, such as objectives, constraints, actions and procedures. An example of a behavior model is the belief-desire-intention model as discussed previously. Behavior models often employ planning and reasoning systems, allowing the model to reason and learn. In addition, behavior models are often based directly on the observed behavior of humans and thus seem more natural and intuitive to the user. Applying natural language to a behavior model and behavioral concepts is more intuitive than translating the language into scripting elements or rules.

Behavior development represents the ability of the operator of the system to readily create, modify, manipulate and manage behaviors. The assistive agent in this dissertation attempts to develop behaviors interactively, in real time; and manage those behaviors to deal with inconsistencies, incompleteness and conflicts. A behavior model approach is required in order to accomplish this level of behavior manipulation.

Interactive Behavior Development

There are several systems capable of developing behavior through interacting in real time. The earliest of these systems were referred to as *command and control*, in

which commands would be given by a human operator. In these early systems, the only feedback presented by the system to the operator was either error messages or pre-programmed informational messages. There was no context, let alone any kind of dialogue. These systems pioneered the translating of spoken and typed sentences into a semantic representation for execution.

Task Representation

In natural language driven systems, the utterances of the operator must be translated into a semantic representation, which the system can understand and execute. There have been a variety of specifications for action, objective and constraint representation in intelligent agent systems, but only a couple have had the capabilities of representing information conveyed in language, the most notable of these systems the parameterized action representation, PAR [6].

Types of Interaction

Previous research [37] presented various types of dialogue used in interactive behavior development. This list is not exhaustive, but encapsulates the current capabilities of many dialogue capable agents.

The types of dialogue are organized into several groups, as illustrated in table 6.

The first group, *command execution*, includes the basic capabilities of providing commands and information to an intelligent agent. This represents primarily one-way communication in which the user speaks and the agent listens. The second group, *command feedback*, allows the intelligent agent to communicate back to the user. Thus, the user and the agent can hold discussions. The types of discussion are limited to clarifying or providing feedback on commands and orders. Both of these groups directly affect the behavior of the agent. The fourth type of group, *system interaction*, pertains mainly to querying information such as the state of the environment or the agent's behavior. This group has little to do with affecting the behavior of the agent.

The third type of group, *knowledge transfer,* allows information to be exchanged, thus allowing for both learning and complex reasoning. The last group, *behavior synthesis,* pertains to learning procedures or communicative aliases. Both of these groups affect the behavior engine of the agent and thus the behavior indirectly.

Table 6: Dialogue Types of Interactive Behavior Development

- Command Execution
  - Simple Instruction
  - Complex Instruction
  - Conditional Instruction
  - Single Event
  - Continuous Event
  - Asserting Constraints (Standing orders)
  - Relaxing Constraints
  - Asserting Objectives
  - Relaxing Objectives
- Command Feedback
  - Clarification
    - Incomplete Task
    - Inconsistent Tasks
    - Incapable Task
- Knowledge Transfer
  - Perception Knowledge
  - Domain Knowledge
  - World Knowledge
  - Consequence Implication
- System Interaction
  - Information Query
- Behavior Synthesis
  - Learning by Description
  - Learning by Inquiry
  - Leaning through Discussion (abstract representations, plans or recipes)

Synopsis – Melding Perspectives

This section provided glimpses of the relationships between communication and behavior inside the intelligent agent by providing background material pertaining to their integration. The qualities of this relationship are important, as the next chapter will describe the theory behind a marriage of these technologies.

The section began with the communication among agents, discussing both the broad transport level between them as well as the specific languages they may use. Various discourse conventions and dialogue systems were discussed. The section finished up with a discussion over the various behavior development strategies and some preliminary work in the area.

Empiricism and Rationalism

Before protocol engineering techniques and methodologies are discussed, it is important to understand more of the philosophical views behind this area. This section will discuss several important perspectives that can be directly related to the study of artificial intelligence as well as the design, verification and testing methodologies of various engineering disciplines. These perspectives will enhance the understanding behind the separation of both fields as well as provide reasoning behind the actions and choices made in this dissertation.

Russell and Norvig [47] describe four core perspectives that segment the study of artificial intelligence. These four segments are created by the combination of two properties. The first property is whether the intelligent of an agent relates to 'rational behavior' or 'human behavior'. The second property is whether the agent 'thinks' or it 'acts'. These four segments are composed of subfields as follows.

Agents that *think rationally* are based on concrete well-founded logics, often employing expert systems and other reasoning engines that precisely calculate their behavior. Agents of this nature often fall into the concept of rational agency. Agents that

act rationally are based on developing engines that mimic intelligent behavior, but need not act as a human would act. For example, these types of agents would probably not emulate emotions. Agents that *act like humans* are also based on developing engines that mimic human behavior much like the chat bots discussed previously. Finally, agents that *think like humans* are often based heavily on cognitive or psychological foundations, attempting to simulate brain functions or mental models. Agents of this nature might employ neural networks to accomplish their behavior.

The distinction between acting and thinking in the views discussed above create a perspective of internal and external observation. The internal view of an agent, how it thinks or operates, will be referred to as a *rational view,* or the view of a *rationalist*. Similarly, the external view of an agent, how it acts and reacts with its environment, will be referred to as an *empirical view*, or the view of an *empiricist*.

The original idea behind these scientific views was inspired by the article introducing the *interactive machine* [60]. The *Turing machine*, named after Alan Turing, is a computational model capable of calculating most of the known algorithms. For intended purposes, the Turing machine is extremely rational, constructed by understanding the core theory of computation. How it operates and its computational power is often emphasized more than how it interacts with its environment. Often, the Turing machine cannot react at all, but has all of the tape, or memory, of its calculation in advance.

The interaction machine describes a machine that interacts with its environment. How the machine is constructed is of no consequence, but rather its capabilities when responding to environmental changes is emphasized. The interaction machine is at least as powerful as a Turing machine by simply encapsulating a Turing machine inside any given interaction machine, however it is impossible to encapsulate a interaction machine inside a Turing machine. The interaction machine can explain many problems such as how an ant can successfully navigate a huge and complex environment such as a beach.

The Rationalist

The rationalist is often on a quest for certainty and completeness. They use tools of thought such as reasoning and deduction and focus on logical and mathematical models.

When trying to prove the correctness of a computer system, a rationalist will use proof-theoretic approaches such as process algebras and formal languages. They will prove that a system fits to its specifications through direct logical reasoning.

When modeling natural language, the rationalist approach would be to create a specific protocol, which would be complete and well-formed, then force a human participant to use this protocol when interacting with an agent; thus bringing the human from uncertainty to certainty by only accepting certain inputs in certain states.

The Empiricist

The empiricist is an observer, collecting information about the world through observation. They use tools such as generalization of partial knowledge and abstraction and focus on models of interaction.

When trying to prove the correctness of a computer system, the empiricist will use model theoretic approaches. In model theoretic approaches, it is often impossible to demonstrate the non-existence of incorrect behavior. Rather, the goal is to demonstrate the existence of correct behavior. This will lead to the inability to prove that a system is correct and instead lead to the assessment of the amount of correctness, or lack of incorrectness, in a system.

When modeling natural language, the empiricist approach would leverage the observations of human-human dialogue to attempt to create specific behaviors and incorporate these behaviors into a protocol. They will allow this protocol to be incomplete, based only on partial observations. This will bring the agent from certainty to uncertainty to avoid inhibiting the human's expressiveness.

Communication and Behavior

No specification can fully encapsulate the complexities of the human language. In fact, no specification can fully encapsulate the complexities of even a human task-oriented language. The specification must be made to change, because of not only the mutating and fluctuating nature of human language, but also it must be made to expand as more and more capabilities of natural language are understood and introduced.

Abstraction is a key tool in simplification. The empiricist uses it in order to focus on subsets of relevant attributes and ignore irrelevant ones. In natural language, in particular, all of the irrelevant attributes such as colloquiums, slang and idiomatic expressions can be ignored and the true meaning of utterances can be processed.

Abstraction produces incompleteness. Incompleteness implies that proving the correctness of a model is impossible. Therefore, proof theoretic approaches would be impossible. However, model theoretic approaches allow the specification of behavior and subsequent tests of that behavior.

Because of the very rational nature of agent design and the inability to specify completely the complexities of the human language, a bridge must be created between the agents understanding and the human's expressive capabilities. The empiricist approach to this gap is the more interesting field of research. The incompleteness in the models can only be approached by model theoretic views, which allow the specification of behavioral properties and subsequent verification of that behavior. The use of abstraction will simplify the model as well as its understanding. In addition, model theoretic approaches provide a better foundation for asynchrony and nondeterminism than do proof theoretic approaches.

Therefore, this dissertation must give up the goal of complete behavior specification, and replace it with partial specification. Because the models are based on the behavior specification of interfaces, views and modes of use, software engineering

and protocol engineering, methodologies can be used to specify the interfaces and provide a beginning for the development of formal techniques.

Protocol Engineering

The recent advances in speech and language processing technology, along with an increase in the demand for more natural human computer interfaces have produced new techniques for the understanding and modeling of human practical language. These modeling techniques are applied to human-agent communication, yielding a variety of ways to communicate with an agent. However, successful integration of these models is difficult. The majority are created based on unique features of the language being modeled or the application on which the model is implemented. This creates significant architectural differences and other incompatibilities, which make the integration of these techniques difficult if not impossible.

In order for an agent to understand and communicate with a human, the human practical language, although complex, must eventually be converted into a form that a computer can manipulate, interpret and understand. Because an agent is based on concrete mathematical and algorithmic principles, the nature of their understanding of a given dialogue will allow that dialogue to be conformed to the majority of the properties found in a computer protocol.

Protocol specification, test and verification techniques have matured over the past quarter century to yield systems capable of modeling a set of protocols, verifying properties about them such as completeness, safety and liveness, as well as generating test suites capable of proving the conformance of a protocol implementation to a corresponding standard.

Informally, a communication protocol can be defined as a set of rules that govern the communication between various components of a system. Although traditional coding practices can be used, and even have been proven successful at designing and

implementing protocols, these practices have also produced a high frequency of undesirable behaviors in most protocols. This argues that formal specification and formal techniques for the design, implementation and maintenance of protocols is desirable. In fact, many of the arguments for the application of formal techniques to software testing can be used as arguments for protocol testing.

The Beginning of Protocol Engineering

Protocol Specification, Testing and Verification, is the mature field of applying engineering principles to the design, implementation and maintenance of protocols.

The first major push for formal protocol specification techniques came with the development of the open systems interconnect model, also popularly referred to as the OSI 7 layer model. Standards organizations such as ISO and CCITT that were developing the OSI model saw the need for formal protocol specification and formal description technique working groups. The purpose of these groups was to research the possibility of using formal specifications for OSI protocols and services [9].

These groups laid the groundwork for the development of conformance testing and property testing. These original working groups were also responsible for the development of the first adaptations to formal specification languages for application to protocols, which led to the languages ESTELLE, LOTOS and SDL. Each language comes from a unique methodology and provides the complete semantics of a valid specification. This initial thrust set the pace and focus of the field for the next twenty years.

Rationalism and Empiricism in Protocol Engineering

Although there are many approaches to the modeling of protocols for the purpose of specification and verification, these methods fall into two main categories, the rational approach and the empirical approach. These two scientific views are described in the previous section. A rational approach can sometimes be transformed into an empirical

approach, as with the case of only verifying a set of cases, rather than every case. However, it is impossible for an empirical approach to transform into a rational approach. The empirical approach is often used when the protocol is too complex for the rational approach to execute in any reasonable time, or when attempting a protocol as indefinable as the human language, the empirical approach will be the only choice.

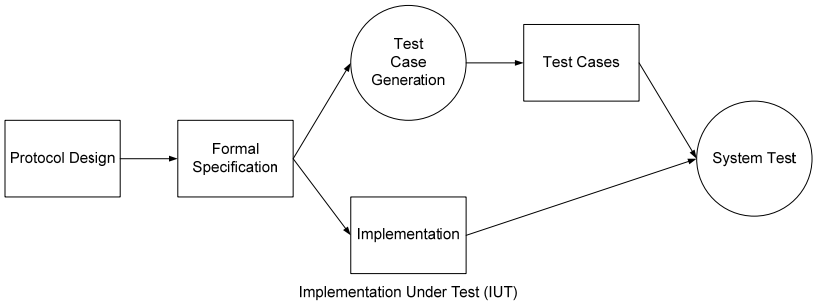

Figure 2: Empirical Approach to Protocol Testing.

The empirical approach to protocol testing is illustrated in figure 2. The protocol is first designed and specified. The specification is broken into two parts, a system which is implemented using the specification, and a suite of test cases generated by a special test-case generating tool. The test cases are then run against the implementation to verify that it follows all of the behavior of the test cases. This test is not complete, because it can only show the presence of correct behavior. However, it cannot prove that incorrect behavior is never present. Only the rational approach, through proofs or complete testing can prove a system does not contain incorrect behavior.

Protocol Testing Suites

Although there are a variety of techniques to test a protocol, some protocols are packaged with specific testing suites that are generated automatically by the specification

as well as any available usage statistics. These testing suites serve two main purposes conformance testing and implementation assessment.

The purpose of a conformance test is to verify that the protocol implementation under test conforms to the provided protocol specification. This ensures that protocols can interoperate with each other, which is also another valid conformance test in its own right. The conformance test is a perfect example of black box testing, where only the outward viewable properties are tested, and none of the inner workings are addressed.

Implementation assessment is a process by which to use available tools, such as test suites and benchmarks, to gather metrics about a given protocol implementation. Such metrics will give an assessment about a given implementation's robustness, or how well the system handles problematic behavior, and performance, or how well the system performs the specified behavior, as well as interoperability, or how many other implementations or systems are compatible with the implementation under test.

Communication and Interaction as a Protocol

In order to be successful in incorporating the complexities of the human language into an intelligent agent, an abstracting view of communication is required. Inspired by the successful model of computer communication, human communication can be modeled as a protocol. An example of protocol layering for human speech communication is provided in figure 3.

The key to incorporating this model with TCL is realizing that at some point, a human is thinking about a task, relating to specific concepts such as an object, an action or an objective. These notions are abstracted as part of the protocol to allow an intelligent agent to share those same abstractions. This allows the agent better understanding of the frame of mind the human is using to communicate.

Similar protocol stacks should be present for other modalities; however, they should be abstracted of all modalities by at least the task model layer. The task model

layer specifies a point at which both humans and agents neglect the underlying communicative structure and abstract directly on the problem (or task) they are interacting over. The task-model abstraction presents a frame-of-mind, which through communication, attempts to be mutually understood.

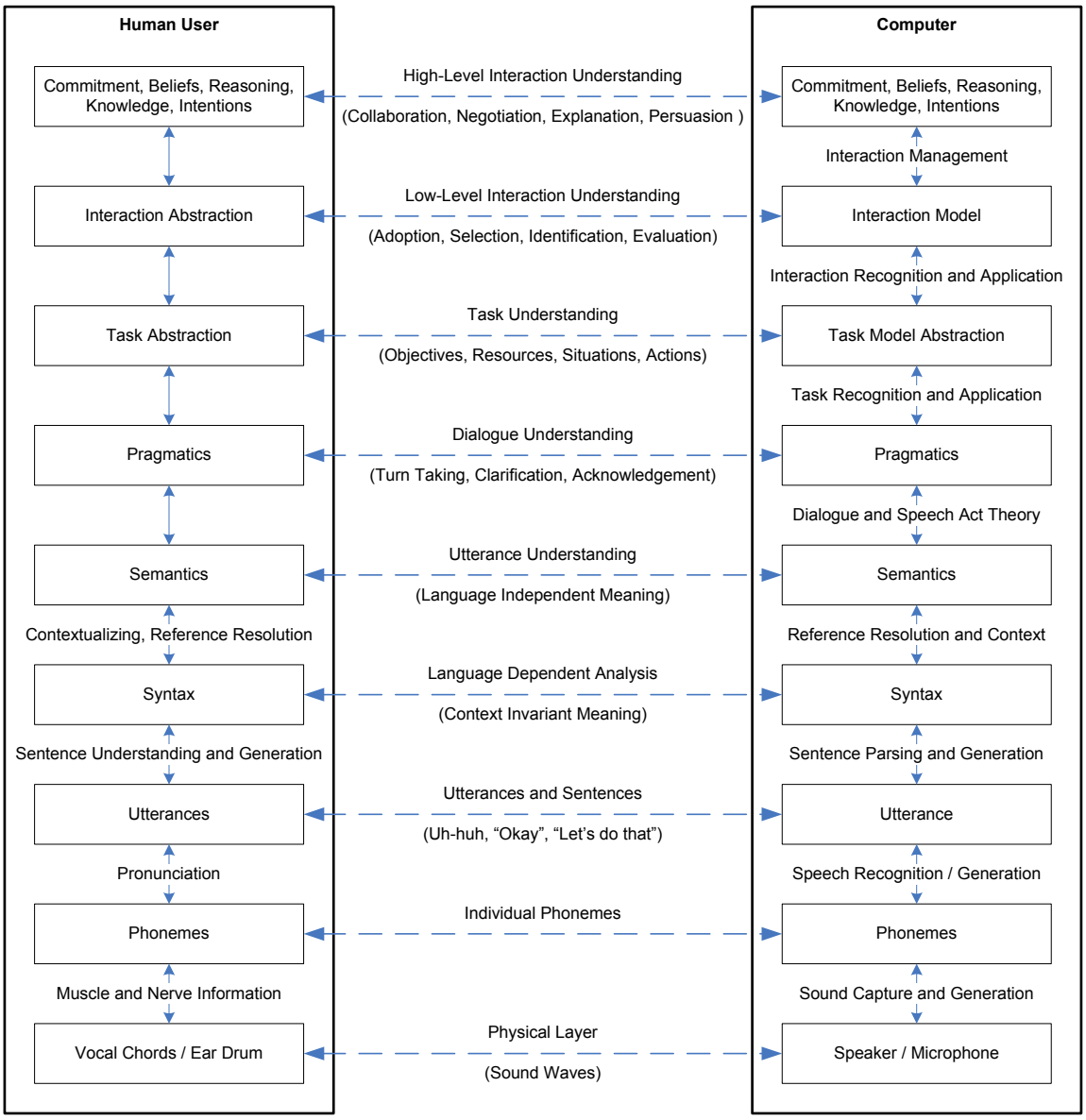

Figure 3: Human Computer Communication as a Layered Protocol

Figure 4: Validation of the Interaction Protocol

The specific formal techniques and methodology developed in this dissertation cover the layers above pragmatics. Mainly, that of high-level interaction understanding such as collaboration, negotiation, explanation and persuasion; low-level interaction

understanding such as adoption, selection, identification and evaluation; and task understanding such as objectives, resources, situations and actions. The layers above and below may be abstracted away as with other layered protocols.

In order to design and implement a system for this layer alone, the lower-layer systems will be replaced with a non-software based alternative as seen in figure 4. Specifically, it will be replaced with a human, which will allow us not only to verify our systems correctness, but also to validate the system with real world experimentation that would otherwise not be possible with today's technology, or lack thereof. This type of experimentation is referred to as wizard-of-oz, as described previously.

This replacement will allow the system to be developed ahead of the development curve, thus it will be ready when the technology catches up, as well as motivate current linguistics to accomplish the required development knowing that it will used as soon as it is available.

Design and Verification Process

Miller [42] discusses an iterative approach to specification, treating verification as a design stage, which provides immediate feedback to the designer. The verifier works on a large number of small problems, rather than a small number of large problems. This helps to avoid exploding the state space and thus the testability of the model.

The model, as seen in figure 5, specifies the system through developing specifications for each of the major components of the system. The components are then verified against the specification of the system. This is the ground case, or base case, and generally based upon the stakeholder needs and system requirements. Once each component has been adequately verified, each component is then broken into smaller components, and the iterative step will verify that these sub-components are specified, verified against the original component specification and iterated upon themselves.

Another iterative verification process is taken in [36], which attempted to assist field experts in recognizing the linguistic instances of a set of concepts in texts. In their system, the text is analyzed and as much information is extracted as possible. Then several instances are highlighted and the expert is able to introduce new knowledge into the system to assist in the analysis. The system is then re-run against the original text iteratively. Each successive addition of knowledge and test against the text produces more and more coverage of the understanding.

A hybrid approach has been developed leveraging the advantages of both of the above iterative design processes and will be discussed in chapter 4.

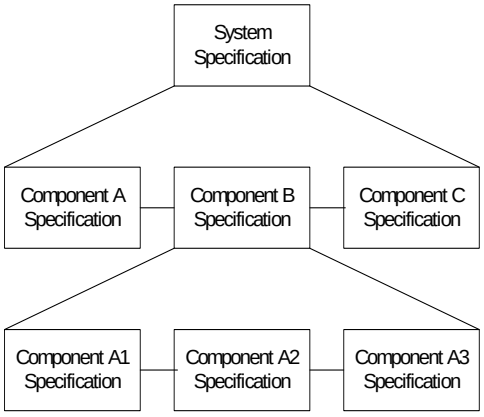

Figure 5: Iterative Approach to Specification

Protocol Engineering Properties

Before the discussion of the various approaches to protocol engineering, two important properties of protocols should be introduced.

The first property, *mutual exclusion*, or the safety property, states that not more than one process can be executing inside a critical section at any given time. Furthermore, if a process is trying to enter its critical section and no other processes are executing their critical sections, than the first process should not be prevented from

entering its critical section. The *liveness property* adds an additional constraint. A process that is attempting to enter its critical section will eventually succeed.

In proving these properties, the *method of projections* technique is often used. In this method, an *image protocol* is created through the aggregation of states, messages and events. This forms a smaller and less complex protocol abstracted from the original, containing only the interested properties. If created successfully, any safety property of the image protocol must hold for the original protocol. An image protocol is said to be *faithful* if any safety or liveness property holds for the image protocol if and only if it holds in the original protocol. Furthermore, if a path in the original protocol can be extended, then the image path can also be extended in the same way as the original protocol. Similarly, if a path in the image protocol can be extended, any path in the original protocol can also be extended in the same way.

The importance of abstracting protocols to image protocols and their relation to various properties of that protocol will be important in the development of the methodology presented in chapter 4.

Approaches to Protocol Modeling

There is a variety of approaches toward the modeling of a protocol for property proving, test generation or validation. Typically, the semantics are based directly on the modeling techniques. However, a few systems take a semantic input representation and translate it another for internal modeling. Only a limited set of modeling methodologies has been chosen for discussion. These have been selected either because they are the most popular used in protocol engineering or have strong implications for modeling conversations.

The Nature of Protocols

Protocols have two strong characteristics that influence their design, specification, modeling and test. First, protocols are very much reactive in nature, seen often as event-

driven and mode-dependent. This leads the behavior-based approach to protocol design. Second, protocols are very much communicative in nature, seen as constantly passing information back and forth, manipulating and modifying data through a series of data-transforming processes. This leads to the communicative-based approach to protocol design.

Quite a number of the approaches listed focus directly on one characteristic of the two listed above, most likely based on the original software engineering model. Then they attempt to extend the model in order to incorporate the other characteristic.

In addition, it is in a protocols nature to be nondeterministic, which makes it extremely hard to test. For example, it is difficult to predict how many times a specific input sequence has to be tested in order to achieve a specific output sequence, or how long the test will have to be in order to view all achievable observations. A protocol may also be partially specified, which also causes problems in the approach.

Finite State Machines

Some of the first approaches were in the extension of the finite state machine, or FSM. This is often attributed to the finite state machine's ability to describe the behavior of a system, and protocols have very strong behavioral characteristics.

Finite state machines also have a set of relations associated with them that can be used as conformance relations. Specifically: the equivalence relation, whether one FSM is equivalent to another; the quasi-equivalence relation, whether two FSMs behavior is the same; and reduction relation, whether one FSM is a reduced form of another.

Fault models are generated for FSMs through the application of a mutant function. The typical mutant function types for the FSM are output faults, where the output of a given transition is wrong; or transfer faults, where the next state of a given transition is wrong. Fault model based testing is an excellent diagnostic tool in that it does not only detect erroneous behavior, but also can provide possible causes, and

locations, for that behavior and can determine possible corrective actions. In natural language, for instance, by applying a misinterpretation based mutant function, it is easy to detect that a misinterpretation has occurred, as well as the point of the conversation that was misinterpreted and how to correct the current understanding path.

Fault models for FSMs however, can lead to what is known as state space explosion. As the size of the finite state machine grows, the number of possible mutation sequences grows exponentially. In order to control this explosion, a series of heuristics are usually been developed. These heuristics are based on information about the implementation, the severity of faults, or hypothesis testing. In addition, a major limitation of fault models is that it does not do well in nondeterministic and partially specified systems and unfortunately the nature of many protocols, as well as natural languages, is to be nondeterministic or partially specified.

There are three major methods used to convert partially specified protocols into fully specified protocols. *Implicitly defined* transition adopts the completeness assumption in which all do not care transitions either are looped or go to an error state. In language, this would be the equivalent of answering, "I don't understand" when any utterance is not understood. *Undefined by default* adopts the notion that all do not care transitions could go to any state with any output. This would be the equivalent of skipping over any utterance that was not understood, and attempting to proceed with the conversation. This is also known as *weak conformance*. *Forbidden transition* treats all do not care transitions as forbidden. This would be the equivalent of immediately halting the conversation if any utterance is not understood. The testing of reaction to unexpected transitions is known as *strong conformance*.

Extending Finite State Machines

Finite state machines alone cannot adequately describe the communicative properties of a protocol and thus several finite state machine extensions were developed.

The leading method, communicative finite state machine, or CFSM, imposes the communication on the underlying behavior structure by associating various transitions within the FSM with transmission or receiving messages. The transitions cannot be traversed unless a certain message is sent or received. The fault model is then extended for the CFSM by specifically looking for a deadlock, unspecified receptions, unreachable transitions and unbounded behavior, such as buffer overflows.

Like the FSM, the CFSM state space usually explodes. CFSM can then be augmented with process variables, which allow the association of predicates and actions with transitions rather than just messages alone. However, it is difficult to adapt fault models to this newly augmented system. Some work has been done in attempting to partition the states and transitions; however, the resulting system is too high level to be of any realistic use.

Petri Nets

Petri Nets are graphical representations that provide well-defined semantics for modeling the behavior structure of a system, much like the FSM. Also like the FSM, Petri Nets are a general description technique that is used across a variety of disciplines [35]. Petri Nets also provide a means of organizing the system into hierarchical descriptions and allow control and synchronization to be integrated with a description of data manipulation [3]. Petri Nets suffer from many of the same problems that FSMs encounter, however Petri Nets are a lot more expressive in their capabilities and thus may hold potential to express naturally the more complicated properties of human-agent communication protocols.

Formal Grammars

Unlike CFSM and Petri Nets, formal grammars focus on the communicative properties of protocols rather than the behavioral. The main approach is to use a formal grammar to describe all of the allowable sequences of a particular protocol. Regular

expression-based grammars translate directly to and from finite state machines and thus hold no added interested for discussion. However, one may be interested in the study of context free grammars and context-sensitive grammars.

Process Algebras

Process Algebras is the algebraic approach to the study of concurrent processes. In this approach, the protocol is specified as a series of equations, and a set of process-based axioms is used to translate and manipulate the form of those equations. Literals of such equations are atomic actions, or steps seen as processes, which are not subject to investigation. Basic process algebra provides three operations: sequential composition ($\bullet$), in which one process is followed by a second; alternative composition (+), in which one process or another process occurs; and parenthesis, which can be used for order of operations. The five core axioms of most process algebra systems are listed below.

$$X + Y = Y + X$$

$$(X + Y) + Z = X + (Y + Z)$$

$$X + X = X$$

$$(X + Y) \bullet Z = (X \bullet Z) + (Y \bullet Z)$$

$$(X \bullet Y) \bullet Z = X \bullet (Y \bullet Z)$$

Process algebras have traditionally been a proof theoretic approach attempting to demonstrate that an implementation and its specification are identical through a series of axiom-based transformations. However, the manipulation techniques have also been used to transform the system into a representation ideal for the generation of tests.

Abstract Data Types / Nondeterministic Data Types

Nondeterministic data types attempts to model the protocol based on data and value passing. Unlike CFSM, Petri Nets and process algebras, this model is extended by superimposing action and control flow on an underlying data-centric structure.

This model is mathematically constructed by associating a special mapping with each data type. Some researchers maps each operation name to a binary relation between the domain and the codomain while other researchers maps each operation name to a set of functional relations between the domain and codomain. Although there are differences in the nomenclature, these two mappings are found to be equivalent.

High-Level Programming Languages

High-level programming languages have often been used to specify protocols. UML based schemes may also fit within this category. Traditionally, software engineering based techniques are used for the testing and verification of these systems, generating test cases from the available use-cases, or the collection of specifications pertaining to how the system is to behave. However, without well-defined system specifications, testing is impossible.

Theorem Proving for Agent Communication Protocols

There has been some work on theorem proving of the various high-level agent communication languages [11]. Specifically, FIPA maintains a collection of communication protocols in AUML. Because several of the communication protocols have ambiguities, inconsistent states and the possibility of deadlocks, a four-stage technique for checking a multi-agent system communication protocol has been developed. The technique is like any other protocol engineering sequence: build a model, implement it in a language for a model checker, create some property, run the model-checker to verify that property. Because of the generality of these techniques as well as their application only to high-level transport layer protocols, they will not be discussed further.

Protocol Variations

Several variations within the nature of protocols apply to conversational modeling. Nondeterministic protocols allow protocols to be specified in such a way to allow multiple possible paths of interaction. These protocols are studied heavily within distributed systems and several adaptations of many formal methods have been developed. Fault-tolerant protocols attempt to recover gracefully from disallowed exchanges within a protocol. The scalable processor-independent design for extended reliability, or SPIDER, system uses fault-tolerant protocol modeling. Probabilistic protocols allow for nondeterminism but with the ability to assign probabilities to each possible path of interaction. The probabilistic symbolic model checker, or PRISM, operates on probabilistic models relevant to protocol engineering.

Communication and Protocols

Several properties of human communication influence the design potential of protocols as well as point out the various limitations of protocols that must be overcome in order to form a successful marriage.

Most importantly, a protocol alone has trouble with global coherence and thus it may not be able to model a conversation well. In the section on agents and communication, the various aspects of a dialogue manager were discussed. Although the world knowledge model, the domain model and the user model can be abstracted in the short run, and the intention recognition and content planning can be isolated entirely; the dialogue history and the context are critical in maintaining this global coherence. In addition, the various models of conversational capabilities must be directly reflected within the protocol itself.

The human practical language is too complex and changes too quickly to be fully specified; therefore, the protocol must be capable of partial specification. Incremental specification would allow the changing of the protocol specifications during the course of

a conversation. Furthermore, the integration of partial specifications into a single specification model would allow the various behavior specifications to be incorporated while alleviating a designer from the integration details.

Certain responses or reactions are unknown within a conversation and there are many possible responses for a given utterance; therefore, the protocol must be able to support nondeterminism. Furthermore, certain responses or groups of responses can be expected within a conversation; therefore, the nondeterminism should support probabilistic properties. This would also yield the capability of evaluating a metric for the correctness of a given interpretation.

The interpretation mechanism will be overwhelmed when attempting to understand the utterances from a human. Even humans can misinterpret one another in conversation. Therefore, the protocol should support fault-tolerance. This should be achieved by allowing the protocol to hold inherent properties such as clarification dialogues and the statement of misinterpretation. Furthermore, a self-correcting protocol, would allow the backtracking and reinterpretation of conversational history to correct and guide the ongoing conversation.

There may be more than one topic of conversation at any given time, and these topics may be discussed through the interleaving of utterances. Therefore, it is essential that the protocol allow for multiple threads of interactions, not only to model multiple threads of conversation, but also to handle delayed responses, branching and converging. Along the same lines, a conversational topic can often be interrupted and later resumed. Therefore, the protocol should also handle interrupts and resuming.

Many natural dialogues adapt to the situation, experience and capabilities of both speakers. For example, one would talk differently to a preschooler than one would speak to a college student. Therefore, the protocol should support adaptive properties. Being able to change based on the conversational needs of the participants, possibly through

user modeling. Other adaptability would allow the protocol to learn throughout the conversation, picking up new nomenclature, procedures or conversational capabilities.

Lastly, in interacting with an assistant agent, a context-driven protocol would benefit by allowing the conversation to be task-oriented, goal-oriented or knowledge seeking in addition to being able to take on such modes as persuasion or negotiation.

As one can see by examining the above list, protocol engineering falls short of being able to model the complexities of the human language. However, protocol engineering provides a foundation of formal techniques and methodologies and it is believed that the majority of these limitations can be overcome. These limitations are addressed by the interaction model validation in chapter 3.

Revisiting the Spectrum

This chapter has provided all of the background necessary in understanding the concepts of this dissertation. The communication-spectrum was laid out, from the models of how humans communicate to the models of agent behavior and the relationship between them. The idea of abstraction, the dichotomy of certainty (rationalism) to uncertainty (empiricism), and the vast complexities of the human language and its impact on communication protocol modeling have all been plotted along this spectrum.

CHAPTER 3
PRACTICAL COMMUNICATION LANGUAGE

The previous chapter introduced a philosophical spectrum spreading from communication to behavior and relating that spectrum to the intelligent agent. It is the goal of this chapter to continue along this spectrum by developing the necessary theoretical concepts to bring the spectrum to realization.

The Practical Language

The first steps toward a foundation for the communication between a human and an intelligent agent was the introduction of two important hypotheses in [1]. The first, the *practical dialogue hypothesis*, is stated below.

> The conversational competence required for practical dialogues, while still complex, is significantly simpler to achieve than general human conversational competence.

The importance of the practical dialogue hypothesis is to focus on those parts of the language that can be developed with the knowledge and technology of today, abstracting the complexities of the human language into what is achievable. The second hypothesis, the *domain-independence hypothesis*, is stated below.

> Within the genre of practical dialogue, the bulk of the complexity in the language interpretation and dialogue management is independent of the task being performed.

The importance of the domain-independence hypothesis is to focus on building a dialogue manager that is abstracted from all domains, and which can be used and reused for a variety of applications.

Although both of these hypotheses allow for the development of generic dialogue systems, they do not provide a common foundation upon which dialogue systems and models can be unified. It is essential that this common foundation conceptualize notions of communication and interaction. This leads to the formation of a third hypothesis, the *practical communication language hypothesis*.

> There exists a language between that of a human conversational participant and that of an intelligent agent. This language is capable of abstracting away the complexity of human language while yet maintaining the practical information of the conversation.

The practical communication language, or PCL, hypothesis is built on the idea that human-agent communication and interaction can be modeled as a protocol. There is cognitive and psychological justification for viewing the interaction of humans and devices as a hierarchy of protocols [23]. In addition, spoken language interpretation is performed as a layer of protocols, as illustrated in the left side of figure 3 in chapter 2.

The true practical communication language is ideal and volatile. This is due to the definition of 'practical' and its ability to continually evolve and expand. For example, the PCLs of the past may have been first order logic semantics for command and control, but recent developments in modeling have greatly expanded the vocabulary of speech-acts. This new vocabulary allows aspects like prosody for detecting notions such as sarcasm, levels of commitment or knowledge certainty.

The idea behind PCL is to carry aspects of meaning and attempt to handle the majority of recognition and tagging at lower levels. The following beliefs are held true in the pursuit of the ideal practical communication language.

- PCL should be abstracted of all region and dialect aspects of a language.

The goal of the practical communication language is to be a unifying language, to all humans and all agents. For practical purposes, an agent should not have to have mastery of multiple languages, and every human language should be able to communicate with an agent.

- PCL should be abstracted of all informal, colloquial, slang and idiomatic expressions.

It is impractical for every agent to know all of the nuances of a particular language. Rather, this information should be abstracted before the practical communication language, so that the agent does not have to deal with this knowledge.

- PCL should be abstracted of all modality.

PCL should be abstracted of modality, including input modalities such as spoken, written or gestural; as well as more subtle modality such as prosody, body position and rhythm. It is impractical for all intelligent agents to know and understand how to react to all of these types of modality; however, this information should not simply be thrown away. For example, [15] demonstrates that modality can influence the establishment of common ground. On the contrary, this type of information should be encoded into PCL in such a way that the meaning of a particularly modality is conveyed rather than the actual semantics of that modality.

- PCL should avoid indirect intention recognition.

Intention recognition pertaining to the meaning of the language should be performed for PCL; however, the actual intended impact of the meaning should not. The domain context and agent rationality is required to perform this recognition and these parts of the agent should be separate from the language used.

For example, the utterance “John is in the basement” should state the given fact rather than remind us that John is in the basement, or to try to get us to go into the basement to see John, or ever more so, to hint that we should avoid the basement all together because John is down there. This should not be handled in the conversational manager, but rather in the agent. The agent should have the ability to use this information to change the state of the conversation based on its own interpretation, not based on direct feedback from some translation function.

- PCL should avoid defining exact terms.

Although the core vocabulary should be well defined, it is impossible to represent the exact meaning of an utterance; especially, when in natural language, the term can vary greatly by speaker and situation and it is not practical to encapsulate. Even though identifying semantics for primitive messages as well as sequences of messages will provide a clear and unambiguous message exchange, a human should not have to know

or follow any of the rules of PCL in order to communicate effectively. PCL attempts to encapsulate the information being communicated and reacting to that information in a rational way. A human participant should not be rationalized to mean explicit dialogue moves, but rather the utterance can mean so many different things, varying by context, speaker, situation, mood and so forth.

Origins of PCL

The idea behind a language that exists in between human language and agent behavior is not new. There are many other examples of such a language. For example: application programmer interfaces, or API, such as the task management interface of TRAINS; specialized languages such as the artificial discourse language of Collagen; a universal communication language such as Interlingua; language interpreted into a machine readable form such as the parameterized action representation, PAR; discourse and speech act tags; agent communication languages or even natural language itself.

Translating natural language into a middle representation before it is processed by a software system has been the overwhelming approach to natural language understanding, whether that representation is an API, a language, a formal logic or something else entirely. The novelty that the practical communication language approach adds is in both its adoption of speech and dialogue act theories, as well as its close relationship to agent communication languages. In addition, PCL based models provide the computational mechanisms for modeling dialogues and behavioral aspects of communication as well as the ability to model advanced conversational capabilities such as negotiation or coordination. Furthermore, the wide foundation of PCL allows many disparate conversational capabilities to all be modeled by the same system, which is the thesis of this dissertation.

Hints of PCL

Of the examples of middle representations above, there are several of worthy note. [40] attempts to demonstrate the use of natural language as an agent communication language. In their experiments, one agent generates a natural language output for another agent to parse and interpret accordingly. However, in most cases these natural language messages were merely predefined statements that were extremely clear on their meaning.

[24] introduces the universal communication language based on the universal network language, which is used to allow communication among people of different languages, or Interlingua. Interlingua comes with a library of universal words, which are translatable into every language, as well as relation and attribute labels. Although the work demonstrates the ability of an agent to understand universal concepts and relations, it is unclear how this leads to change in the agent's behavior or the modeling of conversational capabilities.

In their work on the generic dialogue shell and the TRAINS system, [7] refers to the communication between the dialogue manager and the domain agent as the *interaction act*. Sidner [52] describes the use of an *artificial discourse language* for collaborative negotiation [51] along with an utterance interpretation module and an utterance generation module. However, as opposed to [7], which places the interaction acts after the dialogue manager, Sidner places the utterance intention language before the discourse manager. This leads to the belief that it may be possible to build a practical communication language on either side of the dialogue or discourse manager; or as will be demonstrated in the next section to build the dialogue manager inside the practical communication language itself.

[57] discusses how discourse models based on *joint intentions* or *shared plans*, such as Collagen, are not enough to account for dialogue coherence in cases where agents

do not support mutual high-level goals. The example they provide, “Do you have the time?” does not fit into ether joint-intention or shared-plan.

Both joint-intentions and shared plans assume that the agents are mutually cooperative and come together in a conversation whose purpose is to achieve a task or goal. This is not adequate to model situations where they are not mutually cooperative, such as when one agent is trying to hide information from another agent. Thus, these models, no matter how detailed and complete for their applications, cannot be adopted and extended into the future of all types of discourse, because they have been designed fatally from the beginning.

Communication and Behavior

The theory developed through the remainder of this chapter will be imposed directly upon the communication behavior spectrum. The spectrum is illustrated in figure 6, and future additions will be imposed on this diagram as is appropriate to convey how the theoretical pieces are interconnected.

Figure 6: The Communication Behavior Spectrum

In this particular application of the communication behavior spectrum, one can assume that the human conversational participant will be on the communication end of the spectrum, and the intelligent agent will be on the behavior end of the spectrum.

Although, this is not necessarily true in all circumstances, it is assumed for purposes of this dissertation.

Finding the Glue

In order to create a connection between communication and behavior, a shared representation or medium is required. However, before the discussion of the shared medium itself, it is important to gain an understanding of the exact placement and bounds of the representation. These bounds, along with the shared medium, are illustrated in figure 7.

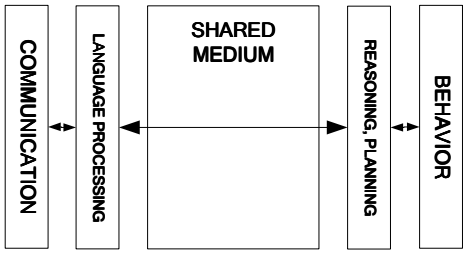

Figure 7: The Shared Medium

On the human, or communication, end of the spectrum, it is assumed that all of the necessary language processing has been performed before the medium is reached. This includes potential speech recognition, part-of-speech tagging, parsing and word-recognition. For purposes of protocol modeling, this would include all layers underneath the task abstraction layer in the left side of figure 3 in chapter 2.

Similarly, on the agent, or behavior, end of the spectrum, it is assumed that the necessary reasoning and planning are performed above the level of the medium, and that the medium only need to interface to the agent through concepts related to knowledge, reasoning and planning. For purposes of protocol modeling, it can be assumed that all of the layers below the task model abstraction layer in the right side of figure 2 in chapter 2,

have been performed by the entity interacting through the shared medium, or in this case, the language processing side.

In this way, the lower end of the shared medium is abstracted to a task abstraction layer, and the shared medium includes the abstract task model, the interaction model and concepts such as commitment, belief, reasoning, knowledge and intentions. What is done with these concepts is performed at a higher level inside the intelligent agent.

A Message-Based Medium

There are two traditional paradigms to consider when developing an interactive medium, one based on messages, or packets of discrete information; and one based on streaming, or continual, information. There is psychological justification [23] that this particular level of interaction may be viewed discretely. Furthermore, dialogue managers, agent communication languages and current agent technology is based on discrete information processing. Therefore, a more traditional message based approach will be more beneficial to current research.

A *message* is a discrete collection of knowledge and information being exchanged or processed. In the practical communication language, a message is most intimately tied to an utterance on the communication side, or a behavioral action on the agent side. How the message transforms between an utterance and a set of behavioral concepts is the work of the next and subsequent sections. The remainder of this section will discuss the message itself, first by covering the meta-information or the external perspective of a particular message, then the message contents, or the internal perspective that acts atomically as the shared medium.

Message Header

The informational data describing the external perspective of a message, referred to as the message header, is outlined in figure 8. The message header includes information that can be used to describe either an utterance or a behavioral concept.

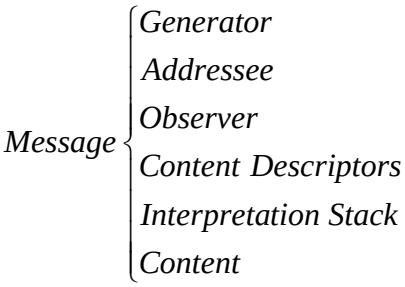

Figure 8: Message Header

The generator of the message represents the participant, human or agent, which generated the utterance or concept. The addressee represents the participants that the message was intended for, while the observer represents all of the participants that were able to receive it. Expressiveness is introduced by allowing the values of these fields to contain a first-order logic expression that not only includes the various participant identifiers, but also various extended concepts from confidence ratings to uncertainty factors. For instance, it may not be obvious who generated a message, or if a particular agent was able to receive it.

Content descriptors allow further knowledge to be conveyed about the message itself. For example, if an originating utterance was in the English language, it would be noted in the content descriptors, and the intended receivers would contain expressions that the participant may need to be aware of the English language in order to have understood or received the message. Further descriptors can account for the level of language and vocabulary, e.g. fifth-grade level as opposed to fluent, as well as a variety of modalities, e.g. if the message was seen or heard and the agent can see or hear. This may even include such detail as which words or expressions the participant may or may not have known.

Although the content descriptors allow for an incredible amount of detail and expansion, they will not be used for the purposes of this dissertation. Furthermore, for

the purposes of the Stratagus agent, the generator is either the player or the agent, and the receiver and observer fields are the other participant, the agent or the player. In addition, it is assumed that messages are received and in a language that is understood.

The interpretation stack holds all of the information obtained at all levels of translating the original perception, such as sound, text or movement data. If created on the behavior end of the spectrum, then the interpretation mechanism holds the rules, intentions or other concepts that were used to develop the concept. The interpretation stack can be used within the system along with a feedback mechanism, for improving the accuracy of the interpretation mechanisms once the actual meaning has been confirmed, as well as to allow various messages to be reinterpreted if there is a misinterpretation detected.

For the purposes of this dissertation, the interpretations of various utterances have been directly input into the system and the interpretation data has not been collected. Furthermore, the feedback mechanism and its development are natural language processing and thus beyond the scope of the research herein.

Finally, the content portion of the message represents the knowledge being exchanged and processed. The content will be described in more detail shortly.

Conversational Paradigms

The descriptive portion in a message of the practical communication language allows various social aspects of the conversation to be defined for later expansion of conversational paradigms. The Stratagus agent described in this dissertation falls only into the single human to single agent paradigm, and the manager / assistant relationship. However, future work may introduce the participant-type paradigms of table 7 and the sub-paradigms of table 8 and more.

From the perspective of the intelligent agent, the human-human paradigm applies to the observation of humans interacting with one another. Through observation, the

agent can learn tasks and procedures as well as conversational obligations and protocols. The agent-agent paradigm pertains to the communication among agents, including agent communication languages and such research areas as the semantic web.

Table 7: Human-Agent Conversational Paradigms

	Human Participant	Agent Participant
Human Participant	Human-Human Communication	Human-Agent Communication
Agent Participant		Agent-Agent Communication

Table 8: Cardinal-Variant Human-Agent Single-Conversation Paradigms

	Single Human	Multiple Human
Single Agent	Human-Agent Communication	Humans-Agent Communication
Multiple Agent	Human-Agents Communication	Humans-Agents Communication

In human-agent interaction alone, many factors further segment various sub-paradigms. For example, consider just the cardinality of human or agent participants as illustrated in 8. In the simplest case, a single human and a single agent, further sub-paradigms include the manager/assistant, student/teacher or coach/player relationship, or perhaps a relationship among peers. Systems including a single human and multiple agents are often used for simulation and training, as well as the push for the interconnectivity of various consumer devices. Systems including a single agent and multiple humans often apply to mediators, discussion leaders, team coordinators or even referees. Examples of multiple humans and multiple agents include the semantic web, online marketplaces and teamwork applications.

Conversational paradigms can even extend beyond single connection groups, such as the IDOCS system [62], which is an online collaborative tool in which a human participates through an agent that communicates with other agents, each communicating to their own human. In addition, paradigms can contain multiple layers or segmentations, such as a conversation between two viewers discussing an ongoing conversation among characters in a television show.

Message Contents

Now that the various external perspectives of the message have been examined along with their implications to the future modeling of practical communication languages, it is time to delve into the internal perspective, or contents of the message, that which is the foundation of a shared medium between communication and behavior.

The foundation of the practical communication language is based upon an abstraction layer in which a human can abstract the details of a problem into various core concepts that can then be manipulated and operated upon during communication and interaction with an intelligent agent. The meaning-action concept, described in chapter 2, provides the basic vocabulary blocks of these concepts. Unlike speech acts, which are performative based, meaning-action concepts can refer to concepts in the conversation, relationship or paradigm, often encapsulating concepts in the domain. The vocabulary of meaning-action concepts is broken down into two essential layers.

The first layer is the core concepts of the paradigm. For example, the task-oriented paradigm may include domain-independent concepts such as ‘action’, ‘goal’ or ‘object’. Various core concepts may be well described. For example, an ‘action’ may be concrete as in the case of a specific action performed or ready to be performed, or generic as in the case of the notion of some form of an action. In addition, they may include properties such as duration, cause and affect, methodology and so forth.

Core concepts create a needed separation between the domain-independent and domain-dependent aspects of a message. In the Stratagus domain, an 'action' may be further defined as gathering resources, building facilities, training personnel and attacking enemy targets. This separation allows the domain-independent aspects to be modeled, reasoned and processed, transforming the message between communication and behavior.

The second layer adds operators to the core concepts. These operators may be tied to communication, relating directly to speech-acts, or they may be tied to the behavior or execution end of the spectrum. Operators from the communication end include such acts as the proposal or rejection of an action, assertion of a mutual goal or the request for information. Operators from the behavior end include such acts as the performance of an action, the evaluation and adoption of a mutual goal or the seeking of information.

In this way, a human conversational participant can introduce various core concepts through performative-like operators. These operators are transformed to behavior-like operators that the agent can then reason over and execute. The agent can provide resultant behavior-like operators as feedback, which can then be translated back into performative-like operators to be communicated to the human participant. The transformation between the different styles of operators is the topic of subsequent sections.

The practical communication expression, representing the contents of a message, or the guts of the utterance with respect to communication, is contained in a single root operator. However, this root operator may contain any number of branching concepts, which provide the general structure of the message and its contents. This representation allows for conjunction, where a particular utterance can carry multiple meanings. For example, "Alright, what do we need to do?" captures both the essence of agreement and the start of a plan. This representation also allows for disjunction in which a particular meaning is ambiguous. It also allows the expression of a variety of complex utterances.

In the Stratagus domain, this may include utterances such as “upgrade all soldiers” or “For each soldier, if there is any enemy soldier nearby then attack the enemy soldier.”

Vocabulary structure and organization

As opposed to a speech act, a meaning-action concept not only has a general categorization, but also carries a signature, categorizing the contents of the performative. For example, there is a distinction between the proposal of an action, and the proposal of a goal. Furthermore, meaning-action concepts are divided into layers according to the relationship model, and may be nested in definition. For example, in the task domain, one can query the justification for the rejection of an action.

Meaning-action concepts are also defined in an ontological format that allows for rollback to known concepts. For instance, a counter-proposal is a child of proposal. The distinction of concepts is made for intelligent protocol modeling systems as well as for generation mechanisms, such as “instead why don’t we…” or “nah, how about…” As another example, confidence ratings may vary. For example, “I’ll get on that right away!” may correlate to a commitment with a confidence rating of 100% while “Well, I don’t know… I’ll see what I can do” may correlate to a commitment with a confidence rating of 15%. The determination of these confidence ratings is left to speech recognition and user modeling. If a particular agent implementation did not know how to handle confidence ratings, then it may treat both as only a commitment that will commit to what is in context.

In addition, the ontological organization allows for the mappings of meaning-action concepts to a root dialogue tag for the incorporation of dialogue tag-sets and associated benefits into a dialogue manager. These mappings will provide useful when the translation mechanism is layered, as will be discussed in subsequent sections.

Following the nature of the complexity of human language, the vocabulary space of meaning action concepts will explode. Using an ontological hierarchy is essential in

the management of future vocabulary spaces. Furthermore, an ontological structure provides decoupling for the dual evolution of the language interpreter, the agent implementation and the practical communication language processing. This allows the various pieces to be evolved separately as the encapsulation of what is deemed 'practical' continues to expand.

Shared Medium Semantics

This section will provide the semantics for the meaning-action concept based shared medium within the practical communication language as described above.

Design Goals

Deciding upon the mechanisms and formalization of concepts is a non-trivial task. Many knowledge representation formalisms of machine understandable concepts are readily available. However, the design criteria for this formalism must be decided before any educated decisions can be made upon which foundation to build. The design goals are outlined below along with the decisions that have been made based on those goals.

- Due to the complexity of the human language and the rapidly expanding capabilities of intelligent agents, the representation should be considered quite volatile and leave mechanisms for evolution and refinement.

The individual concepts should each be expandable, leveraging one another. Therefore, a class-based approach to modeling various concepts has been adopted, leveraging inheritance and other relationships for the easy expansion of concepts. Placing the concepts themselves into a hierarchy, or taxonomy, and requiring that implementations are able to map back to parent classes through abstraction allows for the two sides of communication to be implemented both parallel and independently. As the language encoding mechanism understands new phrases and meanings, it may expand upon previous concepts. The implementation may then advance to incorporate these new concepts. As the implementation expands and adds concepts, various encoding

mechanisms, such as languages, idiomatic dialects and personalities, can be expanded in time to incorporate the new concepts.

- Dialogue tagging and markup as well as initial investigations into representations have revealed that the state space of concepts and their properties can quickly explode.

Therefore, PCL should have a means of organizing these concepts as well as incremental or grouped expansion upon these concepts. Organizing these concepts into a taxonomy and allowing relationships between the concepts will greatly assist in the organization and containment of the concept space.

- PCL should be designed separate of any agent implementation. Any agent implementation supporting the TCL language should be able to utilize it, regardless of its design.

Because PCL is itself a language, it can be independent of any agent implementation. The knowledge communicated in PCL should not imply forward, backward or heuristic chaining. The knowledge should be easily translatable into logic-based, rule-based or class-based implementations. The agent should be treated from a black-box approach, defining only how the agent is to interact with PCL.

Core Concepts

Core concepts represent the first layer of the shared medium. There are two types of concepts as illustrated in figure 9.

The most important aspect in understanding the core concepts is the distinction between abstract and concrete concepts. Abstract concepts represent typing information, describing an entire class of concepts while concrete concepts describe individual instantiations. For example, the idea of a generic objective is an abstract concept where various specific objectives such as gathering more resources or eliminating enemy targets are concrete concepts.

Each abstract concept is given an identifier, which makes it unique from all other abstract concepts. Similarly, each concrete concept is given an identifier that makes it unique from all other concrete concepts. A concrete concept must be the implementation of a specific abstract concept. The concrete concept is given an abstract-identifier that specifies this relationship.

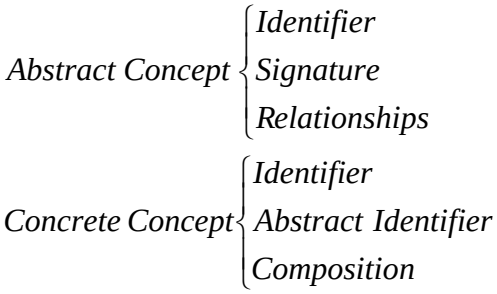

Figure 9: Core Concepts of the Shared Medium

There are important relationships among abstract concepts that allow them to be organized in an ontological format. The *is-extension-of* relationship allows various abstract concepts to be further refined as the model grows in complexity. For example, the 'counter-proposal' *is-extension-of* 'proposal', allows the counter-proposal to be added for systems that can distinguish it from a generic proposal. However, systems that do not know how to handle or process counter-proposals may treat it as a proposal.

Another important relationship of abstract concepts is composition, or the *has-a* relationship. This allows abstract concepts to contain references to other abstract concepts, which may be used during processing within the discourse model. For example, an abstract procedure concept may be composed of actions and may include an objective. Various properties of abstract concepts are also described using this relationship. The *signature* allows the individual *has-a* relationships to be distinct by

giving an identifier to each relationship. In addition, the *has-a* relationships may also be designated to be required or optional as well as denote cardinality possibilities.

The only relationship a concrete concept is allowed is the composition relationship. Composition allows the procedure of 'gathering resources' to be the concrete actions of 'moving to the resource', 'mining the resources' and 'delivering the resource'.

All compositions are associated with an identifier that makes that relationship unique to the other relationships of the same concept. Furthermore, some of the composition relationships in a concrete concept share the same identifier with the composition relationships in their root abstract concept. This correlation preserves the signature that is shared between the composition in both concrete and abstract concepts.

Concept Operators

Concept operators represent the second layer of the shared medium. Similar to core concepts, there are two types of concepts as illustrated in figure 10.

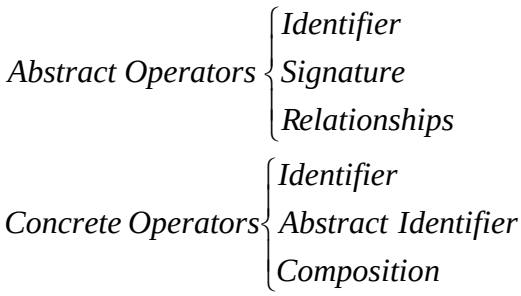

Figure 10: Core Concept Operators of the Shared Medium

The core operators are broken into abstract and concrete operators. Operators can be nested. Similar to core concepts, the abstract operators represent typing information, describing an entire class of operators while concrete operators describe individual

instantiations. In addition, both operator groups are also given identifiers, unique within their respective groups.

As with abstract concepts, the *is-extension-of* relationship can also be found among abstract operators. This relationship allows various abstract operators to be further refined as the ontology grows in complexity. For example, the relationship 'commitment-with-confidence-rating' *is-extension-of* 'commitment', allows a confidence rating to describe the level of commitment a given concept may have. If a system does not know how to deal with this added information, then it may treat the operator as the commitment operator.

The signature of an abstract operator allows the operator to describe to which abstract concepts the operator applies. The signature is composed of a set of unique identifiers, each identifier including a reference to an abstract concept or set of abstract concepts. The identifiers are used to distinguish abstract concepts in the event that there is more than one abstract concept type within the signature. Each signature identifier also includes a property stating if the given identifier is required or optional. In addition, in the case of a set of abstract concepts, the concepts may or may not be ordered. Only abstract concepts may be used in the signature, as the operator itself is only abstract.

A concrete operator must be the implementation of a specific abstract operator. The concrete operator is given an abstract identifier that specifies this relationship. In addition, each concrete operator is given composition, which is a set of concrete concepts. This set includes identifiers, which map up directly to the signature of the corresponding abstract operator.

Dialogue Models

The process of transforming performative-oriented meaning-action concepts to behavior-oriented meaning-action concepts is performed by various rules within the dialogue model. Without a dialogue model, the meaning-action concepts would serve

merely as an application programmer interface. It is the introduction of these meaning-action concepts as well as their respective translation and processing rules that is the novelty of this dissertation in addition to the formal methodology surrounding these concepts and rules.

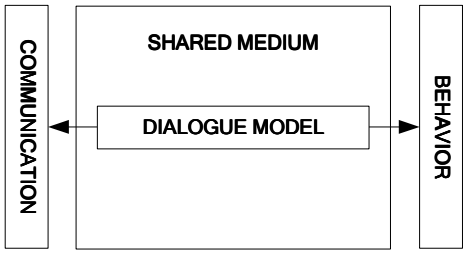

Figure 11: The Dialogue Model

The dialogue model is placed directly inside the shared medium as illustrated in figure 11. This restricts the dialogue model to operate only on the core concepts and their operators and avoids any language processing or behavioral reasoning. The dialogue model is not only capable of interpreting and responding to various speech and behavioral acts, but it is also capable of modeling high-level interaction including negotiation, explanation, mutual planning and more.

This is performed through understanding the relationships between the various concepts and operators and utilizing policies to specify these relationships. For example, upon the proposal of a goal, the agent should evaluate whether or not it should pursue the goal. If the evaluation leads to the adoption of the goal, than the agent conveys that it has accepted the proposal, otherwise it should convey that the proposal was rejected. The agent may provide a justification along with the rejection if pertinent.

In this simple example, a proposal is evaluated and either accepted or rejected, depending on if the goal was adopted. However, the example can quickly get much more

complex when considering the following variations. The proposal itself may demonstrate misinformation on the behalf of the generator that would result in statements that neutralize the proposal. If the proposal was not understood, a sub-clarification dialogue in resolving the misunderstanding should result. Furthermore, if the proposal intended a specific consequence that would be achieved through a better course of action, then a counter-proposal may result. A counter-proposal may also be used for refining the proposal during negotiations, to reach a compromise for instance.

As can be seen, even when dealing with something as simple as a generic proposal, there is a variety of rational ways an agent can react. Therefore, a generic dialogue model is needed which can account for these variations through a simple and deterministic approach.

Message Processing

As illustrated in figure 11, the dialogue model is situated directly in between the communication with a human participant and the behavioral operations of an intelligent agent. The dialogue model transforms performative-oriented messages and behavioral-oriented messages.

Generally, when a human participant produces an utterance, various speech and language tools are used to transform that utterance into a message. From the perspective of the dialogue model, the message is simply produced by that language end of the spectrum. Similarly, when there is a change in the environment or in the internal state of the agent, the agent may produce a message intended for the human participant. From the perspective of the dialogue model, the message is simply produced by the behavioral end of the spectrum.

The goal of the dialogue model is not to simply transfer the message from one end of the spectrum to the other, but reason about and react to the message according to its knowledge of the context, and various discourse conventions.

Discourse Reasoning

The dialogue model developed in this dissertation follows the idea of rational communication by leveraging a reasoning system to process messages. This reasoning system utilizes discourse rules and knowledge to model obligations and policies as well as a structured context system that records and tracks the history of the interaction as well as the focus, intentions, interpretations and rationale of messages in multiple conversational threads.

Discourse Structure

The discourse model is composed of three primary pieces. The first piece is the discourse structure, which is capable of tracking multiple conversations, recording interpretation history, and monitoring obligations and contexts.

Messages can only be generated by a participant in the conversation. The human generates a message through interaction, while the agent typically generates a message through reasoning. Upon the arrival of each message into the system, all of the top-level concepts in the messages signature along with any sub-concepts as provided through composition are all added to a collection of *Shared Concepts*. Each is given an identifier if not already present.

The operators of the message will invoke various rules to be fired calling subsequent operators with their concepts. As each rule is fired and concepts are transformed or created, these are each added to the *Shared Concepts* accordingly. In addition, the history of interaction that leads to each concept is recorded in a *Shared Concept Graph*. The introduction of the shared concept graph is one of the novelties of this dissertation. The shared concept graph holds the key information for reinterpretation, back tracking, correction and many other conversational capabilities as will be described later. In addition, a *Shared Concept List* provides the most recently added or addressed

concepts. This is useful in searching for related concepts that are in the current focus of the conversation.

Operators are also recorded in the Shared Concept Graph both through their ties to the various concepts as well as through discourse obligations. When the agent produces a message that expects a reply, the operator of the message is added to the *Obligations*. The next message from the other participants is checked against the obligations to see if it is a reply. If it is deemed a reply, the obligation is removed.

Discourse Operators

Only operators, as opposed to concepts, may be used as top-level facts within the reasoning engine. There are four types of operators used within the discourse model. More on each rule can be found in their corresponding section in the partial TCL language definition provided in appendix A.

Interactive operators influence concepts toward communication and interaction. These include operators that interpret or generate interaction such as text. Top-level interaction operators are generally shared by more than one participant.

Agent operators represent actions taken internally by the agent. These operators are generated through transformations of interactive operators. The operators lead directly to reasoning within the agent, which then leads to the production of new interactive operators.

Helper operators never reach either participant. They do not interact with any rule, or any outside concept or operator. Furthermore, there is only one definition allowed for each helper operator. In addition, helper operators always return a value. In this way, helper operators can be viewed as non-overloaded inline functions.

Helper operators work directly with the structure of concepts as well as the shared concept graph. They detect structural overlaps such as collisions or containment, and create new concepts through the merging or influence of multiple concepts.

An exception to the traditional functional role of helper operators is that they may also be used as a placeholder to hold information in obligations until a future rule can use this information. In this way, the helper operator is terminated as a fact to be used by future operators.

Macro operators are similar to helper operators in that they never reach either participant, or interact with any outside concept or operator. However, macro operators often are overloaded and produce new interaction operators. Macro operators help to detect various operator signatures and cause a change in resultant operators based on that signature.

Discourse Rules

Discourse rules translate among the various messages, manipulate the shared concept graph, and create and destroy obligations through multiple threads of conversation. Discourse rules generally affect the most recent operator and do not interact with one another directly, but rather from one operator to the next forming a branching chain.

A discourse policy is a collection of rules that interact with one another towards a specific purpose, such as modeling a conversational capability or behavior. These are often organized both in layers and in groups.

Every discourse rule has three main parts as illustrated in figure 12. The *operator* specifies the operator that will trigger the rule; the *conditions* specify what must be true for the rule to fire, and the *results* specify the operators that will be generated if the rule is successful. Each section may carry a complete signature structure of a concrete operator including concepts. Variables may be assigned to individual concepts that are bound at the time of rule firing. If there is a collision with the signature of the operator or any of the conditions, then the rule fails.

$$DisrouseRule \begin{cases} Operator \\ Conditions \\ Results \end{cases}$$

Figure 12: Discourse Rule

Multiple variables may be specified in each operator of the rule, and the variables may be shared among multiple operators. Variables may only be bound to operators or concepts.

```
while execution stack is not empty
    pop operator (Oper) off execution stack
    for each rule (Rule) in dialogue rules ordered by priority
        if bind variables of Rule-operator and Oper are not successful then next rule
        for each operator (Condition) in rule-conditions
            if bind variables of Condition is not successful then next rule
            if execution of operator Condition is not successful then next rule
        for each operator (Result) in rule-results
            bind variables of Result
            push Result on execution stack
        exit for each
```

Figure 13: Dialogue Model Execution

Once an operator is defined, complete with signature, a set of operators representing a condition may be defined. Condition operators may only be obligations, concepts in the shared concept list, helper operators, macro operators or agent operators. The agent operators involved in the condition section do not generate, but rather allow the agent to decide whether specific conditions have been met.

If all of the conditions have been met then the rule executes, or fires. This means that the operator that triggered the rule is removed, and the set of operators in the result section are introduced. These introduced rules will trigger and fire other rules in their own way, leading to a sequence of operators that are chained, branching at any particular rule.

Only one rule may fire for each operator. If there is more than one rule, then the rule with the highest priority which has all of its conditions met, will fire. If no rule fires, that the operator is considered to be causing an 'unspecified transition', which is considered an error. The described algorithm is illustrated in figure 13.

In example, consider the rule from the human trial of TCL as illustrated in figure 14. This rule states that if an order is received that is an action; it should be evaluated with the intent to execute the action. Typically, if an incoming message is the order operator of an action concept, this rule will fire. The action concept will bound to the variable '?A'. This rule will remove the original order message and replace it with a messsage for the agent to evaluate the action concept as provided through '?A'.

$$FollowOrders \begin{cases} Operator = (\text{Order} < \text{orders Action} : ?\text{A} >) \\ Conditions = None \\ Results = (\#\text{EvaluateAction} < \text{action } ?\text{A} > < \text{intent execute} >) \end{cases}$$

Figure 14: Follow Orders Rule

The Interaction Model

The dialogue model introduced in the last section outlined the rational transformation between performative-oriented and behavior-oriented meaning-action concepts. As the dialogue model represented the internal mechanics that orchestrate this process, the interaction model defines the general interaction patterns found within this

orchestration. The introduction of the interaction model is one of the novelties of this dissertation. The interaction model provides the foundation for the formal methodology that allows the verification of the soundness of the dialogue model as well as the validation of the dialogue model against actual conversations as well as their properties.

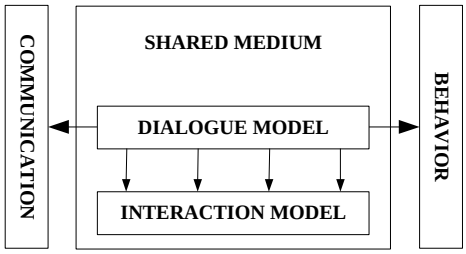

Figure 15: The Interaction Model

The *interaction model*, as illustrated in figure 15, is placed directly inside the shared medium, restricting it to operate only on the core concepts and their operators. The interaction model is separated from the dialogue model as well as the stream of messages between the communication and behavior sides of the spectrum. It used only for the verification and validation of the dialogue model.

The interaction model is automatically generated by tracing all of the possible routes of interaction operators, the performative-style meaning-action concepts, within the dialogue model. This interaction represents the collection of the various possible sequences from the communication side of the spectrum and therefore, it strongly resembles a protocol validation suite. Because the behavior of an agent cannot, and should not be included because the agent is abstracted, the behavior end of the spectrum must also be abstracted. This leads to weak-conformance of the dialogue model. Future work may entail including this agent-verification during the construction of the interaction model to produce stronger conformance tests.

The interaction model itself includes several layers, which range in conformity and properties. These layers are introduced by varying the condition mode during generation. The condition mode determines which conditions will be branched and which conditions will be checked, as will be described. These layers allow the paths within the interaction model to expand and retract, the bounds of which are used during the debugging process of conformance failures.

Interaction Model Generation

```
for each rule (Rule) in dialogue rules order by priority
    Operator = Rule-operator
    for each operator (Condition) in Rule-conditions
        if Condition is type Obligation
            add Condition to Restrictions-Obligations
        if Condition is a Shared Concept
            add Condition to Restrictions-Concepts
    if Operator→* (Link) exists in Interaction-Paths
        add Link to Restriction-Priority
    for each operator (Result) in Rule-results
        add link Operator→Result to Interaction-Paths
```

Figure 16: Generation of Interaction Model

The generic algorithm can be seen above in figure 16. Every possible operator-to-operator transition is recorded within an interaction graph. However, there are a series of checks to label the transitions in specific ways. This labeling creates the layers within the interaction model.

The weakest interaction model allows all conditions to branch in all possibilities, allows all rules to fire without noting priority preferences and does not check either the

shared concepts or obligations for success. This allows every single transition to be added and is the biggest of the interaction models. Models are then restricted by allowing obligations to be tested, tracking shared concepts, or adding priority preferences or a combination there-of. The strongest interaction model considers every possible conditional check.

Interaction Verification

The verification of the dialogue model is performed during the generation of the interaction model. Any paths that are terminated without reaching a message can be revealed which represent under-specification in the dialogue model.

Interaction Validation

The interaction verification process resembles the iterative verification process discussed in chapter 2. This process is followed by applying the interaction model against known annotated corpuses of dialogue sequences. If the dialogue sequence is covered within the interaction model, then that particular sequence is validated. However, if the dialogue sequence is not present, then the interaction model should be examined to see if it could be expanded to include the cases that the dialogue sequence reveals. These additions are made to the dialogue model and a new interaction model is generated and tested against the corpus. This process can be iterated until the variations within that corpus have been accounted for.

If a given dialogue sequence is present within the weakest interaction model but not present in the strongest interaction model, then in practice it has been a problem with priorities, reliance of shared concepts or a missing obligation as indicated by the differences between the two models.

In addition to validation against known dialogue sequences, various image protocols can be created through abstracting the various performative-style messages using the ontological hierarchy of the operators. These image protocols can be used for

validating nondeterminism, probabilistic-determinism, multiple conversational threads, delayed response, interruption and adaptation; as well as various conversational capabilities such as clarification, re-interpretation, negotiation and so forth.

Practical Communication Language Methodology

Following the general process for creating a dialogue model, the conversational capabilities are recorded in a dialogue sequence. This sequence represents the design requirements for the system. Formal specifications are then introduced by adding the necessary rules to model this dialogue sequence.

The core concepts and operators provide an adequate specification language to formalize these dialogue sequences as design requirements. Furthermore, the verification and validation techniques defined in this section provide effective to check the requirements against these definitions. Therefore, the practical communication language techniques outlined in this chapter provide a methodology for designing and verifying dialogue systems.

Synopsis

This chapter continued along the communication-behavior spectrum by developing the necessary theoretical concepts. This includes defining the practical communication language, as well as meaning-action concepts and the shared-medium, message based interaction, the dialogue model and the interaction model. This has provided the foundation for the novel contributions of this dissertation, the practical communication language methodology. The next chapter will continue this foundation through its implementation within the task-oriented domain, and the subsequent chapter through its implementation within Stratagus.

CHAPTER 4
TASK COMMUNICATION LANGUAGE

Due to the complexity of the practical language introduced in the previous chapter, a much simpler problem of task-oriented language has been the focus of implementation. Task-oriented language is defined as language pertaining to the planning, management, monitoring and execution of tasks and procedures. This chapter introduces the design and reasoning behind the task communication language. A partial TCL language definition can be found in appendix A.

Task Concepts and Operators

The theory developed in the last chapter discussed the formation of language through a set of concepts and operators on those concepts. However, before the necessary concepts and operators are introduced, the problem of properly representing and organizing a task-oriented language will be addressed.

Representing Task-Oriented Concepts

TCL has one major goal in that it must be rich enough to store the complexity of action expressible in natural language. Very few action representations take into account the linguistic information of natural language instruction [25]. Many simply map input to recipes in a plan library, rather than allowing this linguistic information to morph the core recipe into something new and interpretable. Many times, the linguistic information points to methodology that suggests the purpose for the action, which demonstrates the intentions that influence the actions and goals of the agent.

There has been extensive work in translating natural language to action representations [6] [37] [25]. However, these representations do not take speech acts and dialogue modeling into account. They are not capable of readily incorporating notions such as negotiation or persuasion. Therefore, a new hybrid language must be

constructed; one that follows the path of the practical communication language but yet is also able to incorporate the expressive concepts found within task-oriented domains.

TCL will map directly to a set of core concepts, which are then expanded through linguistic expression. Any system capable of using these extensions may do so, but those not yet capable will operate directly on the core concepts.

Representation

Core domain-dependent concepts follow the object modeling approach where each concept in the domain is modeled as an object. Each object has a set of associated properties and values, which may be terminal or may be another object. Objects also have a set of relationships, which associate them with other objects in meaningful ways.

For instance, in the Stratagus domain, if we wanted to know how much health a soldier had, we would query for the health property of that particular soldier object. If we wanted to know which units were inside a particular vehicle we would query the inside-of relationship of the corresponding vehicle object. This enables the ability to put together semantics to access almost all important domain specific knowledge.

In an effort to expand the knowledge representation, specialized concepts are introduced for both values and relationships. These consist of AND, OR, NOT, IF, ELSE-IF and ELSE. The AND and OR concepts allow a set of sub-concepts or values to be combined together into a group for easier representation and referencing. The actual meaning of these concepts is dependent on its reference and usage. For example, when an AND concept is referenced as a goal, then all of the branches of the AND concept must be met in order to satisfy the goal. Similarly, if the OR concept is referenced as a procedure, then any of its branches will satisfy the execution of that procedure. The NOT concept is the complement of its reference, and the IF, ELSE-IF and ELSE concepts follow their respective programming ideas.

Message Validation

TCL is checked against a schema to ensure that it follows the concept hierarchy the schema defines. However, the validation stage only goes so far in proving the correctness of the concept. Additional verification steps may be required by an implementation but is not addressed by the TCL framework. Certain human utterances and gestures may also generate incomplete or even conflicting semantics, however because this is possible in the human language, it should not restrict these utterances from TCL. Instead, the agent implementation is forced to deal with such utterances accordingly.

Task Concept Construction

Although a number of TCL concepts and operators are not included in this dissertation due to length considerations, a partial language specification is provided in appendix A. This section will provide a brief introduction into some of the core concepts and types, providing a solid foundation for understanding the TCL framework. Readers are referred to the appendix for a more comprehensive construction of the TCL shared medium.

As previously discussed, the knowledge representation follows a scheme of concepts and operators with associated values, properties and relationships.

Core Types

As with any language, a few core types are required to begin the foundation of TCL. The *Expression* uses the values of concept parameters or other expression concepts through any number of specialized aggregators as defined above. An expression generally only returns a truth-value and does not provide common properties other than structural composition. Expressions may also deal with abstract notions to generate formulas applicable to any concept. Expression concepts are typically used in evaluating IF and ELSE-IF concepts.

The *State* concept refers to a particular concept's parameter, an expression or an aggregation of both. States represent general knowledge provided through the interface of known values, whether internal or in the environment. A state also generally returns only a value and does not provide common properties other than composition. A state is satisfied if it matches a value or a particular set of values.

Entities

Several important entities exist within the task-oriented domain. Typically, there is some notion of an *Object* that refers to an object within the environment, but also as in [37] can also refer to non-existent ideas. The various properties of an object are domain-dependent. In addition, there is notion of an *Actor*. An actor refers to an object within the environment that is capable of performing actions. The actions an actor can perform are domain-dependent. An object may also be a *Resource* that is to be measured and monitored. Sometimes a resource can refer to a particular amount of a given element available while other times it can refer to a specific tool or device. Resources are typically application-specific.

Goals and Objectives

In a task-oriented domain, there is a notion of a goal or objective that must be carried out to complete a task. Multiple research fields present many conflicting insights on how a goal is to be represented. This work attempts to accommodate most of these. The core set of [12] has been adopted and expanded upon.

First is the *Goal* concept itself, which represents something that can be achieved. The goal concept has a great number of properties that all add to its definition. Most important of these properties is the goal's *Type*, which reflects how the goal is to be satisfied. The satisfaction requirement of the goal is typically referred to as the *Result*.

TCL developed in this dissertation accounts for the following goal types: In *Achieve*, the goal is to achieve a particular result. It can be either an action to be carried

out or a state of the world to satisfy. In direct opposition is *Avoid*, which is the goal to avoid a particular state or action. TCL represents constraints in this way. *Avoid* goals are typically continuous within their contexts.

A *Maintain* goal type tries to maintain a given state. Anytime the state is unsatisfied then it is to be achieved, if it is satisfied then the agent is to plan to keep it satisfied. Along the same lines is *Preserve*, which is identical to *Maintain*, except that if the state is ever unsatisfied, then the goal is expired or abandoned.

A *Cease* goal type is to undo a particular result or reach the negation of the result. In the case of a state, the goal is to unsatisfy the state. The *Test* goal is to test a given condition but not necessarily achieve it if it is unsatisfied.

These types make up the various types of objectives that exist within a task-oriented domain. TCL models almost all of these concepts as extensions to the core goal concept. This core concept also has a number of properties as well.

First, each goal has an *Origin*, which states if the goal was provided by the system, the agent, another user or perhaps another goal or some reasoning. Also, the goal has a *Synthesis*, which describes how it came to be, whether stated explicitly or through inference. Both the origin and the synthesis properties of a goal are provided by TCL through the shared concept graph.

A goal may have any number of *Sub-goals*, a collection of goals that must be satisfied in order to satisfy the goal. Each sub-goal's *Parent* is the owner of the collection it is in, if applicable. Furthermore, each goal can have a *Priority*, *Method*, *Scope*, *Applicability*, *Composition*, *State*, *Progress*, *Estimates* and more. However, due to length considerations, is deemed beyond the scope of this dissertation.

Action

The goal and the action are the most important concepts in a task-oriented domain. Moreover, they are also the most complex. The action concept developed in

this dissertation attempts to maintain the natural language concepts of [25] while maintaining the practicality of [6] and [37]. There is one core abstracted-action type, which reflects on all actions. Then there are two important sub-types: the recipe, which represents the un-instantiated action; and the action, which represents an instance of a particular un-instantiated action.

This section will not go into too much detail on the action concept due to length considerations. Rather, the intent of this section is to provide an adequate synopsis of the task-oriented domain for understanding of how the practical communication language is applied.

Most importantly, all actions require *Participants* that define who is performing the action, who is assisting in the action, what objects are being performed on and what objects are being used. Each of these is represented in a respective field. Furthermore, all actions require an applicability that describes to which state the action applies, along with prerequisites and effects that describe what must be true when the action begins and what will be true when the action completes. In addition, there is a duration-state, which describes what must be true throughout the course of the action.

An action may also have a *Preparation* describing actions to be carried out in advance. Most are dampened with an IF concept to test a condition and carry out an action if that condition is not met. If all of the preparations and prerequisites are met, the execution may be carried out.

In addition to these properties above, actions also have a *Parent*, *Manner*, *Termination* conditions, *Results*, *Maintain*, *Duration*, *Purpose*, *Concurrent*, *Priority* and more. Furthermore, other application specific properties may be attached to the action. Instantiated or un-instantiated actions or even plans may all be referenced through the action concept. This is because although their implementations are diverse, the communication over them is homogeneous.

Other Concepts

In addition to the two most important concepts, action and goals, TCL also has notions of *belief*, what a participant holds to be true; *reasoning*, or the evidence why a certain belief is held; *desires*, the goals a participant wants to achieve, *intentions*, the current plans of a participant; along with other notions such as *certainty* or *correctness*.

Task Operators

Typical task based models of multi-agent interaction, such as [12], use a generic set of operators which operate on all shared concepts. However, it is essential to infuse performative-based meaning into these operators to account for the expressiveness of the human language. Therefore, rather than building a question with 'selecting' a query concept and answering that question by 'selecting' another concept, TCL uses operators such as 'query' about a concept, and 'answer' with a concept. Not only does this help to trace the concept through the shared concept graph and aid in the disambiguation of meaning with generic operators, but it also allows various discourse sequence patterns to be detected and traced, such as question-answer. Although this added information in the operator allows the communicative translator to more naturally and more expressively communicate with a human participant, the agent side of the spectrum is abstracted from having to deal with this added information through either the discourse rules or the ontological hierarchy of operators.

Dialogue Modes and Sequences

Following the rule definitions of the practical communication language, various rules utilize intent and obligations in order to chain together operators in a specific way as to create a dialogue sequence. For instance, a question is generally answered. The specific settings and usage of intents and obligations throughout a particular dialogue sequence is referred to as a dialogue mode. For instance, when the mode of the rules is set to 'mutual planning', each participant attempts to refine a shared plan until it is

satisfactory to all participants. The human participants can drive the conversation in any way they see fit, however the agent will attempt to drive the conversation back toward the current mode until completed or abandoned.

Understanding rules and how they change the concepts and operators, as well as having procedures or patterns that modify the context in a predefined way, gives the agent the ability to push the conversation in such a way to achieve a specific goal. Furthermore, the agent may set the direction for series of conversational patterns depending on their implementation.

Layering of Dialogue Modes

The ability to push the conversation in a particular way, as well as utilize a series of dialogue sequence patterns allows a conversational pattern to be layered. For instance, a round of question and answer patterns may be strung together to produce information-seeking behavior.

The lowest level of patterns found within TCL rules allow for the following modes: inquiry, statement or disagreement of belief, deliberation, formal argumentation, information argumentation, clarification, explanation, information absorption and active listening. The higher levels of modes used by the agent within TCL include persuasion of belief, reaching mutual understanding, negotiation, learning by description, command and control, mutual planning and learning through orders.

Protocol Engineering Revisited

This section attempts to return to the protocol engineering topics described in chapter 2 and apply those topics to both the practical language theory and the newly formed task communication language.

Utilizing Petri Nets

The dialogue rules described in chapter 3 represent a transformation of data. They are designed to remove the operator enabling the transition as well as produce new operators as a result. Therefore, this transformation may also be viewed as a state transition between operators. A state machine is inadequate in modeling the various obligations and rules because multiple operators can exist simultaneously in addition to the overwhelming connection to the shared concept, shared concept graph and obligations. However, the similar paradigm of Petri Nets, or more specifically Prioritized Hierarchical Colored Petri Nets, provides an excellent modeling methodology.

TCL is modeled as Petri Nets by placing abstract-operator types in various places, allowing each place to store the complex concrete operator of that abstract type. A place in a Petri Net is similar to a state in a finite state machine. Then, any rule that triggers on that operator type is attached as a transition from that state to all of the states corresponding to the operators that the rule generates. Conditions are added to the transition matching the conditions of the rule.

The shared concepts, the shared concept graph and obligations, are added as specialized places and connected to the transitions of the corresponding rule. If any data is removed during the enabling of any particular transition, that transition is responsible for adding the data back. This handles the case of some Petri Net implantations that require all places to be stripped of enabling conditions. Furthermore, each transition is responsible for adding information to these specialized places upon enabling. The specialized places are referenced by the rules as a place. However, in actual implementation each place is hierarchically defined as many smaller shared places and transitions that emulate the behavior of each structure according to the definitions of the last chapter.

The prioritized feature of Petri Nets allows the rules to be prioritized as described in the last chapter. Therefore, only one transition is made for each place. The

hierarchical nature of Petri Nets allows various layers to be modeled within the Petri Net. The rules are designed in such a way that the mode of dialogue corresponds to various hierarchical containments.

Modeling the task communication language within a Petri Net tool has provided invaluable graphical visualization for understanding and debugging as well as the generation of various interaction models and the subsequent validation of those interaction models as described in the previous chapter.

Natural Language to TCL

As a brief aside to linguists, a full natural language front-end can be developed for TCL. It is recommended to abstract the grammars into separate layers, particularly the task-model layer and the domain-implementation. The task-model layer would pertain all of the linguistic information required for generic actions, orders and objects. The domain-implementation would include the formal wordings of these concepts. For example, objects in the Stratagus domain include 'engineer', 'crystal' or 'training camp'. This allows the task-model layer to parse domain independently, such as, '<action> 3 more <object>', 'first, <action> then <action>' or 'if <expression> then <action>'.

As a reminder, it is not the goal of this dissertation to build a translator for the front-end of TCL.

Synopsis

This chapter dealt with the application of the theory of practical communication languages to the task-oriented domain. It discussed the design and reasoning behind the task communication language. Interested readers are strongly encouraged to see the partial specification in appendix A for a semantic treatise.

CHAPTER 5
EMPIRICAL INVESTIGATIONS

The thesis of this dissertation states that there exists a language between that of human natural language and the behavioral reasoning of an intelligent agent, and that this language is capable of not only unifying the various models used in literature, but also provides the foundation for a theoretical framework for an engineering methodology for building such models. In order to prove this hypothesis, an intelligent agent has been constructed using the implementation details of appendix B. This agent has been placed inside a resource-management simulation. An example session with a human participant can be found in appendix C. This chapter will emphasize a proof-of-concept that demonstrates the viability of the theoretical methods discussed and reinforce the thesis.

It is the goal of this chapter to argue the thesis through demonstrating the core conversational capabilities of the intelligent agent through references to the example session in appendix C. These core capabilities are expanded into demonstrating a variety of current human-agent and agent-agent interaction models such as argumentation and negotiation.

Conversational Capabilities

A *conversational capability* is defined as being capable conversing in a particular facet of conversation. Examples of conversational capability include negotiation, mutual planning, answering questions and so forth.

This section discusses the conversational capabilities that the intelligent agent of this dissertation has been endowed with, and reinforces that capability with examples from the listing of the provided human session in appendix C.

Turn Taking

The term turn taking refers to the fact that multiple participants in a conversation cannot generate messages at the same time, and therefore, must wait while one of the

participants has their turn. Although turn taking is a critical aspect of spoken dialogue systems, it was not a necessary component of the empirical investigations within Stratagus. Therefore, turn taking was not taken into account. Rather, the agent's messages were displayed on the screen for the human participant to read at their leisure.

Human Initiative

The core conversational capabilities are broken into three sections based on the manager and assistant paradigm. *Initiative* in reference to conversation refers to a participant's ability to take control and lead the conversation. The human initiative section includes various message patterns that are generated by the manager, including command and control; as well as the generated by the human, such as dealing with linguistic cues. The section also limits itself to single participant patterns.

Command and Control

Command and control has primarily been the most prominent interaction in task-oriented domain. In fact, some of the first work on natural language recognition has been for the command of robots. Command and control is continually expanded as more and more basic patterns and recognition schemes increase what is deemed practical for interpretation. Earlier work on command and control [37] has shown great promise in the field. This section hopes to build upon that work. This particular subsection of command and control will be limited to the manager only. Command and control is expanded later in the section on multiple participants.

Simple Instruction

An instruction represents the core element of command and control. A simple instruction is taken to be an order given to an assistant to be carried out. Simple instructions are instructions that can be directly executed with no need for resolution. The available human session includes dozens and dozens of examples of simple

instruction from, “mine titanium” to “build a training camp to the northeast.” One need only glance at the provided session to see simple instructions at work.

Complex Instruction

Complex instructions are connected with other instructions to create sequences, precedence and other compositional structure. They also include instructions that carry multiple atomic actions within them in a non-obvious way. In the human session of appendix C, the statements "After that have another engineer build 2 generators" on line 45 and the combination of "First, train six soldiers" on line 231 and "Then, group them together" on line 242 represent complex instructions that are bound to each other or other instructions. The first example is precedence that the previous action in context is to be taken before the action on line 45 is to be taken. The second example combines both line 231 and line 242 into a single complex instruction of an action sequence.

An example of an internalized complex instruction would be "Make two more squads, one at each camp" on lines 270 or 388. Both of these actually represent the conjunction of making a squad at camp1 and making a squad at camp2, as illustrated on line 278 or shared concept number 86.

Through these examples, the TCL implementation has demonstrated the capability of understanding complex actions.

Suggesting Instruction Methodology

In addition to complex instructions is the ability to express a methodology on how they are to be carried out. Examples in the provided session include “use a new one” on line 75. Other methodology such as which tools to use, what constraints to uphold and how to optimize the action, such as quickest or least resources, is also allowed in TCL.

Conditional Instruction

The third type of instruction is a conditional instruction, which ranges from one-time checks to standing orders to constraints. Typically, conditional instructions take one of the following forms. “Whenever X do Y” is an action in which the condition X is checked continuously and whenever true the action Y is performed. “When X do Y” is more ambiguous whether it be continual or once only. However, X is generally found to be an event and Y is the action to do when that event occurs. If X is a state it is usually wrapped inside an event concept. “If X do Y” is dependent on the tense found in X. If X is future tense, then the instruction is deemed a conditional instruction waiting for an event. If X is present or past tense then it is generally considered a one-time check. If the condition X is true than Y will result, otherwise no action is taken. “Do Y until X” performs the action Y until the event or state X is reached. If the phrase “Do Y while X” then the event may be continual, meaning it may re-assert if X becomes true again. One can quickly see the expressiveness found within natural language, even when relating to conditional instructions. These forms can even be extended using ‘Else’ and ‘Otherwise’.

In general, it is up to the agent implementation to understand when an instruction is to be run once, or whether it is to be run every time the context is reached. However, some cues may add helpful information such as “When X, always do Y”. When the keywords, ‘Never’ or ‘Do not’ are used, typically the action falls into a constraint, such as “Do not walk on the grass.”

In the provided session, the statements "Whenever a squad is ready, have it attack the enemy" on line 520 and "Make squads as necessary" on line 549 represent standing orders, a form of continual conditional instruction, thus demonstrating the capability of TCL to model conditional instructions.

Prioritizing Instructions

The ability to prioritize instructions is one of the additions to the task-oriented domain through expressive interaction. The provided session yields two expressions of prioritization, the first of which expresses the necessity to carry out the action quickly on lines 75 and 109, “as soon as possible.” The second expresses precedence, one action is going to be carried out before other actions, on line 99.

Statement of Objectives

The last form of single-participant manager-initiative command and control is the assertion or abandonment of objectives and desires. In the Stratagus domain, these typically fall under such statements as, “We need more crystal” or “I want 2 more engineers.” The provided session reveals "But, I really want missiles" on line 317, whose unmodified meaning on line 318 represents the assertion of a desire.

Statement of Knowledge

An important aspect of single-initiative communication is the ability to state knowledge, whether it is knowledge about the world, or internal knowledge of the participant such as beliefs. For example, in the provided session, line 478 asserts dissatisfaction; line 644 asserts the correctness of the agent; and line 806 asserts a correction of meaning. Even stronger, most answers are viewed as statements of knowledge until they are taken into the question / response layer.

Linguistic Cues

The usage of cues demonstrates advanced conversational capability in understanding human generated statements. How TCL addresses and generates cues are described all through out this dissertation. The provided session reveals the statement combinations on lines 231 and 242; 448, 449 and 450; and 748, 749 and 750, which demonstrate the ‘first, then, finally’ cue combinations. Lines 162 and 438 demonstrate

'then' alone. Lines 317, 567 and 718 demonstrate 'but' cues. Lines 038, 045, 54 and 263 demonstrate 'more, other, another' cues. Lines 75 and 109 demonstrate 'new'. Line 100 demonstrates 'also'. Lines 270 and 388 demonstrate 'each'. Finally, lines 487 and 500 demonstrate 'more'. Additionally, 'our' and 'their' are used internally.

Agent Initiative

Agent initiative includes various message patterns that are generated by the assistant of the system, through the monitoring and execution of actions, events and resources.

Notification and Bother

The notification of conditions or events is a feature that allows the manager to let go of their attention of the system by allowing the assistant to monitor it. The assistant is capable of bringing the managers attention back to the system through notification. Notification is a form of 'Statement of Knowledge' described in the previous subsection. The provided session demonstrates notification through the statements "The training camp has been finished" on line 207, "The second training camp has been finished" on line 221, "We can now make squads twice as fast" on line 531 and "We have finished researching explosives" on line 534.

One of the important aspects of the agent's ability to manipulate the attention of a human is the decision of whether or not and when to perform that manipulation. The agent in this dissertation follows the general guidelines of [19].

Multiple-Participant

Multiple participant interaction pertains to the interaction of two or more participants, particularly when the messages they produce correlate with one another.

Questions and Answers

The earliest and most widely adopted multiple participant interaction type is the ability to ask and answer questions. Typically questions take one of two forms, a polarity based, yes-or-no, question and a content related question, such as who, what, when, where, why and how.

Yes-or-No Questions and Answers

The polarity-based question and answer are the easiest to recognize and connect. This is because the various statements that answer a polarity-based question, such as 'Yes', 'No', 'Maybe', 'I don't know', 'Not sure' and so forth, are all easily recognized as answers. This is mainly because they are not able to stand on their own as a statement. There is no meaning behind telling, asserting or proposing 'Yes'. However, the word 'maybe' may also be a forward-looking statement that correlates to confidence such as, "Maybe we should…" or "Maybe I need…" and so forth.

The provided session is filled with polarity based questions and answers.

Concept Questions and Answers

The content-based question and answer are a little trickier to recognize and connect. This is because the answer is often recognized as an assertion of knowledge. In addition, most questions have an ambiguity behind what they are actually asking for. For example, in the Stratagus domain, the question "How many engineers are there?" may be met by any one of the following responses. "No idea", "Not sure, maybe 14?", "Anywhere between 10 and 14.", "At least 14 that we have seen", "14", "14 engineers, 10 of ours," "We have 10 engineers", "There are 4 enemy engineers", "14, 10 of ours, 4 of theirs that we know of" and so forth. The construction of the knowledge pertained in an answer is the responsibility of the agent, whereas the presentation or translation of that knowledge is the responsibility of the text-generation mechanism.

The provided session is filled with concept questions and answers.

Delayed Response

In some cases, it is not possible to answer a question until later knowledge is acquired or events occur. In this case, the answer is typically an acknowledgement of the question and the assertion of the postponement of the obligation to answer the question. Such a statement would be something like, “I’ll let you know.” This is recorded within the TCL agent as an obligation to be addressed later, and the dialogue model is capable of using this type of response to complete the question-answer matching.

Information Seeking Dialogue

A series of questions pertaining to the same concept or a set of concepts is referred to as an information seeking dialogue. This type of dialogue is constructed by connecting the various question-answers together and recognizing the intention of the questions toward a specific objective. An example of an information seeking dialogue would be along the lines of “How many airline flights go to Detroit.” “When is the first one?” “How long is it?” “How long are the others?” A typical information seeking dialogue will utilize a number of modifiers and references to connect to the previous question or question’s answer.

An example of an information seeking dialogue in the provided session would be the focus of attention on engineers from line 652 to line 830. This particular dialogue is introduced by the statement on line 652, “How many engineers do we have?” This statement creates a set of engineers, which are then selected, queried and iterated. "What are they doing?" on line 662. "Show me the ones that are standing." on line 678. "What is its history?" on line 697, and so forth.

Information seeking dialogue is a domain in itself outside of the task-oriented domain. However, it is demonstrated that a great many properties of the information seeking domain are modelable inside TCL.

Clarification Dialogue

Another important pattern found within task-oriented dialogue is the ability to use a question-answer pattern for such purposes as resolving a misunderstanding, conducting refinement, confirming interpretation and resolving ambiguity. These types of dialogue are referred to as a clarification dialogue where a query is constructed in order to bring an operator and concept into better understanding. The intentions behind a clarification dialogue are called grounding, which is the desire of both participants to confirm that they are discussing the same concept. Examples of clarification dialogue within the provided session are abundant.

Command and Control

Now that the various question-answer patterns have been described, the multiple-participant variations of command and control can follow. Although there are many multiple-participant variations, three core mixed-initiative variations will be described in this section.

Incomplete Instruction Resolution

When an instruction is ordered but there is insufficient information for that instruction to be carried out and that information cannot be easily deduced; a clarification dialogue must be used to fill in the missing information. This pattern is referred to as incomplete instruction resolution. Several instances of instruction resolution can be found within the provided session. The first instance includes the original order "Send 7 engineers to mine" on line 1, which is incomplete because the agent does not know what resource the engineers should mine. Therefore, using clarification it introduces the question “What should they mine?" on line 21. This question is later answered on line 29 with "Crystal" which is resolved as the answer to question 21, and thus completes the clarification dialogue. The answer is then combined with the original order in line 32, which is then sent for execution in line 35.

Another instance of this type of dialogue starts with the order "After that have another engineer build 2 generators" on line 45. This order is incomplete because the agent is not sure where to build the generator, therefore using clarification it introduces the question “Where?" on line 53. The answer to this question is provided on line 54, "To the west, near the other generators”. It is deemed a valid response and resolved as the answer to question 53 and thus completes the clarification dialogue. The answer is then combined with the original order on line 59 and is sent for execution on line 63.

Instruction Problem Resolution

Another type of instruction resolution comes from the fact that the agent is not always able to carry out the order because of some restriction in the environment, it does not have a required permission or it opposes the shared objectives of the human and agent. In this case, the agent must use another type of dialogue, such as a request for permission, a rejection of the action or an address of shared objectives.

In the provided session, the order "Send 7 engineers to mine” is given on line 1. This order cannot be carried out by the agent because there are only two engineers in the game. Therefore, the agent responds with the rejection of the action, “I can't, there are only two engineers” on line 8. This is similar to an error message one might receive when attempting functionality in a computer program. However, the agent is not done. Rather, it produces a proposal of a solution that would take care of the problem with executing the command. Its solution, “Should I train more?" is produced on line 12. When the solution is approved on line 13, the system builds a new plan that incorporates the solution on line 18 that is then evaluated for execution. In this way, an agent was able to use a proposal pattern to resolve the conflicts with the command.

Another example of incompatible instruction resolution can be found on line 66 in which an engineer is sent to build a training camp. There are not enough engineers to fulfill this order, so the agent asks for permission to take an engineer away from another

action, “May I take one away from mining?" on line 74. In this case, the agent uses a proposal to overcome the conflict with the instruction. However, as opposed to the previous example, the agent is denied permission on line 75, but is given a different methodology for overcoming the conflict. The agent is then able to use this new methodology, combine it on line 87 with the original action and send it for execution on line 90. Other examples of instruction problem resolution are omitted due to length restrictions.

Order Conflict Resolution

Order conflict resolution is a special case of instruction problem resolution that involves a conflict between multiple orders. Due to length considerations, discussion is omitted.

Objective Conflict Resolution

Objectives can conflict with themselves as well as orders. It is the assistant’s job to monitor these circumstances, detect any conflict and bring it to the attention of the manager. In some cases, the agent may refuse or reject specific orders because they go against objectives.

In the provided session, the human gives the order “mine titanium” on line 705 with reference to an engineer that is standing. However, when the agent evaluates the order, it realizes that energy is running low, so it rejects the order with the proposal “We should have the engineer build two more first” on line 722. In this way, an agent can reject orders and provide feedback when those orders conflict with shared objectives.

New Instruction Development

One side effect of the ability to discuss tasks and procedures is the ability to discuss new tasks and procedures. In this way, a human can give new orders through a sequence of orders or reasoning. This is called learning by description or learning by

discussion, depending on the patterns used. There are a variety of patterns and sequences, which lead to the learning of instructions. Therefore, due to length considerations, only one is outlined with reference to the provided session to prove that the agent is capable of this conversational capability.

On line 222, the agent is given the order “Create a squad.” However, the agent does not know how to perform this action. Therefore, it creates a clarification dialogue with line 230 with the question “How do I create a squad?” An answer is then provided to the agent, “First, train six soldiers” on line 231. This completes the clarification dialogue and builds a procedure for creating a squad. However, the statement "Then, group them together” on line 242 later is reasoned to connect with line 231 and the concepts are merged and the procedure is extended to include this new information.

Explanations

Explanations represent a specific type of question-answer pattern that provide the reasoning behind an action, objective or state of the world. The ability to communicate this reasoning is an important aspect of being able to communicate over a task-oriented domain. Explanations can be found within the provided session on lines 162, 176, 314, 596, 612 and 641. Further discussion is omitted to do length considerations.

Discussion

Discussion is defined as an extended multiple-participant communication focused around a particular topic or group of topics. In this case, a topic is realized as a concept or group of concepts. The actual patterns used within a discussion sequence vary greatly, however, the distinction of a discussion from other patterns are still important. There are two good examples of discussion within the provided session. The others are omitted due to length considerations.

The first discussion pertains to the amount of time reaming in the game. This discussion is started by the human with the statement "How much time is left in the

game?" on line 129. The agent, who does not have enough computational power or domain knowledge to exactly predict the outcome of the game, replies with “I’m not sure” on line 136. However, the human continues the subject with the question “More than 5 minutes?” on line 137. The agent, realizing that there is indeed more than 5 minutes in the game responds affirmatively, “Yes” on line 152. The human wants to know why there is more that 5 minutes, so they ask “Why?” on line 153, to which the agents responds with the reasoning behind the answer on lines 178 and 179. The human thinks about this response and concludes that there is 10 minutes left in the game. They question this conclusion to the agent on line 180, “So, 10 minutes?” The agent analyses this new conclusion and realizes that there should be much more than 10 minutes left, and formulates the argumentative response give on line 205. As can be concluded, this discussion did not have a specific goal in mind other than discussing how much time was remaining in the game.

The second discussion is over the difference in the number of training camps. This difference may have come from the forgetfulness of the human, or through actions or events that the human did not notice. The discussion starts with the question, “How many training camps do we have?” on line 557, to which the agent replies, “4” on line 566. This simple question-answer sequence is continued with the rejection of the answer using the statement, “But there were two before” on line 567. The agent evaluates this rejection using the justification provided on line 576 and formulates the response “Now there are four” on line 582. At this point the human wants to know why there were two and now there are 4 so they ask, “What happened?” on line 583. The agent responds with the actions that lead to this change on line 598 with “I built two.” The human wants to know why these were built, so they ask “Why?” on line 599. The agent determines why and responds, “In order to build squads faster” on line 614. The conversation continues for quite some time discussing the autonomy behind the action. In this way, discussion

can be observed which shifts from one concept to the next as the focus of the human shifts with new information.

Iterating Sets

Discussions can also be specialized, as is the case with sets. Discussions over sets can include advancing through the set, narrowing the set, and inquiring about properties within the set. Information seeking dialogue is a type of restricted discussion of sets.

In the provided session, a discussion over sets begins with a single question answer pair, “How many engineers do we have?” on line 652 and “23” on line 661. This creates the notion of a set of 23 engineers. The question “What are they doing?” on line 662 inquires information about the set. This question is paired with the response, “13 are mining crystal, 7 are mining titanium and 3 are standing” on line 667. A subset is created through the order “Show me the ones that are standing” on line 678. The agent realizes that it is impossible to show this subset, but rather than using an instruction problem resolution, the agent focuses on the first item in the set and shows it on line 659, along with the statement “Here is the first” on line 696. The discussion continues with the human inquiring about the history of the engineer, followed by an order with an objective conflict resolution. Once these sub-dialogues have completed, the human brings the focus back to the set with the statement “Next” on line 729, which advances the set to show the second engineer on line 737. After inquiring about the history, the human then asks, “How many are left?” and advances to the final engineer. This example demonstrates the ability to enhance normal discussions with expressive capabilities over unique objects such as set, combined with various multimodal abilities of concepts.

Negotiation

Negotiation allows each participant to propose, counter, offer and reject concepts until the concept is agreed on by all participants. Some negotiations center on making offers or the discussion of an attribute value such as price; while other negotiations center

on the definition of a particular concept, like a bill going through congress, where various party's interests are to be represented.

Negotiation in the provided session begins with the proposal, "Let's make missiles" produced by the human on line 281. The agent evaluates this proposal and realizes that it is not the best course of action to take. Therefore, the agent counters the proposal with "No, I think we should train upgraded soldiers, then build a hospital." At this point, the human wants to know the justification for this, so they introduce an explanation sub-dialogue, "Why?" to which the agent responds, "Because it is faster and easier than missiles." The human is not satisfied with this justification and reasserts the missile proposal with the statement, "But, I really want missiles." This statement also expresses a strong desire to have missiles. The agent, realizing the rejection of its proposal and the desires of the human, tries to make a compromise with "How about we train upgraded soldiers then build missiles?" Success, the human accepts this new proposal and the objective finds agreement.

Mutual Planning

Mutual planning allows each participant to propose, accept, reject, refine and counter propose a shared plan until all participants agree and commit to the plan. This can involve discussions about the various aspects of the plan as well as the justifications of the various acceptances and rejections.

Mutual planning takes place within the provided session through following the negotiation example above. On line 329, the human agrees to the mutual objective of training upgraded soldiers then building missiles with the statement, "Alright, what do we need to do for upgraded soldiers?" This statement brings the focal attention of the participants to discussing a plan for obtaining upgraded soldiers. In addition, it queries the agent with taking initiative over the planning. The agent responds to the question on line 355 with, "First, we should build a research lab." The human looks around the camp

and notices a good spot for the research lab to go. They then respond with the proposal, "How about just north of the vault?" The agent evaluates this proposal, finds it satisfactory and acknowledges its agreement. Then the agent brings focus to the next step in the plan, "Then, we'll need to research explosives."

At this point in the plan, the human shifts the focus to a new order to attack the enemy. Several sub-dialogues occur to fulfill this order and the human returns focus to the plan with the statement, "Where were we?" on line 400. The system finds the context and reasserts the current plan and focus with lines 405 and 407. The human approves this plan, "Sounds good" and questions, "How long will it take?" The agent answers, "1630 cycles." The human then acknowledges the answer and tells the agent to execute the plan. This shifts the autonomy of the agent to handle the plan, and removes it from the focus of discussion.

Interruption

Interruption is an important component of any interactive paradigm. There are two essential aspects of interruption. First, there is the shift in focus to a new topic and then secondly there is a return of focus back to the originating topic. The shift in focus to a new topic in TCL is performed through the initiative of either the human or the agent. The current focus is recorded within both the shared concept graph and the current obligations.

The provided session demonstrates an interruption within TCL through the human's statement, "Go attack the enemy" found on line 369. Before that shift, the focus was on developing a plan to create upgraded soldiers. From there, there is an incomplete instruction resolution dialogue followed by another order. After that, the human brings the focus back to the plan through the statement, "Where were we?" on line 400. The system performs this continuation of focus by finding any obligations. In this case, an obligation of an answer to the shared plan is found. The system first determines the

originating nature of the obligation and states it, “We were planning to build a research lab just north of the vault.” Then it reasserts the operator that caused the obligation, “We then need to research explosives with it.” In this way the plan is continued.

Feedback

Opening up the task-oriented domain to communication and interaction allows interesting new capabilities to be introduced. One of these is feedback, the capability of one participant to respond to another participant’s communication or actions. The response is generally a comment, or reaction to the participant rather than previous responses such as answers to questions and so forth. TCL is capable of modeling feedback on a variety of scales including: polarity, satisfaction, confidence, agreement, relative and some emotion. More on each of these types is described in the feedback section of the partial TCL specification found in appendix A. Feedback generally falls into one of two categories, positive feedback and negative feedback, both of which are found within the provided session.

Negative feedback is introduced by the human on line 478 with the statement, “That’s not good enough!” The system recognizes this statement as dissatisfaction and tries to find with what the human may be dissatisfied. The dissatisfaction of a concept is then transformed into a problem concept on line 481 and the problem is evaluated by the agent on line 483. The agent eventually produces a solution that the human agrees with and they put it into motion. Positive feedback is introduced by the human on line 644 with the statement, “You are right.” The system recognizes this statement as an affirmation of correctness. This is transformed by the system into the acceptance of an earlier statement and evaluated as such by the agent.

The agent is not given much opportunity to provide only feedback to the human. This is because in the current sessions, the agent performs all of the actions while the human only observes and communications. However, the agent is still able to provide a

lot of feedback when it is attached to the rejection of proposals and actions, such as lines 290 and 718. This type of feedback is falls under justification.

Adjusting Autonomy

Another capability opened up to the task-oriented domain is the ability to discuss and adjust responsibility and permissions. This type of capability is grouped under adjusting and discussing autonomy. Autonomy represents the agent's ability to act under its own knowledge and experience. Responsibility and permissions define when the agent should ask the human participant for help or direction; as well as what the agent is or is not allowed to do.

Permissions are generally requested, granted, given or denied. In TCL, the agent may use the pattern for request-grant-deny to ask the human for a permission. If the agent is granted the permission then the agent may follow its intent, such as executing an action. If the agent is denied the permission then the agent must reevaluate the intent with another course of action, perhaps even abandonment or rejection.

This request-grant-deny pattern may be seen within the provided session starting on line 66 where the human asks the agent to carry out an action, "Send and engineer to the northeast to build a training camp." The agent is unable to complete this action because there are insufficient free engineers. Therefore, the agent asks permission on line 74 to take an engineer away form another task, "May I take one away from mining?" The human denies the request, "No" but also adds methodology, "…use a new one, as soon as possible." When the agent re-evaluates the intended action it incorporates the new methodology and realizes that by creating a new engineer, it can overcome this problem. Permission may also be queried like on line 615, "Who told you that you could?"

The responsibility for individual actions may also be shifted during the adoption or acceptance of proposals as well as answers during queries. The following examples

demonstrate this shift in responsibility within the provided session. On line 487, the agent proposes, "We could build more training camps" as the solution to a problem of not building soldiers fast enough. The human produces the response, "Take care of it" which implies that the agent should carry out the plan of building more training camps. Another accept and responsibility shift is performed by the human on line 427 with "Ok, do it" which implies that the agent should carry out the plan of building a research lab and researching explosives. The decision of a question is delegated back to an agent on line 513, when the response to "How many? What should they mine?" is met with "You decide."

Corrective Dialogues

The ability to correct previous statements, including the meaning or the message itself, is one of the interesting aspects that interaction provides the task-oriented domain. Such corrections often require reinterpretation that must have access to a history of the conversation as well as a history of the interpretations and actions performed based on those interpretations. For instance, if a particular statement lead to an action, and then that statement is corrected to mean something else, or the statement is removed such as, "I take that back". Then not only the statement needs to be reinterpreted or removed, but also the operator that resulted may need to be removed or undone, including such things as the consequences of actions and other causalities. Therefore, the shared concept graph is an essential component of the task-model. In TCL, the shared concept graph not only allows for correction, but it also allows for the discussion of interpretations.

In the provided session, the human requests, "How many engineers are left?" on line 789. Because the participants were enumerating the subset of engineers that were standing the agent uses this as a context to answer the question on line 805, "You have seen all of the engineers that are standing." The human then rejects this answer with a corrective statement, "No, I mean in the game." If the agent were intelligent, it may use

this information to realize that 'in the game' is a superset of 'engineers that are standing' and that the human wants to expand the answer to all engineers. However, what actually happens within this particular provided session is that the agent rejects the first interpretation of the question, thus dropping the context attachment and re-evaluates the query. It does take into account the added restriction of 'in the game'. However, this restriction includes everything and is thus not a real restriction.

Another example in the provided session begins on line 633 when the human states, "No, I didn't" in disagreement to the fact that the agent is claiming that the human gave the agent permission to build more training camps. In its own defense, the agent finds the statement that it believes granted responsibility to build the training camps, and asks forms an explanation of the interpretation. This is produced by the system on line 643 as "Oh, I thought that's what 'Take care of it.' meant." It is important to note that the agent has no idea of the actual linguistics of the message, e.g. 'take care of it', but rather knows that there is some form it out there that the dialogue model knows. Although this example demonstrates the ability of the agent to provide interpretation-based justification, it is not corrective because the human then agrees with the argument by stating, "Yes, you are right."

The recognition of misinterpretation is not always provided by the human participant but sometimes through the dialogue model as well. However, it was the purpose of this section to cover corrective dialogues.

Multimodal Interaction

One last important aspect of interaction worth mentioning is that interaction need not always be spoken or written. Multimodal interaction applies to interaction through a variety of means including gestural, visual, auditory, written and so forth. Although the designed system is limited to written text, the agent may also observe selection with the mouse and may manipulate the graphical display to show various viewpoints. Even in

the provided session, the agent needed to detect the current viewpoint of the user in order to help resolve such statements as ‘to the west’ or ‘to the northeast’. These statements were ultimately modeled as directions relative to the current view of the user. In addition, the agent was able to shift the view to show various objects on lines 219, 695, 737 and 782.

Enhanced Agent Capabilities

The intelligent agent is able to gain several important capabilities through interaction with a human participant in a task-oriented domain. Although these are not communicative or interactive competencies themselves, communication or interaction enhances these agent abilities.

Reinforcement Learning

The interaction provided through the dialogue model allows the human to provide direct feedback on the actions and communication of the agent. Through this feedback, the agent is able to learn what the user likes and dislikes as well as gain an empirical rating for its performance in certain key areas. This is accelerated by the fact that the dialogue system is responsible for figuring out what the feedback applies to, so that the agent can utilize this feedback without the need for interpretation.

Knowledge Learning

Although knowledge learning is a general capability of agents, the interaction provided by the dialogue model allows new ways in which an agent can learn. In the ‘new instruction development’ section of command and control, the agent was able to learn new procedures by communicating with a human about them. Furthermore, the discussions of several methodologies in the ‘order conflict resolution’ section expanded the action knowledge of the agent.

In addition, the agent is able to absorb information told to it by the human participant through the dialogue system. For instance, in the provided session on line 255 the human introduces nomenclature, "The group is called a squad" which the agent is able to learn and incorporate into future commands. Thus, future references to squad were understood to mean future references to a group of soldiers.

Learning User Preferences for Adapting Behavior

In addition to reinforcement learning and absorbing direct knowledge, the agent is able to build a user model based on various responses such as answers, justifications and even frequencies of operators. For example, in the provided session, in connection to "Send an engineer to the northeast to build a training camp" the human adds the methodology "use a new one, as soon as possible." Then in the next order, "Also send one to the south to build a training camp" the agent detected a similarity between that order and the previous order to the northeast. The user model suggested that the methodology of creating a new engineer and giving the action a high priority would be a good idea. However, in this case the agent wanted to confirm this methodology with the user and therefore generated the statement "A new one, as soon as possible?"

Another example demonstrates repetitive actions in a given context. The human asked for the history of an engineer being displayed both on lines 697 and 738. When it came time for the third engineer to be displayed on line 782, the agent voluntarily provided this information to the user.

Operator and Concept Layering

An important property that was used in the previous section that should be formally discussed is the ability of the dialogue model to build layers of interaction. It is important to note that these layers are not a requirement of the TCL framework, but rather a product of building the TCL dialogue rules following a specific design.

Layers are constructed by utilizing rewriting rules to recognize and interpret incoming messages and cast them for a higher layer. For example, questions and answers are on a specific layer. When an incoming message is detected as an answer, rather than having the question-answer layer deal with it directly, the message itself is transformed into an answer, which is then passed to the question-answer layer to be dealt with accordingly. Similarly, there are recognition rules that translate messages into rejections, denials, and so forth. The separation of recognition in lower layers and subsequent connection in higher layers allows the higher layers to deal with the handling while reducing the complexity of the overall system.

Further layers are created by interpreting the closure of a pattern and forwarding the intent to the next step in the higher-level pattern. For instance, sub-clarification dialogues are constructed by asking questions about a particular topic, once the answer is received that specific clarification is closed and the system uses handler functions to revert to the original topic. The shared concept graph is essential in finding the original context of the concepts before the sub-clarification dialogue took place. The use of layers was described within the multiple participant command and control as well as the multiple participant question and answers sections above.

Another layer is created by allowing the agent to use specific patterns such as request-approve-deny or propose-counter-accept-reject. The agent's ability to control the focus of conversation presents the agent its own layer, which can then be abstracted even further within the agent.

Synopsis

The goal of this section was to use evidence from the included example human session in appendix C to reinforce the thesis that a single language created between natural language and agent behavior reasoning could be constructed in such a way to unify the various dialogue models used in literature. This was accomplished by

demonstrating many of the conversational capabilities within a real TCL session, including: command and control, information seeking, notification and bother, clarification dialogues, explanations, discussions, negotiating, mutual planning, interruption, feedback, adjustable autonomy, corrective dialogues and more.

Furthermore, the TCL system is designed in such a way that it is independent from the various conversational capabilities presented. Therefore, new capabilities may be added through introducing the appropriate concepts, operators, rules, intents and layers accordingly.

CHAPTER 6
DISCUSSION

This chapter concludes the body of research presented in this dissertation. The chapter begins by discussing the implications this research has on related fields. Then major upcoming problems are discussed, followed by a short summary and conclusion.

Related Fields

The theories and technology developed in this dissertation apply to many areas that involve either intelligent agent interaction or dialogue modeling. This section will discuss the implications affecting some of these areas including agent-agent communication, business process management, the semantic web, dialogue management and human robot interaction.

Agent-Agent Communication

This research was able to present a great increase in the conversational capabilities and expressive power of task-oriented communication, including reducing the interpretative and dialogue-modeling load of the agent into a separate system. This has great implications for agent-to-agent communication. However, the shared medium, the task concepts and operators, is the core medium of interaction. Therefore, either a translator would have to be developed to translate this medium into a form native to the intelligent agent, or the intelligent agent would have to adopt the medium. Although the medium may become a standard, it is extremely volatile and changes quickly as it is expanded by varying parties encompassing more and more 'practical' features. Therefore, it would be hard to use as the foundation for communication among heterogeneous agents. However, using homogenous agents, or at least agents having homogonous communication components, the TCL framework would be a fine candidate.

In addition to providing ACL conversational modeling to multiple agents, the core TCL operators may be mapped to specific performative-style speech acts. This mapping allows either the KQML or FIPA-ACL transport layer to be used with little effort.

Business Process Management

Business process management is currently a very popular area in both industry and academia. The management of business processes represents the ability of an organization to optimize or adapt their business processes as the organization changes. The maturity of the field has produced semantic representations for the modeling of business processes. These models often refer to the communication between customers and internal mechanisms. Consider the action-workflow-loop of [31]. In this workflow, a customer places a request for work, the performer then does the work, the performer then reports that the work is done to the customer and the customer then acknowledges. This type of interaction can be seen as higher-level layers of the conversational patterns discussed in the last chapter. Furthermore, an agent can be used to fill many of these roles both in implementation and in the simulation of individual business processes.

The Semantic Web

Another very popular area is the Semantic Web. The semantic web attempts to take the paradigm of the World Wide Web and enhance it with semantics, giving well-defined meanings to the information contained within. It is hoped that this new semantic web will allow both humans and agents to find, share and combine information more effectively. The semantic web relies heavily on machine-readable information and ontology markup languages that encode the structure of this information so that agents can understand it more easily.

There has recently been a push to create an agent based front-end for the semantic web. One that would allow a human to use an agent to access the web much like the manager-assistant paradigm used in this dissertation. However, rather than the basing the

shared medium on the planning, management, monitoring and execution of task and procedures, the shared medium would have to be modified for the search, sharing and combination of information in semantic form. The TCL based approach shows promise for performing this role as long as the shared medium concepts are presented in a way that can be easily rationalized and communicated by a human participant. Because the task communication language was based on the theory of practical communication language, it is highly promising that such a new language can be created.

Dialogue Management

The TCL framework shows great promise in improving the theoretical aspects of dialogue management. The most important contribution to dialogue management is the introduction of shared concepts, including the shared concept graph. The ability of a dialogue manager to go beyond mere dialogue tags and operate on complex structures is an essential part of why TCL is so successful. Another important component is the reliance on a reasoning engine for dialogue management rather than a specific algorithm. This allows the dialogue manager to quickly expand and adapt to new features, as well as the ability to layer dialogue protocols on top of one another.

Human Robot Interaction

Human robot interaction shows the most promise for the TCL research area in the next five to fifteen years. Human robot interaction is the study of the interaction between humans and robots, including psychological, sociological and engineering aspects.

Current research in robotics has been dealing with very specific robotics issues such as navigation, grasping, moving obstacle detection and avoidance, object and facial recognition, even multiple robot or human-robot object balancing. There has been a great push for an anamorphic robot capable of interacting directly with humans in daily life. If these robots are expected to interact with humans in activities of every-day living, much greater communication and interaction technologies are needed. The TCL theory

developed in this dissertation shows great promise for integrating many models of communication into a single working human-centered application.

Future Work

This section attempts to outline some of the major upcoming problems that will limit the TCL theory. These problems are beyond the scope of this dissertation. However, they need to be addressed and solved for the future success of TCL as well as many communication-enabled agents.

Understanding contexts

Various concepts and operators were connected in this dissertation using modifier keywords such as 'first, then, else'. This allowed the system to recognize when a concept referenced a previous concept and when a certain focus was completed. However, without these modifier cues the problem becomes much more complex.

Consider the following dialogue sequence: The human orders, "Create two squads." to which the agent replies, "How do I create a squad?"

In the first variation, the human would then state, "Train 6 soldiers." Then, "Have them attack the enemy." Is the second statement about attacking the enemy tied to the answer of how to create two squads or it is a separate order that is given after the answer has completed? In other words, is it "Train 6 soldiers in order to create a squad. Now go attack the enemy" or is it, "Train 6 soldiers and have them attack the enemy in order to create a squad."? As a human, one can postulate that it is probably the first, because attack and create are different concepts. Nevertheless, how is an agent to know this?

In an even harder variation, what if the original statement was "Create 2 squads" instead of "Create a squad." Furthermore, what if the second statement was "Group them together" rather than "Have them attack the enemy." Would it then be "Train 6 soldiers and group them together to create a squad. Create two squads" or would it be "Train 6 soldiers to create a squad. Create two squads and group the squads together." In the

second variation, future references to the squad or groups of squads may hint about how this should be interpreted. However, the agent cannot rely on this information because it is unpredictable.

In order to understand the differences in these variations successfully, the agent must place much more reasoning behind reference resolution, including the intentions, outcomes and consequences of all of the various interpretation possibilities.

Shift in control

The research represented in this dissertation represents a shift in control. The dialogue manager is no longer the center of control for the communication of an intelligent agent, but rather acts as a pass-through creating TCL for every user utterance or gesture and generating output when receiving TCL. The task-communication model is now the enabling force in the communications aspect of the agent, and the agent is given control when to send messages.

Mixed-Initiative Interaction

Mixed initiative interaction is the interaction among multiple participants in which any participant may take the conversational lead. This is an important aspect in the future of human-computer interaction. When should the agent take the lead in the conversation and guide it toward a specific objective? When should the agent leave the conversation open and follow the lead of the other participants? There are many researchers working on these questions. However, their work needs to be incorporated within the agent, or within the TCL dialogue model.

Turn Taking

Another aspect of multiple participant conversation is turn taking. Turn taking is the ability to detect when another participant is speaking or has the conversational floor, and waiting until that participant is done, or the conversational floor is available. There

have been a great many studies on how the turn is passed or kept. However, these types of ideas need to be incorporated within the TCL framework for the agent to communicate successfully in mediums that rely heavily on turn taking such as speech.

Summary

This dissertation addressed the problem of the overwhelming variety of ways intelligent agents rationalize communication by integrating many of the disparate dialogue models found in human-human, human-agent and agent-agent communication into a single model. Sound theory, including the practical communication language hypothesis was created. Several major enhancements were introduced to dialogue management including the formation of the meaning-action concept and its use as a shared medium as well as the introduction and incorporation of the shared concept graph. The ideas behind the speech-act and dialogue-act were extended by introducing behavior-based operators and allowing the operators to apply to specific, structured and well-defined concepts. An accompanying engineering methodology was defined for the construction of concepts, operators and rules that create the language and model of a specific domain, including methodology for the verification and validation of that language and model. This practical communication language methodology, based in part on the theory rational communication, was used to construct a task-based language and model called the task communication language framework. This framework was then implemented within an intelligent agent in a real-time resource management simulation. A sample output listing from actual human interaction with that implementation was used to demonstrate that the resulting framework did indeed incorporate many of the disparate models of communication and their corresponding capabilities. This provided a proof of concept, proving the thesis: there exists a language between that of human natural language and the behavioral reasoning of an intelligent agent, and that this language is capable of not only unifying the various models of communication, but also provides the

foundation for a theoretical framework for an engineering methodology for building models of conversational capabilities.

Conclusion

It is unknown why so many researchers continually introduce new features of communication without working to integrate those features into a common foundation, let alone a working prototype. The research presented in this dissertation attempts to solve this problem by providing a common foundation for the introduction and implementation of new communication capabilities within an intelligent agent for experimentation in both human-agent and agent-agent communication.

The results from this dissertation show great promise for the future capabilities of intelligent agent conversation and consequently human robot interaction, human computer interaction, user interfaces, intelligent agents and many other unforeseen areas.

It is the hope of this author to continue to push towards the advancement of these conversational capabilities in an effort to motivate the advancement of speech recognition, and natural language understanding.

APPENDIX A
PARTIAL TCL LANGUAGE DEFINITION

This appendix provides a subset of the task communication language including all of the concepts used in the annotated example human session in appendix C. The appendix is broken down into four sections. First, abstract concepts are described followed by both interaction and agent abstract operators. Helper functions are then defined, including how they work within the structure of the shared concept graph, followed by macro functions.

Abstract Concepts

Abstract concepts of the practical communication language theory are defined in Chapter 3. They represent the individual meaning-action concepts that are shared across the communication-behavior spectrum, realized in the mind of the individual participants in a conversation. Because of this shared mindset, it is essential that these concepts can be naturally conceptualized by a human participant.

These concepts relate directly to both task orientation, such as goals and actions, as well as some common domains such as time, space and causality. These concepts are defined in individual sections organized by their commonality. Each section builds upon the previous until an entire task model is constructed.

Notation

$$\begin{array}{c} Identifier \\ extends: Parent \\ casts: Facade \end{array} \quad \begin{cases} requiredAttribute^{R} \leftarrow \{allowed\_type1, allowed\_type2\} \\ optionalAttribute \leftarrow \{allowed\_type\} \end{cases}$$

Figure A1: Abstract Concept Key

The notation used for defining each concept is illustrated in figure A1. The abstract identifier is found on the left-hand side of the figure, in this case 'Identifier'. Beneath the identifier is an optional extends field that designates the parent concept in an is-extension-of relationship. All concepts have an is-extension-of relationship with the parent abstract concept. However, this particular relation is implicit and is not specified. An optional casts field is also shown beneath the identifier. This field designates any can-cast-as relationships. This relationship implies that the concept may be used in composition under the signature of the Façade concept. For example, an ActionSequence concept may be used as an Action concept under composition.

On the right-hand side of the figure is the signature, the list of attributes and their associated types. A superscript 'R' denotes that the attribute is required and a superscript 'M' denotes that the attribute takes multiple values. In some cases, the allowed type is a value type, realized as an attribute-less abstract concept. In most cases, a list of these possible types will be provided beforehand.

The same notation is used for all four types of abstract operators.

Core Types

In order to begin construction of the task model, several core types must first be defined. These core types create the foundation of which other conceptual notions can be modeled. The types: numeric, integer, percentage, accuracy and participant are straight forward as defined below.

Type: numeric ← any representation of a real number, including decimals and fractions.

Type: integer ← an integral representation of a real number.

Type: percentage ← numeric that is between 0 and 100.

Type: accuracy ← {'approximate', 'definite', 'average'}

Type: participant ← {'agent', 'human', 'you', 'me', 'opponent', 'us', 'them'}

Once those types have been defined the root abstract concept, the parent of all concepts, may be defined.

$$Concept \begin{cases} confidence \leftarrow \{percentage\} \\ who \leftarrow \{participant\} \end{cases}$$

Figure A2: Abstract Concept – Parent Concept

The core abstract concept is illustrated in figure A2. The 'who' attribute of the concept defines the owner, or producer, of the concept and the 'confidence' attribute relates their individual confidence in the concept. As with the other concepts in this section, other attributes have been removed due to length considerations.

Once the abstract concept has been defined, a specialized type, 'concept_attribute', can be defined. This type refers to a specific attribute within an existent concept. The 'concept_attribute' type is similar to a pointer, which points to the storage of an attribute rather than the attribute value itself.

Type: concept_attribute ← an attribute in a concept; may pass through composition.

$$Object \begin{cases} name^{R} \leftarrow \{identifier\} \\ history \leftarrow \{(Concept)\} \end{cases}$$

Figure A3: Abstract Concept – Object

The object concept, illustrated in figure A3, is the parent of all objects in the task model. The generic notion of an object has been defined in chapter 4. Each object has a name, or identifier, as well as a history. The history is entirely application or context

dependent and may range from a list of actions that acted upon the object to a list of events that the object participated in.

Once the core concepts have been defined, the Conjunction concept and the Disjunction concept start to form a basis for the structure of complex concepts.

$$Conjunction \begin{cases} first^{R} \leftarrow \{(Concept)\} \\ second^{R} \leftarrow \{(Concept)\} \end{cases}$$

Figure A4: Abstract Concept – Conjunction

Conjunction is typically used for placing multiple concepts within the same context or container concept, but can also imply to multiple simultaneous interpretations.

$$Disjunction \begin{cases} first^{R} \leftarrow \{(Concept)\} \\ second^{R} \leftarrow \{(Concept)\} \end{cases}$$

Figure A5: Abstract Concept – Disjunction

Like conjunction, disjunction is another way to group concepts together. Typically, disjunction implies either choice or ambiguity.

Modifiers

One of the most fundamental essentials to building a solid task model that can model the rich expressiveness of natural language is to account for the modification of concepts through various notions. The modifier concept represents that additive information by leveraging the following types.

Type: modifier_context ← {‘another’, ‘other’}

Type: modifier_existent ← {‘old’, ‘new’}

The modifier_context type hints that the agent should not reference recently accessed concepts, while the modifier_existent type hints whether the agent should look at previously defined or non-existent concepts.

Type: modifier_attachment ← {‘also’, ‘but’, ‘furthermore’, ‘therefore’, ’except’}

Type: modifier_sequence ← {‘first’, ‘then’, ‘finally’}

The modifier_attachment and modifier_sequence types hint to the dialogue model that previous or future concepts should be referenced in relation to the current concept.

Type: modifier_ownership ← {‘our’, ‘their’, ‘my’, ‘your’}

The modifier_ownership type implies which collection or sub-collection of objects to search when finding a reference.

Type: modifier_selection ← {‘all’, ’each’}

The modifier_selection type allows for entire groups of concepts to be referenced in varying ways.

Type: modifier_negation ← {‘no’, ‘do-not’, ‘not’}

The modifier_negation type allows the concept to be complimented without requiring a complex function to handle the complementation of every known concept structure.

Type: modifier_condition ← {always, never}

The modifier_condition type assists the agent in establishing continual concepts or constraining concepts.

Type: modifier ← {modifier_context, modifier_existent, modifier_attachment, modifier_sequence, modifier_ownership, modifier_selection, modifier_negation, modifier_condition}

Other than trying to figure out how an agent should respond to a particular modifier type, the modifier concept is straightforward. The ‘content’ attribute represents the concept being modified and the ‘modifier’ attribute represents the type of

modification being performed on the concept. It is important to note that the modifier concept casts to the type of its content. Therefore, it may be used in place, as a container, for a concept of almost any type.

$$Modifier \quad \begin{cases} content^{R} \leftarrow \{Concept\} \\ modifier^{R} \leftarrow \{modifier\} \end{cases}$$

Figure A6: Abstract Concept – Modifier

Quantization

An essential aspect to developing a foundation for a model is the ability to represent cardinality to any concept, whether exact or suggestive. However, before any concepts may be defined, several cardinality types must first be defined.

Type: custom_numeric ← {‘none’, ‘a couple’, ‘a few’, ‘some’, ‘most’, ‘almost all’, ‘all’}

Type: numeric_relative ← {‘more’, ‘less’}

Type: numeric_change ← {‘increase’, ‘decrease’, ‘same’}

Type: numeric_required_relative ← {‘insufficient’, ’sufficient’, ’deficient’, ’excess’}

The numeric type is expanded to represent ambiguous or interpretable numeric representations in custom_numeric. Furthermore, numeric_relative, numeric_change and numeric_required_relative add the notion of comparability to cardinality. These types represent some of the ways to express cardinality in natural language.

The Quantity concept, illustrated in figure A7, represents the duplication of a concept, or a group of concepts with a specified size. The ‘value’ attribute represents the number of concepts while the ‘content’ relationship represents the concepts that are quantized. Both of these attributes are required. The ‘accuracy’ attribute may also be added indicating the accuracy of the value provided. For relational purposes, a Quantity

concept is equivalent to nested conjunctions of the same content that includes the same number of duplications as the given value.

$$Quantity \begin{cases} value^{R} \leftarrow \{numeric, custom\_numeric\} \\ content^{R} \leftarrow \{Concept\} \\ accuracy \leftarrow \{accuracy\} \end{cases}$$

Figure A7: Abstract Concept – Quantity

$$\begin{matrix} ChangeQuantity \\ casts: Consequence \end{matrix} \begin{cases} content^{R} \leftarrow \{Concept\} \\ type \leftarrow \{numeric\_change\} \end{cases}$$

Figure A8: Abstract Concept – ChangeQuantity

One of the many relational quantity concepts is illustrated in figure A8. This particular concept, the ChangeQuantity concept, allows the model to represent a change in the number of a concept. This particular concept may also be cast as a Consequence type, which means it may fill the consequence attribute of any concept.

Sets

The last essential aspect in developing a foundation for a model is the ability to represent sets beyond the ability of conjunction, disjunction and quantification. There are two important absolute sets, ‘everything’ and ‘nothing’. Everything represents the set of all things, or every concept in the model. Nothing represents the empty set, a set with no concepts. Other custom sets can be defined per domain such as ‘everyone’, which is the set of everything such that the thing is a person.

Type: set_absolute← {‘everything’, ‘nothing’}

Type: custom_set ← {'everyone'}

$$Set \quad \{ cardinality \leftarrow \{integer\}$$

Figure A9: Abstract Concept – Set

The set concept is constructed in several ways, however all methods share a common foundation in both casting and functionality. Therefore, all sets share a common parent known as the Set concept, as illustrated in figure A9.

$$\begin{array}{l} EnumeratedSet \\ extends: Set \end{array} \left\{ \begin{array}{l} first \leftarrow \{Concept\} \\ second \leftarrow \{Concept\} \\ third \leftarrow \{Concept\} \\ \ldots \end{array} \right.$$

Figure A10: Abstract Concept – EnumeratedSet

One of the ways to represent a set is through enumeration in which each element of the set is specifically listed. The enumerated concept, illustrated in figure A10, is built in such a way. The enumerated set is useful for when each concept is listed through language or for structure such as a sequence of actions. The enumerated set extends the set concept, which means that the set concept is a parent concept and the enumerated set is a child concept. All of the attributes of the parent concept, set, is included within the extended set. Furthermore, any attribute that contains a set concept may contain an enumerated set concept as that attribute.

Another way to represent a set is through restricting a particular set with a specific constraint. The subset concept is constructed just that way. The concept has a 'superset'

attribute that represents the original set and a ‘restriction’ attribute which represents how to filter out the original set, the concepts that do not get filtered out become the new set.

$$\begin{matrix} Subset \\ extends: Set \end{matrix} \begin{cases} superset^{R} \leftarrow \{Set\} \\ restriction^{R} \leftarrow \{Concept\} \end{cases}$$

Figure A11: Abstract Concept – Subset

Many dynamic sets are modeled in this way within the Stratagus agent. For example, ‘engineers’, ‘soldiers’ and ‘training camps’ are all subsets of everything such that the thing is a specific type.

Other concepts have been created which are equivalent to union, intersection, set difference and so on, but are not defined here due to length considerations.

Time

Now that the essential elements of the task model have been defined, various theories can be constructed, such as time, space and relation. Time is a necessary component of a task-oriented model, from the duration of actions to the gathering and expenditure rates of resources. Time in the Stratagus domain is measured in the number of cycles from the beginning of the game. This provides the agent with an absolute scale in which to measure a timeline.

Type: time_absolute ← {‘begin’, ‘now’, ‘end’}

Certain specific absolute times are predefined. ‘Begin’ or ‘beginning’ refers to the absolute beginning. In Stratagus, this would be cycle 0, the beginning of the game. ‘End’ refers to the end of time. In Stratagus, this would be the end of the game, which may not yet be known. ‘Now’ refers to the current time, or in Stratagus, the current game cycle.

Type: time_relative← {‘before’, ‘after’, ‘earlier’, ‘later’}

Relative time provides a means for expressing chronology, or actions, events and states in chronological order.

Type: rate_absolute← {‘fastest’, ‘slowest’}

Type: rate_relative← {‘faster’, ‘slower’}

Rates provide a means for expressing the gathering or expenditure of resources as well as differences in the duration of actions. For example, if an action took to long to perform a participant may have a desire to increase the speed of the action, or make the action ‘faster’.

Type: time_unit← {‘cycle’, ‘minute’, ‘second’, ‘hour’, ‘day’…}

Type: custom_time_length ← {‘soon’}

Type: time_length← {numeric, (Quantity <content {time_unit}>)}

Various time units are constructed through the Quantity of a particular unit. This provides a simple means of constructing units, but also allows the agent the ability to reason about units the same as they would reason about any other concept.

$$Timespan \begin{cases} begin^{R} \leftarrow \{time\_absolute\} \\ end^{R} \leftarrow \{time\_absolute\} \\ value \leftarrow \{time\_length, custom\_time\_length\} \end{cases}$$

Figure A12: Abstract Concept – Timespan

The Timespan concept, illustrated in figure A12, is capable of measuring an amount of time that passes between two events. This is essential in allowing actions to have duration.

Space

Space is another necessary component of a task-oriented model from providing locations or directions of actions to calculating size and distances. Space in the Stratagus domain is measured in the number of cells. This provides the agent with an absolute scale in which space can be measured. Space is used in the Stratagus domain in order to select locations to construct buildings, explore, mine and attack.

Type: space_set_absolute ← {‘everywhere’, ‘nowhere’}

Type: space_set_relative ← {‘bigger’, ’smaller’}

Space has a set component, which is represented by the space_set_absolute and space_set_relative types. The largest space set is ‘everywhere’ which represents all space, while the smallest space set is ‘nowhere’ which represents the empty set of space. In addition, relativeness is given through notions of ‘bigger’ and ‘smaller’.

Type: space_reference ← {‘the-world’, ‘current-reference’, ‘current-location’}

Space also has a reference type which changes over time. ‘the-world’ refers to the ‘everywhere’ set, ‘current-reference’ refers to the location of the currently referencing object and ‘current-location’, refers to the current location under focus.

Location {

Figure A13: Abstract Concept – Location

The notion of space is constructed first through the notion of a point. In TCL, this is known as a Location concept, illustrated in figure A13. This concept is the parent of all of the various ways to represent a location, including a point in space, a direction or even a restriction with reference to another location.

Type: length_unit ← {‘cell’, ’meter’, ’mile’}

Type: length_absolute ← {numeric, (Quantity <content {length-unit}>)}

Type: custom_length_relative ← {‘near’, ‘far’}

Type: length_relative ← {‘longer’, ‘shorter’}

Type: length ← {length_relative, length_absolute, custom_length_relative}

Distance is measured as the length between two locations. Just as ‘duration’ was measured as a Quantity of time units, Distance is measured by a Quantity of space units.

Type: direction_unit ← {‘degree’}

Type: direction_absolute ← {numeric, (Quantity <content {length-unit}>)}

Type: custom_direction_absolute ← {‘north’, ‘east’, ‘south’, ‘west’, ‘up’, ‘down’, ‘right’, ‘left’}

Type: custom_direction_absolute ← {‘north of’, ‘east of’, ‘south of’, ‘west of’, ‘above’, ‘below’, ‘right of’, ‘left of’}

Direction is measured based on a reference gradient. In Stratagus, north, south, east and west are well defined.

$$\begin{array}{l} DirectionLocation \\ extends: Location \end{array} \left\{ \begin{array}{l} direction^{R} \leftarrow \{direction\_relative\} \\ reference^{R} \leftarrow \{(Object),(Location)\} \end{array} \right.$$

Figure A14: Abstract Concept – DirectionLocation

A location can be defined in terms of a direction relative to another location. This is similar to a subset of space where the restriction is all space that is in a direction with reference to the location. The DirectionLocation concept, illustrated in figure A14, is such a location.

Another way to represent a location is through proximity. This is similar to a subset of space where the restriction is all space that is within a certain distance with reference to a location. The Proximity concept is such a location.

$$\begin{array}{l} Proximity \\ extends: Location \end{array} \begin{cases} distance^{R} \leftarrow \{length\} \\ reference^{R} \leftarrow \{(Object),(Location)\} \end{cases}$$

Figure A15: Abstract Concept – Proximity

$$\begin{array}{l} RelativeLocation \\ extends: Location \end{array} \begin{cases} direction^{R} \leftarrow \{direction\_relative\} \\ distance^{R} \leftarrow \{length\} \\ reference^{R} \leftarrow \{(Object),(Location)\} \end{cases}$$

Figure A16: Abstract Concept – RelativeLocation

The RelativeLocation concept combines both distance from a location and direction with respect to a location into a single concept. This is the equivalent of either Conjunction or Intersection of both Proximity and DirectionLocation where both reference locations are the same.

Other space concepts are used to create paths, measure area, define perimeters and so forth. Furthermore, the combination of time and space concepts can create notions of speed and flow. However, they are not discussed here due to length considerations.

Relation, States and Events

The final necessary component of a task-oriented model defines relations, states and events. These are critical in defining objectives, actions, and the state of the world.

Type: numeric_relationship ← {greater-than, less-than, at-least, at-most, equal}

Type: ontological_relationship ← {is-a, has-a, is-child-of, is-parent-of, is-similar-to}

Type: relationship ← {numeric_relationship, ontological_relationship}

Various relationship types provide a foundation for every way to compare two concepts or a concept to definable values. Only two types are discussed in this appendix.

$$\begin{matrix} MagnitudeRelation \\ extends: Relation \end{matrix} \begin{cases} relationship^R \leftarrow \{relationship\} \\ magnitude^R \leftarrow \{numeric, custom\_numeric, (Quantity)\} \\ reference^R \leftarrow \{(Concept)\} \end{cases}$$

Figure A17: Abstract Concept – MagnitudeRelation

The first type of relation is the MagnitudeRelation concept, as illustrated in figure A17. The 'reference' attribute is a concept that must be quantifiable. The magnitude of the concept is compared with the 'magnitude' attribute depending on the 'relationship' attribute. Relations have multiple purposes. If they are stated, then they are given as a fact, if they are queried then they are asked, and so forth.

$$\begin{matrix} CompareRelation \\ extends: Relation \end{matrix} \begin{cases} relationship^R \leftarrow \{relationship\} \\ reference1^R \leftarrow \{(Concept)\} \\ reference2^R \leftarrow \{(Concept)\} \end{cases}$$

Figure A18: Abstract Concept – CompareRelation

The second type of relation is the CompareRelation concept, as illustrated in figure A18. The 'relationship' attribute defines how the 'reference1' attribute is to be compared to the 'reference2' attribute.

The state in time concept attempts to encapsulate a state in the world with respect to a particular time in the world. The 'time' attribute defines the time, absolute or

relative, while the 'content' attribute refers to the state. In almost all cases, the 'content attribute' of a state is represented through a Relation concept.

$$\begin{array}{l} StateInTime \\ extends: State \end{array} \begin{cases} content^{R} \leftarrow \{(Concept)\} \\ time \leftarrow \{time\_absolute, time\_relative\} \end{cases}$$

Figure A19: Abstract Concept – StateInTime

$$StateChange \begin{cases} from^{R} \leftarrow \{(State)\} \\ to^{R} \leftarrow \{(State)\} \end{cases}$$

Figure A20: Abstract Concept – StateChange

The state change concept describes a transition from one state to the next. This concept is used in describing changes in the world, including consequences or side effects of actions.

Once relations and states have been defined, events can be constructed. Typically, an event represents a change in the state of the world, whether the beginning or end of an action or a change in some attribute of some object.

Type: event ← {begins, ends, completion, created, destroyed, capable, halted, changed}

A generic event type attempts to encapsulate the various changes in the world. The changes 'begins', 'ends', 'completion' and 'halted' refer to actions or the behavior of actors. 'Created' represents when a new object is introduced and destroyed represents when an object is removed. 'Capable' represents when an action can be executed, and 'changed' represents a change in an attribute or object.

$$Event \begin{cases} event^R \leftarrow \{(Concept)\} \\ type^R \leftarrow \{event\} \end{cases}$$

Figure A21: Abstract Concept – Event

A generic event type is able to represent the majority of events within TCL. The Event concept is illustrated in figure A21. The 'type' attribute is the event type as described above and the 'event' attribute is the concept that is checked against the type. For instance, if the event type was 'completion' then the event attribute would be an action, action sequence, plan or procedure.

Objectives

Objectives have been discussed in chapter 4 in the introduction of the task communication language. Due to length considerations, the specification provided below does not have all of the attributes discussed previously, but has enough to understand the basics behind an objective.

Type: objective ← {achieve, avoid, maintain, preserve, cease, test}

Objectives have varying types. 'Achieve' attempts to achieve an action or a state in the world. Conversely, 'avoid' attempts to avoid the action or state. 'Maintain' attempts to reach a state if it is not true and then keep that state true. On the other hand, 'preserve' only attempts to keep it true. If a preserved state fails the objective will terminate, rather than try to achieve the state. 'Cease' attempts to exit a state or action and 'test' attempts to test the states value.

Type: priority ← {'high', 'asap', 'low', 'normal'}

Objectives have varying priority that helps to resolve conflicts as they occur.

The Objective concept, illustrated in figure A22, covers the basic notion of an objective. The 'result' attribute is an action or state and the type describes what should

be done with that state. The 'priority' attribute allows objectives to supersede one another in case of conflicts.

$$Objective \begin{cases} result^{R} \leftarrow \{(Concept)\} \\ type^{R} \leftarrow \{objective\} \\ priority \leftarrow \{priority\} \end{cases}$$

Figure A22: Abstract Concept – Objective

One of the ways the agent receives objectives is through the expression of desire. The Desire concept, illustrated in figure A23, is a basic representation of an objective by wrapping the objective within a desire concept. The 'magnitude' attribute allows the participant to express urgency or the strength of desire, such as 'would be ok' or 'really want'.

$$Desire \begin{cases} objective^{R} \leftarrow \{(Concept)\} \\ magnitude \leftarrow \{custom\_desire\_magnitude\} \end{cases}$$

Figure A23: Abstract Concept – Desire

Actions, Procedures and Plans

Actions are discussed in detail in chapter 4. Due to length considerations, the specification provided below does not have all of the attributes discussed previously. However, it does include several new attributes that were not previously discussed.

Type: action_op ← {execute, halt, pause, resume, postpone}

The action_op type represents what can be done with an action, with respect to acting the action out. The action may be executed, halted, paused, resumed or postponed.

$$
Action \begin{cases}
name^{R} \leftarrow \{identifier\} \\
performer^{R} \leftarrow \{(Object)\} \\
target \leftarrow \{(Object)\} \\
prerequisite \leftarrow \{(Action),(Event),(State)\} \\
location \leftarrow \{(Location),(Distance),(Proximity)\} \\
intent \leftarrow \{(Action)\} \\
priority \leftarrow \{priority\} \\
method \leftarrow \{(Reference)\} \\
team \leftarrow \{modifier\_ownership\} \\
focus \leftarrow \{parameter\} \\
consequence \leftarrow \{(Consequence)\}
\end{cases}
$$

Figure A24: Abstract Concept – Action

The Action concept, illustrated in figure A24, encapsulates the basic notion of an action. The only required attributes are the name of the action and the performer. A target object may be added as the target of the action. The 'prerequisites' attribute states what must be true before the action may execute. The 'location' attribute may either describe where an action must take place, the methodology behind the action or a target location for the action, depending on the domain. The 'intent' attribute describes what action or objective is intended after the action is successful.

One important aspect is the 'method' attribute, by which the participant can assist in defining how an action is to be carried out. Typically, in TCL this is a reference to an object, tool or action.

$$\begin{array}{l} ActionSequence \\ Casts : Action \end{array} \left\{ \begin{array}{l} first \leftarrow \{(Action)\} \\ second \leftarrow \{(Action)\} \\ third \leftarrow \{(Action)\} \\ \ldots \\ consequence \leftarrow \{(Consequence)\} \end{array} \right.$$

Figure A25: Abstract Concept – ActionSequence

The ActionSequence concept describes an ordered set of actions that are carried out in a specific sequence. It is generally used to describe the actions that make up a procedure.

There are various to define how an action is executed, whether it is executed continually, triggered on an event and so forth. Typically, this is based the notion of a condition, which may either be a state or an event. These conditions allow for such actions such as 'whenever X do Y', 'if X do Y', 'while X do Y'.

Type: condition_reference ← {(Event), (State)}

The condition_reference type allows either an event or state concept.

Type: condition_absolute ← {'always', 'never'}

As with most concepts in a task-oriented domain, conditions are broken into sets. The always condition represents the set of all conditions, where the never condition represents the empty set, or under no condition.

Type: condition_custom ← {'whenever-necessary'}

The condition_custom type allows various specialty conditions, such as 'whenever-necessary', which states that the condition is true when it needs to be true.

Type: condition ← {condition_reference, condition_absolute, condition_custom}

$$\begin{array}{c} ContinualAction \\ Casts: Action \end{array} \left\{ \begin{array}{l} condition^{R} \leftarrow \{condition\} \\ action^{R} \leftarrow \{(Action)\} \\ consequence \leftarrow \{action: consequence\} \end{array} \right.$$

Figure A26: Abstract Concept – ContinualAction

$$\begin{array}{c} Procedure \\ Casts: Action \end{array} \left\{ \begin{array}{l} objective^{R} \leftarrow \{(Action),(Objective)\} \\ procedure^{R} \leftarrow \{(Action),(ActionSequence)\} \\ consequence \leftarrow \{procedure: consequence\} \end{array} \right.$$

Figure A27: Abstract Concept – Procedure

$$\begin{array}{c} Plan \\ Casts: Action \end{array} \left\{ \begin{array}{l} objective^{R} \leftarrow \{(Action),(Relation)\} \\ method \leftarrow \{(Action),(Reference)\} \\ duration \leftarrow \{????\} \\ \text{var}iation \leftarrow \{????\} \\ procedure \leftarrow \{(Action),(ActionSequence)\} \\ who \leftarrow \{????\} \\ consequence \leftarrow \{procedure: consequence\} \end{array} \right.$$

Figure A28: Abstract Concept – Plan

The ContinualAction concept, illustrated in figure A26, is one of the many action-execution concepts defined in TCL. This particular concept allows an action to fire whenever the condition attribute has been met. Other types of action-execution are described above; however, their specifications are left out due to length considerations.

The Procedure concept, illustrated in figure A27, represents a particular methodology and sequences of actions that must be carried out in order to perform some objective or higher-level action.

A plan is similar to a procedure; however, the plan concept is typically constructed in order to perform an action under a specific set of circumstances, while a procedure typically defines an action under normal conditions.

$$\begin{array}{l} RestrictionQuantity \\ Casts : Consequence \end{array} \begin{cases} content^{R} \leftarrow \{(Concept)\} \\ current \leftarrow \{numeric, custom\_numeric\} \\ required \leftarrow \{numeric, custom\_numeric\} \\ polarity^{R} \leftarrow \{numeric\_required\_relative\} \end{cases}$$

Figure A29: Abstract Concept – RestrictionQuantity

The RestrictionQuantity concept is one of the various concepts that represent a consequence. A typical consequence of an action is that resources are used in order to perform the action. A consequence of that action would then be reduced resources or even a shortage of resources. The RestrictionQuantity concept models that shortage, but it does need not to be limited to deficiency.

$$\begin{array}{l} ActionPrecedence \\ extends : Precedence \end{array} \begin{cases} first^{R} \leftarrow \{(Action)\} \\ second^{R} \leftarrow \{(Action)\} \end{cases}$$

Figure A30: Abstract Concept – ActionPrecedence

Priorities help to reduce the number of conflicting objectives and actions; however, in some circumstances it is not enough to represent the participant's

preferences. Precedence is one of the various concepts used to resolve those conflicts as they occur. The ActionPredence concept expresses the notion that one action is to take precedence over another.

Queries

Now that objectives and actions have been specified. This section will define some of the concepts that cross over toward the communicative aspects introduced into the task-oriented domain. One of the most fundamental aspects is the ability to ask questions about objects, methods, actions or states.

Questions are handled in a variety of ways as is described below under the query abstract operator. Some of the questions are handled in a structured form, phrasing the question in a particular way. These concepts represent those forms.

$$QueryParameter \; extends : Query \begin{cases} content^{R} \leftarrow \{(Concept)\} \\ parameter^{R} \leftarrow \{concept\_parameter\} \\ options \leftarrow \{(PossibleParameterValues)\} \\ suggestion \leftarrow \{(Concept)\} \end{cases}$$

Figure A31: Abstract Concept – QueryParameter

The QueryParameter concept is used to structure a question relating to the value of an attribute of a particular concept. In the task-oriented domain, the attribute of a concept is also referred to as a parameter. The MissingParameter concept is a child of QueryParameter that defines no new attributes. This is typically used by the agent when a certain parameter is required and must be answered before anything can be continued. The additional child concept is used to structure the phrases in such a way to denote that it must be answered rather than just denoting curiosity.

$$PossibleParameterValues \begin{cases} content \leftarrow \{(Concept)\} \\ parameter \leftarrow \{concept\_parameter\} \\ length^{R} \leftarrow \{integer\} \\ first \leftarrow \{value\} \\ second \leftarrow \{value\} \\ \ldots \end{cases}$$

Figure A32: Abstract Concept – PossibleParameterValues

The PossibleParameterValues concept is a means to enumerate the possible values of a concept's attribute. This is sometimes used in phrasing a QueryParameter query, such as "What do you want to mine, crystal or titanium?", however it can also be used by the dialogue model to match various incoming concepts to previously posed questions.

Changes in Concepts

Another important aspect in expressing concepts in communication is the ability to discuss change, whether it is changes in plans, objects or states. There are several different ways to represent a change. The top three are discussed below.

$$Change \begin{cases} original^{R} \leftarrow \{(Concept)\} \\ new^{R} \leftarrow \{(Concept)\} \end{cases}$$

Figure A33: Abstract Concept – Change

The Change concept refers to a change where the new concept is placed into the exact context of the original. This type of concept is generally used for interpretation correction or additive information.

$$Modification \begin{cases} original^{R} \leftarrow \{(Concept)\} \\ modified^{R} \leftarrow \{(Concept)\} \end{cases}$$

Figure A34: Abstract Concept – Modification

The Modification concept is another way to represent change. This concept refers to changes where the modified concept is the same or a compatible type as the original concept. This type of concept is generally used for discussing the changes in concepts currently under discussion, such as planning or negotiation.

$$Refinement \begin{cases} original^{R} \leftarrow \{(Concept)\} \\ refinement^{R} \leftarrow \{(Concept)\} \end{cases}$$

Figure A35: Abstract Concept – Refinement

The Refinement concept is a restriction on the Modification concept where the refinement concept must be a small deviation from the original. This type of concept is generally used for adding a parameter to a concept, such as adding a target or performer to an action.

Argumentation

Another important communicative aspect of the task-oriented domain is the ability to explain the reasoning behind an action or state. This is done through argumentation, the building of explanations, arguments and conclusions.

The Inference concept is a type of relation that represents a piece of reasonable knowledge, generally in the form ‘If X is true then Y must be true.’

$$\begin{matrix} Inference \\ Casts : Relation \end{matrix} \begin{cases} if^{R} \leftarrow \{(Relation)\} \\ then^{R} \leftarrow \{(Concept)\} \end{cases}$$

Figure A36: Abstract Concept – Inference

$$Argument \begin{cases} givenfact^{M} \leftarrow \{(Concept)\} \\ assumefact^{M} \leftarrow \{(Concept)\} \\ givenrule^{M} \leftarrow \{(Relation)\} \\ assumerule^{M} \leftarrow \{(Relation)\} \\ conclusion^{R} \leftarrow \{(Concept)\} \end{cases}$$

Figure A37: Abstract Concept – Argument

The Argument concept builds a semi-formal proof, using various facts and rules and formulating a conclusion. This concept builds expressions along the lines of 'Because X and Y, then Z'. The various attributes of the argument concept may have multiple values associated with them.

The 'givenfact' attribute quotes a state of the world. The 'assumefact' attribute postulates a state of the world. The 'givenrule' attribute quotes a relation, such as an inference, that represents knowledge of the world. The 'assumerule' attribute postulates knowledge of the world. The 'conclusion' attribute reaches a conclusion about the argument.

$$Explanation \begin{cases} content^{R} \leftarrow \{(Concept),(Operator)\} \\ explanation \leftarrow \{(Argument)\} \end{cases}$$

Figure A38: Abstract Concept – Explanation

The Explanation concept links an explanation, such as an Argument, to a concept, expressing the reasoning behind the concept. For example, if the 'content' attribute is an action that is performed, then the 'explanation' attribute represents why the action was performed.

$$ConclusionOf \quad \begin{cases} argument^{R} \leftarrow \{(Argument)\} \\ conclusion^{R} \leftarrow \{(Concept)\} \end{cases}$$

Figure A39: Abstract Concept – ConclusionOf

Another way to express the conclusion of an argument is to express it outside of the argument itself. This expresses the fact that the conclusion is added, meaning that the generator of the message is making, querying or proposing that conclusion. This is equivalent to using the Modification concept to express the original argument and the new argument with the conclusion internally attached.

Autonomy

Autonomy represents the ability of the agent to act under its own knowledge and experience. Autonomy is a very important aspect of multiple participants in a task-oriented domain. For instance, when should the agent ask the human participant for help, such as a decision or permission, or when should the agent not be a bother. Some of the core concepts of autonomy are discussed below.

One of the ways to represent autonomy is with permission. Permission represents what the agent is allowed, or not allowed, to do. The Permission concept states that the 'responsible' attribute has permission under the authority of the 'authority' attribute to produce the operator or perform the concept under the 'content' attribute.

$$Permission \begin{cases} content^{R} \leftarrow \{(Concept),(Operator)\} \\ responsible^{R} \leftarrow \{participant\} \\ authority^{R} \leftarrow \{participant\} \end{cases}$$

Figure A40: Abstract Concept –Permission

The ActionPermission extends the Permission concept including the required 'responsible' and 'authority' attributes. The 'content' attribute is further restricted to an action and the 'type' attribute indicates what may or may not be done with that action. This type of permission is also extended to describe what resources may be used during a specific action and so forth.

$$\begin{matrix} ActionPermission \\ extends: Permission \end{matrix} \begin{cases} content^{R} \leftarrow \{(Action)\} \\ type^{R} \leftarrow \{action\_op\} \end{cases}$$

Figure A41: Abstract Concept – ActionPermission

An autonomic shift represents a change in the level of autonomy of the agent. Autonomic shifts are generally represented through granting or denying permissions, but the AutonomicShift concept encapsulates that change in autonomy during a discussion.

$$AutonomicShift \begin{cases} content^{R} \leftarrow \{(Concept)\} \\ responsible^{R} \leftarrow \{participant\} \\ authority \leftarrow \{participant\} \end{cases}$$

Figure A42: Abstract Concept – AutonomicShift

Feedback

A relatively new property of human-agent communication in the task-oriented domain is the ability of a participant to provide active feedback to other participants. Because of the recent introduction of this property, this section will go into more detail than previous sections.

Type: feedback_absolute ← {‘positive’, ‘negative’, ‘neutral’}

Generally, feedback is positive, negative or neutral. However, as will be seen in the forthcoming types, feedback can come in a variety of scales.

Type: feedback_satisfaction ← {‘satisfied’, ‘unsatisfied’, ‘over-satisfied’}

Satisfactory feedback designates whether the participant is satisfied, unsatisfied or overly satisfied with any concept in the domain.

Type: feedback_confidence ← {‘sure’, ‘unsure’}

Confidence feedback designates the level of confidence the participant has with a particular concept. For example, if a plan is proposed, than the participant may provide feedback that expresses lack of confidence that the plan will be successful.

Type: feedback_agreement ← {‘agree’, ‘disagree’}

Agreement feedback designates the level at which the participant agrees with or disagrees with a particular concept. For example, if a conclusion is asserted, than the participant may provide feedback that expresses disagreement with the conclusion.

Type: feedback_relative ← {‘better’, ‘worse’, ‘indifferent’}

Along with various scales of feedback, relative feedback designates feedback towards a change. The change may not be implicit, but rather relative feedback could come from the performance of an action, for instance.

Type: feedback_emotion ← {‘pleased’, ‘upset’, ‘angry’, ‘confused’, ‘saddened’, ‘frightened’, ‘bored’, ‘worried’, ‘excited’, ‘eager’}

Another type of feedback that is introduced by the human element is that of emotional response. Emotional response is beyond the scope of this dissertation, as the focus is on practical task-oriented communication. However, it is mentioned here as future work may adapt to such response.

Type: feedback_amount ← {'too-much', 'too-little', 'just-right'}

Feedback may also be generated within a specific context, one of which may be about the methodology of an action, for instance. If one participant performs an action that creates resources, another participant may provide feedback that the first did not create enough.

Type: feedback ← {feedback_absolute, feedback_satisfaction, feedback_confidence, feedback_agreement, feedback_relative, feedback_emotion, feedback_amount, 'no-opinion'}

$$Problem \begin{cases} content^{R} \leftarrow \{(Concept),(Operator)\} \\ focus \leftarrow \{parameter\} \\ type \leftarrow \{feedback\} \end{cases}$$

Figure A43: Abstract Concept – Problem

Of the many ways feedback can be incorporated into the task-model, the two prevailing ones are mentioned here. One relates to positive feedback, while the other relates to negative feedback. In TCL, positive feedback is represented as an affirmation, while negative feedback is represented as a problem.

The Problem concept, illustrated in figure A43, represents a problem the participant has with a concept, operator or an attribute of a concept. The 'type' attribute specifies the type of feedback.

$$Solution \begin{cases} problem^{R} \leftarrow \{(Problem)\} \\ solution \leftarrow \{(Action),(Concept),(Operator)\} \end{cases}$$

Figure A44: Abstract Concept – Solution

If problems are introduced, so must be solutions. The Solution concept represents a simple solution to a problem. The solution is generally an action, but may include any concept or operator.

$$Affirmation \begin{cases} content^{R} \leftarrow \{(Concept),(Operator)\} \\ type \leftarrow \{ feedback\} \end{cases}$$

Figure A45: Abstract Concept – Affirmation

The Affirmation concept expresses positive feedback about a concept or operator. Affirmative feedback is generally used to build into the preferences of the user model of a participant as well as a reward for agents capable of reinforcement learning.

Interpretation

Another relatively new property of human-agent communication in the task-oriented domain is the ability to discuss interpretations, as well as pass interpretations of concepts to the agent. Again, because of the recent introduction of this property, this section will go into more detail that previous sections.

The first step toward introducing interpretation into the TCL model is adding references. References can take a variety of forms, however in the task-oriented experiments performed during the course of this dissertation; object and concept references were prevalent.

Type: reference_object ← {‘one’, ‘them’, ‘it’, ‘that’, ‘him’, ‘her’}

Type: reference_concept ← {‘one’, ‘it’, ‘that’}

Type: reference ← {reference_object, reference_concept}

An object reference is a reference to a particular object, such as an engineer, or a training camp. A concept reference is a reference to a previous concept such as an action, or an answer.

$$Reference \begin{cases} reference^{R} \leftarrow \{(Concept),(Set),reference\} \\ content \leftarrow \{(Concept)\} \end{cases}$$

Figure A46: Abstract Concept – Reference

The Reference concept attempts to enhance the reference type by adding a ‘content’ attribute that can join with the reference upon resolution. Such attachments make ‘one at each camp’ possible. In this statement, ‘one’ is the reference and ‘at each camp’ is the content, modeled as a modifier.

The attached listing from an example human session in appendix C uses implicit references, where ‘an engineer’ is treated as an identifier rather than a reference. This was done to simplify the output listing, but it is important to note that these are references and are resolved as such.

$$Meaning \begin{cases} content^{R} \leftarrow \{(Operator)\} \\ meaning \leftarrow \{(Concept)\} \end{cases}$$

Figure A47: Abstract Concept – Meaning

The next step towards introducing interpretation to the TCL model is by adding meaning. Meaning expresses the meaning of a particular previously uttered operator. This added information helps to resolve the ambiguity found within operators and builds into the user model. The Meaning concept is critical for reasoning about the added information in such statements as 'What I mean is…' as well as corrective dialogues.

$$\begin{array}{rl} Nomenclature & \left\{ usage^{R} \leftarrow \{text\} \right. \\ Casts : Knowledge & \left\{ meaning^{R} \leftarrow \{(Concept),(Reference)\} \right. \end{array}$$

Figure A48: Abstract Concept – Nomenclature

The Nomenclature concept encapsulates the connection between an expression and the meaning of that expression. Nomenclature is essential in building a user model and interpretation model for expanding the known linguistic capability of the interpretation mechanism. The agent does not have to know about the interpretation. However, it should at least know that there is some interpretation for purposes of discussion.

$$Interpretation \quad \left\{ content^{R} \leftarrow \{(Concept)\} \right.$$

Figure A49: Abstract Concept – Interpretation

The Interpretation concept is another way the agent can discuss the interpretation of a particular concept without knowing about the interpretation itself. The concept allows the agent to reference the content attribute's interpretation for purposes of discussion.

Interactive Operators

Now that various concepts have been defined and partially specified, this section attempts to specify the operators that act upon these concepts and what they mean. The operators of TCL are divided into two groups, the interactive operators and the agent operators. Interactive operators influence concepts toward communication and interaction, while agent operators represent actions taken internally by the agent.

Interactive operators of the practical communication language theory are defined in Chapter 3. They represent the operators that act upon the meaning-action concepts that are shared across the communication-behavior spectrum. These operators are defined in individual sections organized by their commonality.

Notation

The notation of operators is the same as the notation used for concepts.

Core Types

In order to construct a task operator, several core types must first be defined. These core types create the foundation of which other operators can be modeled.

Type: intent ← {execute, rate-numeric, record, query, learn, evaluate, adopt, plan, abandon, fix, procedural}

The intent type is essential when dealing with conversational modes, tracking and interpretation. The intent types shown in this appendix are the intents used in the example human session listing in appendix C. The intention refers to what the system is intending to do with the individual concept.

$$Operator \begin{cases} who^{R} \leftarrow \{participant\} \\ intent \leftarrow \{intent\} \end{cases}$$

Figure A50: Abstract Operator – Parent Operator

The core abstract operator is illustrated in figure A50. The 'who' attribute of the operator defines the owner, or producer, of the operator and the 'intent' attribute relates their assumed intent for producing the operator. As with the other operators in this section, other attributes have been removed due to length considerations.

The Conjunction and Disjunction operators start to form structure behind complex operators.

$$Conjunction \begin{cases} first^{R} \leftarrow \{(Operator)\} \\ second^{R} \leftarrow \{(Operator)\} \end{cases}$$

Figure A51: Abstract Operator– Conjunction

Conjunction is typically used to group multiple operators to the same message, when the message carries multiple meanings.

$$Disjunction \begin{cases} first^{R} \leftarrow \{(Operator)\} \\ second^{R} \leftarrow \{(Operator)\} \end{cases}$$

Figure A52: Abstract Operator– Disjunction

Disjunction is typically used to represent ambiguity in the interpretation of a message. Multiple meanings are attached to the same message and the meaning that makes the most sense is generally followed.

Both the conjunction and disjunction of operators execute within the rule engine up until the agent operator by using a ':result#' switch. This switch causes the agent operator to return rather than to fire. This allows ResolveConjunction and

ResolveDisjunction operators to be used to attempt to understand the variations in structure.

Simple Messages

The simplest and most often used message in task-oriented dialogue is acknowledge. The acknowledge operator has no content, other than what it extends from the parent operator.

$$Tell \quad \left\{ content^{R} \leftarrow \{(Concept)\} \right.$$

Figure A53: Abstract Operator – Tell

Another simple message is that of a Tell operator where a concept is told to the participants. Tell is generally the default operator if no other operator is detected.

$$Assert \quad \left\{ content^{R} \leftarrow \{(Feedback), feedback, (Meaning)\} \right.$$

Figure A54: Abstract Operator – Assert

The Assert operator is similar to the Tell operator except that assert is generally used for feedbacks and meanings.

$$Notify \quad \left\{ notification^{R} \leftarrow \{(Event)\} \right.$$

Figure A55: Abstract Operator – Notify

$$Warn \quad \{ content^R \leftarrow \{(Problem),(Consequence),(State),(Event),(Concept)\}$$

Figure A56: Abstract Operator – Warn

Two other simple messages are the notification of an event or the warning of a state, event or problem. These five operators are referred to as simple not only because they do not yield complex structure, but also because they are neither forward nor backward chaining. Although a tell operator can sometimes be interpreted as an answer, typically these operators do not directly link together with other operators.

Orders and Actions

Giving orders, formulating plans, executing or abandoning actions are all essential parts of task-oriented communication.

$$Order \quad \{ orders^R \leftarrow \{(Action)\}$$

Figure A57: Abstract Operator – Order

The Order operator allows the participant to order another participant to carry out an action, action sequence, plan or procedure.

$$Confirm \quad \{ confirmation^R \leftarrow \{(Concept),(Operator)\}$$

Figure A58: Abstract Operator – Confirm

The confirm operator allows the second participant to confirm the orders of the first. The confirm operator also allows all kinds of confirmations including the

confirmation of concepts such as modifications or desires. Operators may also be confirmed such as the confirmation that a notification request was received.

$$Plan \quad \begin{cases} content^{R} \leftarrow \{(Plan)\} \\ focus \leftarrow \{(Concept)\} \end{cases}$$

Figure A59: Abstract Operator – Plan

The Plan operator allows a participant to set the focus of the conversation towards the planning of the 'content' attribute with specific focus on the concept or parameter under the 'focus' attribute.

$$Execute \quad \{ content^{R} \leftarrow \{(Action)\}$$

Figure A60: Abstract Operator – Execute

The execute operator allows actions, action sequences, plans and procedures to be executed.

$$Abandon \quad \{ content^{R} \leftarrow \{(Action),(Concept)\}$$

Figure A61: Abstract Operator – Abandon

The abandon operator allows actions, action sequences, plans, procedures, desires, objectives and other concepts to be abandoned.

Questions

Questions and answers are performed through their respective operators as described below.

$$Query \quad \{ query^R \leftarrow \{(Concept)\}$$

Figure A62: Abstract Operator – Query

The query operator is how questions are postulated by a participant. The meaning behind the query operator depends entirely on the concept being operated upon. If the query is a QueryParameter or MissingParameter concept, then the query is a structured question focusing on a particular attribute of a concept. If the query is operating upon an incomplete PossibleParameterValues concept, then the participant is asking for the possible values of a parameter. If the PossibleParameterValues concept is filled, then the participant is asking if those are the possible parameter values. If the query is operating upon a relation, then the participant is asking if that relation is true; a conclusion, the participant is asking if it can be deduced. The query of an empty explanation means that the participant is asking for an explanation, generally 'Why' for a completely empty explanation.

If the query is operating upon a procedure then the participant is asking for the procedure if the procedure is empty as in "How do I?", or asking if the procedure is the correct one if it is not empty as in "Is this how I?". If the query is operating upon a Focus concept then the participant is inquiring about the current focus of the conversation.

Further queries and what they mean are omitted due to length considerations. However, for the sake of generality, when most incomplete concepts are queried, the

participant is asking for the missing pieces. On the other hand, when most complete concepts are queried, then the participant is asking if it is correct given the context.

Type: answer_polar ← {'affirmative', 'negative', 'ambiguous'}

The simplest answer is an affirmative or negative, yes or no, answer.

$$Answer \begin{cases} polarity^{R} \leftarrow \{answer\_polar\} \\ confidence \leftarrow \{percentage\} \end{cases}$$

Figure A63: Abstract Operator – Answer (yes/no)

The yes or no answer operator leverages polarity in order to state whether it is an affirmative or negative answer. A confidence rating also allows the participant to state their level of confidence in the answer. These types of answer generally result from a query of a completed concept.

$$Answer \begin{cases} content^{R} \leftarrow \{(Concept)\} \\ focus \leftarrow \{(Concept), parameter\} \end{cases}$$

Figure A64: Abstract Operator – Answer (with content)

The other type of answer generally results from the query of an incomplete concept. The answer operator in this case, returns the concept with the missing pieces filled in. It is up to the text-generator to know the differences between the postulated concept and resultant concept and leverage that difference in generation. However, the concept answer operator can also place focus on a particular concept or parameter within the content concept.

Suggestions and Negotiation

Another type of interaction in task-oriented dialogue is the proposal, rejection, agreement and commitment to shared tasks and beliefs. These types of operators allow for mutual planning, negotiation and persuasion.

$$Propose \begin{cases} proposal^{R} \leftarrow \{(Modification),(Action),(Concept)\} \\ justification \leftarrow \{(Concept)\} \end{cases}$$

Figure A65: Abstract Operator – Propose

The Proposal operator introduces a concept into the shared focus of the participants, often requesting their adoption of that concept. For example, an action or plan may be proposed for execution. A justification may be provided during the introduction in order to give evidence suggesting why it should be adopted. Children of the Proposal operator include Suggest and Offer.

$$\begin{matrix} CounterPropose \\ Extends: Propose \end{matrix} \begin{cases} original \leftarrow \{(Modification),(Action),(Concept)\} \\ proposal^{R} \leftarrow \{(Modification),(Action),(Concept)\} \\ justification \leftarrow \{(Concept)\} \end{cases}$$

Figure A66: Abstract Operator – CounterPropose

The CounterPropose operator rejects the concept of shared focus and introduces a new or modified concept. A justification may be provided which either justifies the rejection of the original concept, justifies the proposal of the new concept or both.

The accept operator accepts the given concept, usually under proposal.

$$Accept \quad \begin{cases} acceptance^{R} \leftarrow \{(Concept),(Operator)\} \\ justification \leftarrow \{(Concept)\} \end{cases}$$

Figure A67: Abstract Operator – Accept

$$Reject \quad \begin{cases} rejection^{R} \leftarrow \{(Concept),(Operator)\} \\ justification \leftarrow \{(Concept)\} \end{cases}$$

Figure A68: Abstract Operator – Reject

The reject operator rejects the given concept, which need not necessarily be a proposal. Typical justifications include restrictions when rejecting actions or plans; arguments when rejecting answers; or can even sometimes include desires, as in 'I don not want to'.

Requests

The final type of interaction discussed in this appendix pertains to requests. These types of operators allow for Permissions, Negotiation and shifting the ownership of shared resources.

$$Request \quad \begin{cases} request^{R} \leftarrow \{(Concept),(Operator)\} \\ justification \leftarrow \{(Concept)\} \end{cases}$$

Figure A69: Abstract Operator – Request

Similar to the query operator the request operator takes on different meanings based on the concept or operator being requested. The request of an action, rather than

the order, implies politeness. The request of permission may also imply politeness, or may also imply autonomic boundaries that need to be modified. A request for confirmation leads to reasoning about what is being confirmed.

The request for an operator generally implies that the requested operator be performed by the receiver of the request. For example, the request for the notify operator refers to the request that the receiver notify the producer of a future event.

$$Approve \quad \begin{cases} content^{R} \leftarrow \{(Concept),(Operator)\} \\ justification \leftarrow \{(Concept)\} \end{cases}$$

Figure A70: Abstract Operator – Approve

The approve operator approves requests. If the request is permission, then the participant grants the permission. If the request is a plan or methodology, then the participant approves of the plan or methodology. This does not necessarily lead to an order, but rather approves the plan for future use.

$$Deny \quad \begin{cases} content^{R} \leftarrow \{(Concept),(Operator)\} \\ justification \leftarrow \{(Concept)\} \end{cases}$$

Figure A71: Abstract Operator – Deny

The deny operator typically denies requests, although other concepts may also be denied.

Agent Operators

Unlike the previous concepts and operators introduced, agent operators pertain directly to internalizing concepts within the agent. The previous operators were generated to and from text. However, these new operators ask the agent to operate on the concepts and generate new operators as a response.

Although there are many evaluation-based operators, the evaluation is only a starting point to address the various concepts. For example, the agent may evaluate, agree, adopt and execute an action all in one agent operator. It is up to each individual agent implementation to decide.

$$EvaluateAction \begin{cases} action^{R} \leftarrow \{(Action),(Concept)\} \\ intent \leftarrow \{intent\} \end{cases}$$

Figure A72: Abstract Agent Operator – EvaluateAction

The EvaluateAction operator is responsible for evaluating, planning and executing actions. If the intent given to the operator is ‘execute’ then the agent is to evaluate the action with the intention of executing it. If the intent given to the operator is ‘adopt’ then the agent is to evaluate the action with the intent of adopting it into practice.

$$EvaluateQuery \begin{cases} query^{R} \leftarrow \{(Query),(Concept)\} \\ intent \leftarrow \{intent\} \end{cases}$$

Figure A73: Abstract Agent Operator – EvaluateQuery

The EvaluateQuery operator is responsible for responding to all queries. The agent uses its internal knowledge along with the queried concept to construct a response. Typically, the response is an answer, but may also be a query, for sub clarification.

$$GetPossibleParameterValues \begin{cases} content^R \leftarrow \{(Concept)\} \\ parameter \leftarrow \{parameter\} \end{cases}$$

Figure A74: Abstract Agent Operator – GetPossibleParameterValues

The GetPossibleParameterValues operator is a quick operator into getting the possible values for the concept parameter with respect to the domain and the agent's knowledge.

$$QueryResponseMatch \begin{cases} query^R \leftarrow \{(Query)\} \\ response^R \leftarrow \{(Concept)\} \end{cases}$$

Figure A75: Abstract Agent Operator – QueryResponseMatch

The QueryResponseMatch operator is a quick verification operator responsible for detecting if the response is a valid answer for the given query with respect to the domain and the agent's knowledge.

$$EvaluateProposal \begin{cases} proposal^R \leftarrow \{(Concept)\} \\ intent \leftarrow \{intent\} \end{cases}$$

Figure A76: Abstract Agent Operator – EvaluateProposal

The EvaluateProposal operator is responsible for responding to all proposals, including objectives, plans and procedures. The intent is generally either 'plan' or 'proposal' depending on the context.

$$EvaluateAcceptance \begin{cases} acceptance^{R} \leftarrow \{(Concept),(Operator)\} \\ justification \leftarrow \{(Concept)\} \\ intent \leftarrow \{intent\} \end{cases}$$

Figure A77: Abstract Agent Operator – EvaluateAcceptance

The EvaluateAcceptance operator is responsible for interpreting and recording when a participant accepts a given concept. The justification is used to help the agent build a user-model or also knowledge about why the concept was accepted.

$$EvaluateRejection \begin{cases} rejection^{R} \leftarrow \{(Concept),(Operator)\} \\ justification \leftarrow \{(Concept)\} \\ intent \leftarrow \{intent\} \end{cases}$$

Figure A78: Abstract Agent Operator – EvaluateRejection

The EvaluateRejection operator is similar to the EvaluateAcceptance operator in that it is responsible for interpreting and recording when a participant rejects a given concept. The justification can also be used to help the agent formulate a response, such as a counter-proposal or re-assertion of a modified concept.

The EvaluateProblem operator is responsible for evaluating a problem produced by another participant. If the intent of the problem is 'fix', then the agent should evaluate

whether or not it should, can, or will fix the problem. If the intent is 'evaluate' then the agent should evaluate whether or not the concept is a problem.

$$EvaluateProblem \quad \begin{cases} content^{R} \leftarrow \{(Problem),(Concept)\} \\ intent \leftarrow \{intent\} \end{cases}$$

Figure A79: Abstract Agent Operator – EvaluateProblem

The PermissionDenied operator records when permission is denied and shift autonomy accordingly. The PermissionGranted operator is identical in signature.

$$PermissionDenied \quad \{ permission^{R} \leftarrow \{(Action),(Concept)\}$$

Figure A80: Abstract Agent Operator – PermissionDenied

The RegisterNotification is a quick operator to tell the agent that it should notify another participant when an event occurs.

$$RegisterNotification \quad \{ notification^{R} \leftarrow \{(Event)\}$$

Figure A81: Abstract Agent Operator – RegisterNotification

The ApplyMeaning operator is used by the agent when the meaning of a concept is changed or meaning is added. This is most often used during reinterpretation or corrective dialogues.

$$ApplyMeaning \begin{cases} content^{R} \leftarrow \{(Concept),(Operator)\} \\ meaning^{R} \leftarrow \{(Concept)\} \\ intent \leftarrow \{intent\} \end{cases}$$

Figure A82: Abstract Agent Operator – ApplyMeaning

The EvaluateChangedConcept operator is responsible for evaluating changes, including modifications and refinements. The intent varies from 'execution' to 'planning'.

$$EvaluateChangedConcept \begin{cases} original^{R} \leftarrow \{(Concept)\} \\ new^{R} \leftarrow \{(Concept)\} \\ intent \leftarrow \{intent\} \end{cases}$$

Figure A83: Abstract Agent Operator – EvaluateChangedConcept

$$EvaluateKnowledge \begin{cases} knowledge^{R} \leftarrow \{(Knowledge)\} \\ intent \leftarrow \{intent\} \end{cases}$$

Figure A84: Abstract Agent Operator – EvaluateKnowledge

$$EvaluateAutonomicShift \begin{cases} content^{R} \leftarrow \{(Concept)\} \\ responsible^{R} \leftarrow \{participant\} \\ intent \leftarrow \{intent\} \end{cases}$$

Figure A85: Abstract Agent Operator – EvaluateAutonomicShift

The EvaluateKnowledge concept is responsible for helping the agent to learn, as in the 'learn' intent, or otherwise updating the knowledge of the agent.

The EvaluateAutonomicShift operator is responsible for evaluating autonomic shifts that occur other than when permission is granted or denied.

Helper Functions

Helper functions are inline functions that operate on the shared concept graph or on the structure of shared concepts themselves. They do not produce any rules, nor change the state of the dialogue-reasoning engine. They take the same notation as earlier operators.

$$\begin{array}{l} GetIntent \\ return : intent \end{array} \quad \left\{ content^{R} \leftarrow \{(Concept)\} \right.$$

Figure A86: Helper Function – GetIntent

GetIntent returns the intent of the action by means of the shared concept graph. If the current concept does not have intent, then it traces the concept back to an operator that provides the intent. This is function is used to pass the intent from the operator to the evaluate function.

$$\begin{array}{l} GetGenerator \\ return : (Concept) \end{array} \quad \left\{ content^{R} \leftarrow \{(Concept)\} \right.$$

Figure A87: Helper Function – GetGenerator

GetGenerator uses the shared concept graph to get the generating concept. If the concept was the direct result of an agent operator, then the generator is what was passed to the agent.

$$\begin{array}{c} IsRefinement \\ return : bool \end{array} \left\{ \begin{array}{l} original^{R} \leftarrow \{(Concept)\} \\ refined^{R} \leftarrow \{(Concept)\} \end{array} \right.$$

Figure A88: Helper Function – IsRefinement

IsRefinement analyses the structure of both the original concept and the refined concept. If the 'original' concept contains everything that the 'refined' concept contains, and the added information in the 'refined' concept does not collision with the typing of the 'original' concept, then the function returns true. Otherwise, it returns false.

$$\begin{array}{c} ParameterMatchInFocus \\ return : (Concept) \end{array} \left\{ \begin{array}{l} content^{R} \leftarrow \{(Concept)\} \\ parameter^{R} \leftarrow \{parameter\} \end{array} \right.$$

Figure A89: Helper Function – ParameterMatchInFocus

$$\begin{array}{c} AddParameter \\ return : (Concept) \end{array} \left\{ \begin{array}{l} original^{R} \leftarrow \{(Concept)\} \\ parameter^{R} \leftarrow \{parameter\} \\ value^{R} \leftarrow \{(Concept)\} \end{array} \right.$$

Figure A90: Helper Function – AddParameter

ParameterMatchInFocus looks at recent shared concepts and finds one that is the same type as the parameter in question and analyzes if it will fit into the concept in 'content'. If it succeeds, then the concept is returned. Otherwise, false is returned.

AddParameter creates a duplicate 'original' concept, places the 'value' in the parameter, with respect to the new concept, then returns the new one.

$$\begin{array}{c} ReplaceParameter \\ return:(Concept) \end{array} \left\{ \begin{array}{l} original^{R} \leftarrow \{(Concept)\} \\ parameter^{R} \leftarrow \{parameter\} \\ value^{R} \leftarrow \{(Concept)\} \end{array} \right.$$

Figure A91: Helper Function – ReplaceParameter

ReplaceParameter is similar to AddParameter except that the parameter is replaced rather than added.

$$\begin{array}{c} Collision \\ return:bool \end{array} \left\{ \begin{array}{l} first^{R} \leftarrow \{(Concept)\} \\ second^{R} \leftarrow \{(Concept)\} \end{array} \right.$$

Figure A92: Helper Function – Collision

Collision analyzes the structure of both concepts. If the information in one concept that is not in the other concept fits within the other concepts typing and the same with the other concept, then the function returns true. Otherwise, false is returned.

Collision is used to detect whether two concepts can be merged into a single concept meaningfully, or if there are any conflicts between the structures of the concepts.

It is used primarily in detected if one concept is an extension of another concept, a common property found within dialogue-based representations.

$$\begin{array}{c} Merge \\ return:(Concept) \end{array} \quad \begin{cases} first^{R} \leftarrow \{(Concept)\} \\ second^{R} \leftarrow \{(Concept)\} \end{cases}$$

Figure A93: Helper Function – Merge

Merge is similar to collision. However, merge combines the structure of both concepts into a new concept and returns it.

$$\begin{array}{c} GetRootConcept \\ return:(Concept) \end{array} \quad \{ content^{R} \leftarrow \{(Concept)\}$$

Figure A94: Helper Function – GetRootConcept

GetRootConcept analyzes the structure of the concept and returns the most recent root of the concept. A concept may be composed within another concept, GetRootConcept returns the top most parent that is not composed within another concept.

$$\begin{array}{c} FindParameter \\ return: parameter \end{array} \quad \begin{cases} root^{R} \leftarrow \{(Concept)\} \\ search^{R} \leftarrow \{(Concept)\} \end{cases}$$

Figure A95: Helper Function – FindParameter

FindParameter finds the ‘search’ concept within the ‘root’ concept through analyzing its compositional structure and returns a parameter to the compositional link with respect to the root concept. FindParameter is used in the replacement of concepts.

$$\begin{array}{c} ContainsConcept \\ return : bool \end{array} \left\{ \begin{array}{l} content^{R} \leftarrow \{(Concept)\} \\ contains^{R} \leftarrow \{(Concept)\} \end{array} \right.$$

Figure A96: Helper Function – ContainsConcept

ContainsConcept detects whether the ‘contains’ concept is contained through structural composition within the ‘content’ concept and returns true if it is contained, false if it is not contained.

$$\begin{array}{c} GetNextInSet \\ return : parameter \end{array} \left\{ \begin{array}{l} set^{R} \leftarrow \{(Set)\} \\ focus^{R} \leftarrow \{parameter\} \end{array} \right.$$

Figure A97: Helper Function – GetNextInSet

GetNextInSet is a specialty function that relies on knowledge of Set concepts. It takes the concept under current focus and finds it within the set concept. It then figures out the next concept and returns a parameter to it. If it is not found within the set or it is the last one in the set then it returns false.

Macro Functions

Macro functions are similar to helper operators in that they never reach either participant or interact with any outside concept or operator. Macro functions are often overloaded and produce new interaction operators. Macro operators are responsible for

detecting various operator signatures and causing change in resultant operators based on that signature.

$$ResolveConjunction \begin{cases} first^R \leftarrow \{(Operator)\} \\ second^R \leftarrow \{(Operator)\} \end{cases}$$

Figure A98: Macro Function – ResolveConjunction

The ResolveConjunction and ResolveDisjunction operators detect specialties within the concepts. For example, in conjunction: 'no, do this instead' (Deny, Alternative) implies a form of counter-order; 'No, not unless' or 'No, unless' or (Deny, Conditional) implies a conditional denial; 'No, because' or (Deny, Justification) implies a denial with justification; 'No, <state of the world>' (Deny, State) may imply deny with a possible reason. And so forth.

$$ResolveDisjunction \begin{cases} first^R \leftarrow \{(Operator)\} \\ second^R \leftarrow \{(Operator)\} \end{cases}$$

Figure A99: Macro Function – ResolveDisjunction

ResolveDisjunction is responsible for handling the detection, evaluation and resolution of ambiguity with the assistance of various agent operators.

RewriteQuery generates a resultant concept when a queried concept is merged with its response concept. This function is used when collision and merge fail.

$$\begin{matrix} RewriteQuery \\ return:(Concept) \end{matrix} \begin{cases} query^{R} \leftarrow \{(Query)\} \\ response^{R} \leftarrow \{(Concept)\} \end{cases}$$

Figure A100: Macro Function – RewriteQuery

The result of a various responses such as a query-response is typically placed into a HandleResponse operator that then directs it to the appropriate operator such as EvaluateAction if the result is an action or EvaluateQuery if the result is a query.

$$HandleResponse \begin{cases} content^{R} \leftarrow \{(Concept)\} \\ intent^{R} \leftarrow \{intent\} \end{cases}$$

Figure A101: Macro Function – HandleResponse

$$HandleChange \begin{cases} original^{R} \leftarrow \{(Concept)\} \\ new^{R} \leftarrow \{(Concept)\} \end{cases}$$

Figure A102: Macro Function – HandleChange

The HandleChange macro is responsible for handling when a concept changes due to a Change, Modification or Refinement.

$$HandleFocus \begin{cases} content^{R} \leftarrow \{(Concept)\} \\ intent^{R} \leftarrow \{intent\} \end{cases}$$

Figure A103: Macro Function – HandleFocus

HandleFocus is responsible for finding the focus of the conversation and restarting it. This is performed through a query of focus or sometimes after inactivity, depending on the agent implementation.

$$HandleFeedback \quad \begin{cases} content^{R} \leftarrow \{(Concept)\} \\ feedback^{R} \leftarrow \{(Feedback), feedback\} \end{cases}$$

Figure A104: Macro Function – HandleFeedback

HandleFeedback matches the 'feedback' to the concept being referenced and attempts to build a new concept accordingly. For example, a negative feedback on the statement of a rate of change builds a problem that the rate of change is too fast or slow.

$$HandleReinterpretation \quad \begin{cases} content^{R} \leftarrow \{(Concept)\} \\ intent^{R} \leftarrow \{intent\} \end{cases}$$

Figure A105: Macro Function – HandleReinterpretation

HandleReinterpretation is used after a corrective dialogue has taken place to change the previous meaning of the 'content' concept. The new concept and its original intent should then be re-postulated so that a valid response can take place. This is described in more detail in chapter 5, which discusses the various conversational capabilities including corrective dialogue.

LookForward is an unusual macro in that it is a holder, which is recorded as an obligation. The 'key' is used to determine how to match the obligation with future

operators and concepts. The LookForward macro helps to resolve chaining concepts such as, “First…Then…Finally…”

$$LookForward \quad \begin{cases} concept^{R} \leftarrow \{(Concept)\} \\ key^{R} \leftarrow \{forward\_key\} \end{cases}$$

Figure A106: Macro Function – LookForward

$$Advance \quad \{content^{R} \leftarrow \{(Concept),(Operator)\}$$

Figure A107: Macro Function – Advance

The advance macro is used in determining how to advance different concepts. For example, in set based concepts the focus is incremented to the next position. In other concepts, advancing to the next focal point varies.

$$\begin{array}{l} MergeConcepts \\ return:(Concept) \end{array} \quad \begin{cases} first^{R} \leftarrow \{(Concept)\} \\ second^{R} \leftarrow \{(Concept)\} \end{cases}$$

Figure A108: Macro Function – MergeConcepts

MergeConcepts is an inline function that is used when collision and merge fail. For example, when two actions must be merged, the MergeConcepts function knows to build an ActionSequence for them. Similarly, MergeConcepts can merge an ActionSequence with an action by appending the ActionSequence. MergeConcepts can also work through composition such as Plans and Procedures.

$$if \quad \begin{cases} eval^R \leftarrow \{(Evaluation)\} \\ then^R \leftarrow \{(Concept),(Operator)\} \\ else^R \leftarrow \{(Concept),(Operator)\} \end{cases}$$

Figure A109: Macro Function – if

$$\begin{matrix} > \\ return:bool \end{matrix} \quad \begin{cases} first^R \leftarrow \{(Concept)\} \\ second^R \leftarrow \{(Concept)\} \end{cases}$$

Figure A110: Macro Function – greater-than

$$\begin{matrix} = \\ return:bool \end{matrix} \quad \begin{cases} first^R \leftarrow \{(Concept)\} \\ second^R \leftarrow \{(Concept)\} \end{cases}$$

Figure A111: Macro Function – equality

The if macro function is one of the ways in which functionality can be introduced into the rules of the dialogue-reasoning engine. The 'eval' attribute is examined, if the eval attribute is true then the 'then' attribute is returned, otherwise the 'else' attribute is returned. Other such macros are as straightforward.

Greater-than is an example of an eval attribute used within an if macro. The greater-than macro requires that the concepts be quantifiable or comparable in some fashion.

The equality macro is similar to greater-than except that the concepts can also evaluate if they have the same values in identical structure. Note, this does not mean similar values, as the macro does not have the reasoning to translate between varying forms of a concept.

APPENDIX B
IMPLEMENTATION DETAILS

This appendix covers the implementation details of the intelligent agent, the Stratagus environment and the dialogue-modeling engine as well as their integration. The appendix is broken into sections with each component receiving its on section.

Intelligent Agent Architecture

The intelligent agent used in this dissertation research was created from scratch in C++ using various classes to represent atomic actions, objectives and other concrete concepts used in TCL. Because the details of this dissertation are in the dialogue capabilities and not in the agent implementation, the agent implementation itself was rather simple. Nevertheless, this simplicity demonstrates that the powerful expressiveness the agent is capable of is not because of the agent implementation, but rather the interaction engine to which the agent is connected.

Rather than using an intention recognition or shared-plan based approach, the agent's behavior is determined by the agent's current goals, being reactive in nature. The agent carried out simple orders by modeling the orders as atomic actions then carrying out those actions. Some of the reasoning behind the agent, such as the argumentation provided during extended explanations was done through pre-prepared domain knowledge provided to the agent.

Stratagus Environment

Stratagus is an open-source real-time strategy game engine hosted on SourceForge. The Battle of Survival data set was used. Stratagus was chosen for its dynamic real-time online management of complex resources and situations. The modifications made to Stratagus were minimal, described in the system section below. It is the intention of the author to work with the Stratagus team in incorporating desired features back into the main Stratagus source code branch and made publicly available.

The Dialogue Reasoning Engine

An ad-hoc reasoning engine based on the syntax of CLIPS and JESS was created for the dialogue reasoning engines. CLIPS could not be used due to its lack of nested templates. Concepts, Operators and Rules are all represented in C++ as simple classes; agent operators were linked to function-calls within the agent implementation. There was no optimization or reduction in the reasoning engine.

System Integration

The galaxy communicator [29] was chosen for the system integration because of its flexibility, portability, logging, testing and debugging capabilities. It is a spoke and hub architecture where messages are passed to a central communicator and various components can subscribe to streams of messages. Figure B1 shows the major components and their corresponding message streams.

The Stratagus environment was modified to allow text to be entered from and displayed to a human participant. This was done through slight changes to the games multiplayer chat feature. An AI Automation stream was created which allows the agent to query various objects and properties in the game as well as perform the same actions a player would perform when playing. This was done through manipulating the game's network play feature. A UI Manipulation stream was set up that allowed the control of the screen as well as the detection of selection from the mouse.

The intelligent agent is able to access the UI Manipulation stream as well as the AI Automation stream. A TCL stream allows the agent to send and receive TCL messages as a participant in the conversation. The TCL reasoning engine is embedded into the intelligent agent. The messages sent from the agent correspond to messages sent from the TCL reasoning system intended for the human participant.

The simple parser does not perform natural language understanding but rather triggers specific pre-defined output messages based on the input text. Similarly, the

generation module does not perform actual message generation but rather triggers a specific pre-defined output message based on the output TCL. This is because it was beyond the scope of this dissertation to handle speech recognition and generation, but rather to demonstrate the capabilities of TCL, which is free of natural language.

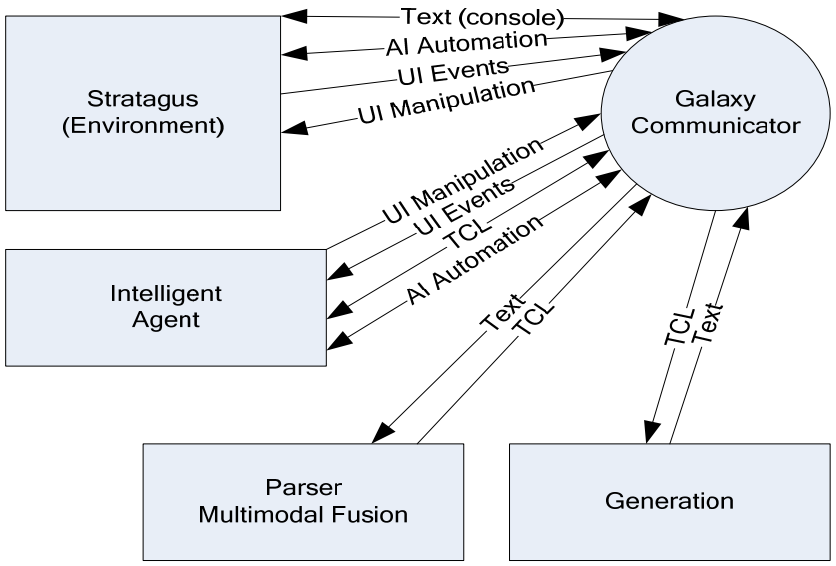

Figure B1: The System Integration

APPENDIX C
EXAMPLE HUMAN SESSION

This appendix contains the output of an example human session of the system described in appendix B. The lines in the listing are numbered as shown. The letter following the line number corresponds to the type of construct that is on the line. Table C1 provides the interpretation of these letters.

Table C1: Session Listing - Line key

	Human	The human entered the text into Stratagus.
H		
S	Shared Concept	A concept was added to the shared concept graph.
M	Message	This is the message (operator) the system is processing.
R	Rule	A rule fired within the system.
#	Agent Execution	Result of an agent operator, including its input signature.
A	Agent	The agent sent a message to the human.
D	Display	Something that changed with the Stratagus display.

The shared concept line includes a number corresponding to the identifier provided for that shared concept. It is important to note that there are more shared concepts that revealed in the listing, but many were removed for brevity. Other such modifications made for brevity include the removal of reference concepts down to identifiers, such as a reference to ‘engineer’.

The definition and descriptions for all of the concepts and operators listed here can be found in appendix A. Additionally, discussions of the conversational capabilities demonstrated within this trial can be found in chapter 5.

Table C2: Example Human Session Listing

```
[001-H]: "Send 7 engineers to mine."
[002-S]: [00001] = (Action
            <name mine>
            <performer (Quantity <value 7> <content engineer>)> )
[003-M]: (@Order <orders (00001)>)
[004-R]: (@Order <orders Action:?A>) => (#EvaluateAction <action ?A> <intent execute>)
[005-#]: (#EvaluateAction <action ?A> <intent execute>)
         => {  ($Reject <rejection ?A> <justification ?B>),
               ($Propose
                  <proposal (Modification
                     <original ?A> <modified ?C>)> ) }
[006-S]: [00002] = (RestrictionQuantity
            <content engineer>
            <current 2>
            <required 7>
            <polarity insufficient> )
[007-M]: ($Reject <rejection (00001)> <justification (00002)>)
[008-A]: >> “I can't, there are only two engineers."
[009-S]: [00003] = (ActionSequence
            <first (Action
               <name train>
               <target (Quantity <value 5> <content engineer>)> )>
            <second (00001)> )
[010-S]: [00004] = (Modification <original (00001)> <modified (00003)> )
[011-M]: ++($Propose <proposal (00004)>)++
[012-A]: >> “Should I train more?"
[013-H]: "Yes"
[014-M]: (@Answer <polarity affirmative>)
[015-R]: {  (@Answer <polarity affirmative>),
            ??($Propose <proposal ?A>)?? }
         => {  (@Accept <acceptance ($Propose <proposal ?A>)>),
               --($Propose <proposal ?A>)-- }
[016-M]: (@Accept <acceptance ($Propose <proposal (00004)>)>)
[017-R]: (@Accept <acceptance ($Propose
            <proposal (Modification <original ?A> <modified ?B> )> )>)
         => {  (%GetIntent <content ?A>) -> ?C,
               (#EvaluateAction <action ?B> <intent ?C>) }
[018-#]: (#EvaluateAction <action ?A> <intent execute>) => {  ($Query <query ?C>) }
[019-S]: [00005] = (MissingParameter <content (00001)> <parameter Action:target> )
[020-M]: ++($Query <query (00005)>)++
[021-A]: >> “What should they mine?"
[022-H]: "What are my options?"
[023-M]: (@Query <query (PossibleParameterValues)>)
[024-R]: {  (@Query <query (PossibleParameterValues)>),
            [$Query <query (QueryParameter <content ?A> <parameter ?B>)>] }
         => (#GetPossibleParameterValues <content ?A> <parameter ?B>)
[025-#]: (#GetPossibleParameterValues <content ?A> <parameter ?B>)
         => ($Answer <content (PossibleParameterValues:?C)>)
[026-S]: [00006] = (PossibleParameterValues
            <length 2>
            <first crystal>
            <second titanium>)
[027-M]: ($Answer <content (00006)>)
[028-A]: >> “Crystal or titanium."
[029-H]: "Crystal"
[030-M]: (@Tell <content Crystal>)
[031-R]: {  (@Tell <content ?A>),
            ??($Query <query ?B>)??,
            (#QueryResponseMatch <query ?B> <response ?A>) }
         => {  (^RewriteQuery <query ?B> <response ?A>) -> ?C,
               (%GetIntent <content ?C>) -> ?D,
               (^HandleResponse <content ?C> <intent ?D>),
               --($Query <query ?B>)-- }
[032-R]: (^RewriteQuery
            <query (QueryParameter <content ?A> <parameter ?B> )>
            <response ?C> )
         -> (%AddParameter <original ?A> <parameter ?B> <value ?C>)
```

Table C2 Continued

```
[033-S]: [00007] = (Action
            <name mine>
            <performer (Quantity <value 7> <content engineer>)>
            <target crystal> )
[034-R]: (^HandleResponse <content (Action:?A)> <intent ?B>)
         => (#EvaluateAction <action ?A> <intent ?B>)
[035-#]: (#EvaluateAction <action ?A> <intent execute>) => ($Confirm <confirmation ?A>)
[036-M]: ($Confirm <confirmation (00007)>)
[037-A]: >> “Ok"
[038-H]: "Have 7 more engineers mine titanium."
[039-S]: [00008] = (Action
            <name mine>
            <performer (Quantity
               <value 7>
               <content (Modifier <modifier more> <content engineer>)> )>
            <target titanium> )>)
[040-M]: (@Order <orders (00008)>)
[041-R]: (@Order <orders Action:?A>) => (#EvaluateAction <action ?A> <intent execute>)
[042-#]: (#EvaluateAction <action ?A> <intent execute>) => ($Confirm <confirmation ?A>)
[043-M]: ($Confirm <confirmation (00008)>)
[044-A]: >> “Ok"
[045-H]: "After that have another engineer build 2 generators."
[046-S]: [00009] = (Action
            <prerequisite that>
            <performer (Modifier <modifier another> <content engineer>)>
            <name build>
            <target (Quantity <value 2> <content generator>)> )
[047-M]: (@Order <orders (00009)>)
[048-R]: (@Order <orders Action:?A>) => (#EvaluateAction <action ?A> <intent execute>)
[049-#]: (#EvaluateAction <action ?A> <intent execute>) => ($Query <query ?B>)
[050-S]: [00010] = (Action
            <name build>
            <performer (Modifier <modifier another> <content engineer>)>
            <target (Quantity <value 2> <content generator>)> )>)
            <prerequisite (00004)> )
[051-S]: [00011] = (MissingParameter <content (00010)> <parameter Action:location> )>)
[052-M]: ++($Query <query (00011)>)++
[053-A]: >> “Where?"
[054-H]: "To the west, near the other generators."
[055-S]: [00012] = (DirectionLocation <reference current-location> <direction west>)
[056-S]: [00013] = (Proximity
            <reference (Modifier <modifier other> <content generator> )>
            <distance near> )
[057-S]: [00014] = (Conjunction <first (00012)> <second (00013)>)
[058-M]: (@Tell <content (00014)>)
[059-R]: {  (@Tell <content ?A>),
            ??($Query <query ?B>)??,
            (#QueryResponseMatch <query ?B> <response ?A>) }
         => {  (^RewriteQuery <query ?B> <response ?A>) -> ?C,
               (%GetIntent <content ?C>) -> ?D,
               (^HandleResponse <content ?C> <intent ?D>,
               --($Query <query ?B>)-- }
[060-R]: (^RewriteQuery
            <query (QueryParameter <content ?A> <parameter ?B> )>
            <response ?C> )
         -> (%AddParameter <original ?A> <parameter ?B> <value ?C>)
[061-S]: [00015] = (Action
            <name build>
            <performer (Modifier <modifier another> <content engineer>)>
            <target (Quantity <value 2> <content generator>)>
            <prerequisite (00004)>
            <location (00014)> )
[062-R]: (^HandleResponse <Action:?A> <intent ?B>)
         => (#EvaluateAction <action ?A> <intent ?B>)
[063-#]: (#EvaluateAction <action ?A> <intent execute>) => ($Confirm <confirmation ?A>)
[064-M]: ($Confirm <confirmation (00015)>)
[065-A]: >> “Ok"
```

Table C2 Continued

```
[066-H]: "Send an engineer to the northeast to build a training camp."
[067-S]: [00016] = (Action
            <name build>
            <target ‘training camp’> )
[068-S]: [00017] = (Action
            <name move>
            <performer engineer>
            <location ‘the northeast’>
            <intent (00016)> )
[069-M]: (@Order <orders (00017)>)
[070-R]: (@Order <orders Action:?A>) => (#EvaluateAction <action ?A> <intent execute>)
[071-#]: (#EvaluateAction <action ?A> <intent execute>) => ($Request <request ?B>)
[072-S]: [00018] = (ActionPermission
            <content (Action <name mine> <performer engineer01> <target crystal>)>
            <type halt> )
[073-M]: ++($Request <request (00018)>)
[074-A]: >> “May I take one away from mining?"
[075-H]: "No, use a new one, as soon as possible."
[076-S]: [00019] = (Action
            <name use>
            <target (Modifier <modifier new> <content one>)>
            <priority asap> )
[077-S]: [00020] = (Action
            <method (Modifier <modifier new> <content one>)>
            <priority asap> )
[078-M]: (Conjunction
            <first (@Answer <polarity negative>)>
            <second (Disjunction
               <first (@Order <orders (00019)>)>
               <second (@Order <orders (00020)>)> )>)
[079-R]: (Conjunction <first ?A> <second ?B>)
         => {  (?A):resolve#->?C,
               (?B):resolve#->?D,
               (^ResolveConjunction <first ?C> <second ?D>) }
[080-M]: (@Answer <polarity negative>)
[081-R]: {  (@Answer <polarity negative>),
            ??($Request <request ?A>)?? }
         => {  (@Deny <content ?A>),
               --($Request <request ?A>)-- }
[082-R]: (@Deny <content (Permission ?A)>)
         => (#PermissionDenied <permission (Permission ?A)>)
[083-R]: (Disjunction <first ?A> <second ?B>)
         => {  (?A):resolve#->?C,
               (?B):resolve#->?D,
               (^ResolveDisjunction <first ?C> <second <?D>) }
[084-M]: (@Order <orders (00019)>)
[085-R]: (@Order <orders Action:?A>)
         => (#EvaluateAction <action ?A> <intent execute>)
[086-M]: (@Order <orders (00020)>)
[087-R]: {  (@Order <orders Action:?A>,
            [Action:?B],
            (!(%Collision <first ?A> <second ?B>)) }
         => {  (%Merge <first ?A> <second ?B>) -> ?C,
               (@Order <orders Action:?C) }
[088-S]: [00021] = (Action
            <name move>
            <performer engineer>
            <location ‘the northeast’>
            <intent (00017)>
            <method (Modifier <modifier new> <content one>)>
            <priority asap> )
[089-S]: (@Order <orders (00021)>)
[090-R]: (@Order <orders Action:?A>)
         => (#EvaluateAction <action ?A> <intent execute>)
[091-M]: (^ResolveDisjunction
            <first (#EvaluateAction <action (00019)> <intent execute>)>
            <second (#EvalauteAction <action (00021)> <intent execute)> )
```

Table C2 Continued

```
[092-R]: (^ResolveDisjunction
            <first (#EvaluateAction <action ?A> <intent execute>)>
            <second (#EvaluateAction <action ?B> <intent execute>)> )
         => {  (#EvaluateAction <action ?A> <intent rate-numeric>) -> ?C,
               (#EvaluateAction <action ?B> <intent rate-numeric>) -> ?D,
               (if
                  <eval (> <first ?C> <second ?D>)>
                  <then (#EvaluateAction <action ?A> <intent execute>)>
                  <else (#EvaluateAction <action ?B> <intent execute>)> ) }
[093-M]: (^ResolveConjunction
            <first (#PermissionDenied <permission (00018)>)>
            <second (#EvaluateAction <action (00021)> <intent execute>)> )
[094-R]: {  (^ResolveConjunction
               <first (#PermissionDenied <permission ?A>)>
               <second (#EvaluateAction <action ?B> <intent execute> )>),
            (%GetIntent <content ?A>) -> ?C,
            (= <first ?C> <second execute>),
            (%GetGenerator <content ?A>) -> ?D,
            (%IsRefinement <original ?D> <refined ?B>) }
         => {  (#PermissionDenied <permission ?A> <intent record>),
               (#EvaluateAction <action ?D> <intent execute>) }
[095-#]: (#PermissionDenied <permission ?A> <intent record>) => {}
[096-#]: (#EvaluateAction <action ?A> <intent execute>)=> ($Confirm <confirmation ?B>)
[097-S]: [00022] = (Action
            <name train>
            <target engineer>
            <intent (Action
               <name build>
               <target 'training camp'>
               <location 'the northeast'> )>
            <priority asap> )>
[098-M]: ($Confirm
            <confirmation (ActionPrecedence
               <first (Action (00022))>
               <last (Action
                  <name train>
                  <target engineer>
                  <intent any> )> )>)
[099-A]: >> "Ok, I will use the next one trained."
[100-H]: "Also send one to the south to build a training camp."
[101-S]: [00023] = (Action
            <name build>
            <target 'training camp'> )
[102-S]: [00024] = (Action
            <name move>
            <content engineer>
            <location 'the south'>
            <intent (Action <name build> <target 'training camp'>)> )
[103-M]: (@Order <orders (Modifier <modifier also> <content (00024)>)>)
[104-R]: (@Order <orders (Modifier <modifier also> <content (Action:?A)>)>)
         => (@Order <orders (Action:?A)>)
[105-R]: (@Order <orders Action:?A>) => (#EvaluateAction <action ?A> <intent execute>)
[106-#]: (#EvaluateAction <action ?A> <intent execute>)
         => {  ($Request <request ?B>)
[107-S]: [00025] = (Refinement
            <original (Action 00023)>
            <refinment (Action
               <method (Modifier <modifier new> <content one>)>
               <priority asap> )>)
[108-M]: ++($Request <request (Confirm <confirmation (00025)>)>)++
[109-A]: >> "A new one, as soon as possible?"
[110-H]: "Yes"
[111-M]: (@Answer <polarity affirmative>)
[112-R]: {  (@Answer <polarity affirmative>),
            ??($Request <request ?A>)?? }
         => {  (@Accept <acceptance ?A>),
               --($Request <request ?A>)-- }
```

Table C2 Continued

```
[113-R]: (@Accept <acceptance (Confirm <confirmation ?A>)>)
         => (@Confirm <confirmation ?A>)
[114-R]: {  (@Confirm <confirmation (Refinement <original ?A> <refinement ?B>)>),
            (%GetIntent <content ?A>) -> ?C }
         => {  (%Merge <first ?A> <second ?B>) -> ?D,
               (#EvaluateAction <action ?D> <intent ?C>) }
[115-S]: [00026] = (Action
            <name move>
            <content engineer>
            <location 'the south'>
            <intent (00023)>
            <method (Modifier <modifier new> <content one>)>
            <priority asap> )
[116-#]: (#EvaluateAction <action ?A> <intent execute>) => ($Confirm <confirmation ?B>)
[117-S]: [00027] = (Action
            <name train>
            <target engineer>
            <intent (Action
               <name build>
               <target 'training camp'>
               <location 'the south'> )>
            <priority asap> )
[118-M]: ($Confirm
            <confirmation (ActionPrecedence
               <first (Action (00027)>
               <last (Action
                  <name train>
                  <target engineer>
                  <intent any> )> )>)
[119-A]: >> "Ok"
[120-H]: "Let me know when the camps are completed."
[121-S]: [00028] = (Event <event camp> <type completion>)
[122-M]: (@Request <request ($Notify <notification (00028)>)>)
[123-R]: (@Request <request ($Notify <notification ?A>)>)
         => (#RegisterNotification <notification ?A>)
[124-#]: (#RegisterNotification <notification ?A>)
         => ($Confirm <confirmation ?B>)
[125-S]: [00029] = (Event <event (00022)> <type completion> )
[126-S]: [00030] = (Event <event (00027)> <type completion> )
[127-M]: ($Confirm
            <confirmation (Conjunction
               <first (@Request <request ($Notify <notification (00029)>)>)>
               <second (@Request <request ($Notify <notification (00030)>)>)> )>)
[128-A]: >> "Ok"
[129-H]: "How much time is left in the game?"
[130-S]: [00031] = (Timespan <begin now> <end (Event <event game> <type completion>)>)
[131-S]: [00032] = (QueryParameter <content (00031)> <parameter Timespan:value>)
[132-M]: (@Query <query (00032)>)
[133-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[134-#]: (#EvaluateQuery <query ?A>) => ($Answer <polarity ?B> <confidence ?C>)
[135-M]: ($Answer <polarity ambiguous> <confidence 20%> )
[136-A]: >> "I'm not sure."
[137-H]: "More than 5 minutes?"
[138-S]: [00033] = (MagnitudeRelation
            <relationship greater-than>
            <magnitude (Quantity <value 5> <content minutes>)> )>)
[139-M]: (@Query <query (00033)>)
[140-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[141-#]: (#EvaluateQuery <query ?A>) => ($Query <query ?B>)
[142-S]: [00034] = (MissingParameter <content (00033)> <parameter Relation:reference> )
[143-M]: ++($Query <query (00034)>)++
[144-R]: {  ($Query <query (QueryParameter <content ?A> <parameter ?B>)>),
            (%ParameterMatchInFocus <content ?A> <parameter ?B>) -> ?C }
         => {  (%AddParameter <original ?A> <parameter ?B> <value ?C>) -> ?D ),
               (%GetIntent <content ?D>) -> ?E,
               (^HandleResponse <content ?D> <intent ?E>),
               --($Query <query (QueryParameter <content ?A> <parameter ?B>)>)-- }
```

Table C2 Continued

```
[145-R]: (^HandleResponse <content ?A> <intent query>) => (@Query <query ?A>)
[146-S]: [00035] = (MagnitudeRelation
            <relationship greater-than>
            <magnitude (Quantity <value 5> <content minutes>)>
            <reference (00031)>
[147-M]: (@Query <query (00035>)
[148-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[149-#]: (#EvaluateQuery <query ?A>)
         => ($Answer)
[150-S]: [00036] = ($Answer <polarity affirmative> <confidence 100%>)
[151-M]: (00036)
[152-A]: >> “Yes"
[153-H]: "Why?"
[154-S]: [00037] = (Explanation)
[155-M]: (@Query <query (00034)>)
[156-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[157-#]: (#EvaluateQuery <query ?A>) => ($Query <query ?B>)
[158-S]: [00038] = (MissingParameter
            <content (00037)>
            <parameter Explanation:content> )
[159-M]: ++($Query <query (00038)>)++
[160-R]: {  ($Query <query (QueryParameter <content ?A> <parameter ?B>)>),
            (%ParameterMatchInFocus <content ?A> <parameter ?B>) -> ?C }
         => {  (%AddParameter <original ?A> <parameter ?B> <value ?C>) -> ?D ),
               (%GetIntent <content ?D>) -> ?E,
               (^HandleResponse <content ?D> <intent ?E>),
               --($Query <query (QueryParameter <content ?A> <parameter ?B>)>)-- }
[161-R]: (^HandleResponse <content ?A> <intent query>)
         => (@Query <query ?A>)
[162-S]: [00039] = (Explanation
            <content ($Answer
               <polarity affirmative>
               <confidence 100%> )>
               <content (00035)> )>)
[163-M]: (@Query <query (00039)>)
[164-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[165-#]: (#EvaluateQuery <query ?A>) =>
            ($Answer <content ?B>)
[166-S]: [00040] = (Quantity <value 1> <content opponent>)
[167-S]: [00041] = (CompareRelation
            <relationship equal>
            <reference1 (Modifier <modifier our> <content resources>)>
            <reference2 (Modifier <modifier their> <content resources>)> )
[168-S]: [00042] = (StateInTime <time begin> <content (00041)>)
[169-S]: [00043] = (MagnitudeRelation
         <relation equals>
         <magnitude 0>
         <reference (Modifier <modifier their> <content resources>)> )
[170-S]: [00044] = (Inference
            <if (00043)>
            <then (Event <event game> <type completion>)> )
[171-S]: [00045] = (Plan <objective (00043)>)
[172-S]: [00046] = (Timespan <value (00045):duration:minimum>)
[173-S]: [00047] = (MagnitudeRelation
            <relation at-least>
            <magnitude (Quantity <value 10> <content minutes>)>
            <reference (00046)> )
[174-S]: [00048] = (MagnitudeRelation
            <relation at-least>
            <magnitude (Quantity <value 10> <content minutes>)>
            <reference (00031)> )
[175-S]: [00049] = (Argument
            <givenfact (00040)>
            <givenfact (00042)>
            <givenrule (00044)>
            <assumerule (00047)>
            <conclusion (00048)> )
```

Table C2 Continued

```
[176-S]: [00050] = (Explanation
            <content ($Answer
               <polarity affirmative>
               <confidence 100%>
               <content (00035)> )>
            <explanation (00049)> )
[177-M]: ($Answer <content (00050)>)
[178-A]: >> “Given that there is only one opponent, and they start with the same
             resources."
[179-A]: >> “It would take at least 10 minutes to build an army, find them and destroy
             what resources we know they have."
[180-H]: "So, 10 minutes?"
[181-S]: [00051] = (ConclusionOf <conclusion (Quantity <value 10> <content minutes>) >)
[182-M]: (@Query <query (00051)>)
[183-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[184-#]: (#EvaluateQuery <query ?A>) => ($Query <query ?B>)
[185-S]: [00052] = (MissingParameter <content (00051)> <parameter Conclusion:argument> )
[186-M]: ++($Query <query (00052)>)++
[187-R]: {  ($Query <query (QueryParameter <content ?A> <parameter ?B>)>),
            (%ParameterMatchInFocus <content ?A> <parameter ?B>) -> ?C }
         => {  (%AddParameter <original ?A> <parameter ?B> <value ?C>) -> ?D ),
               (%GetIntent <content ?D>) -> ?E,
               (^HandleResponse <content ?D> <intent ?E>),
               --($Query <query (QueryParameter <content ?A> <parameter ?B>)>)-- }
[188-R]: (^HandleResponse <content ?A> <intent query>)
         => (@Query <query ?A>)
[189-S]: [00053] = (ConclusionOf
            <argument (00049)>
            <conclusion (Quantity <value 10> <content minutes>)> )
[190-M]: (@Query <query (00053)>)
[191-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[192-#]: (#EvaluateQuery <query ?A>) => ($Reject <rejection ?B> <justification ?C>)
[193-S]: [00054] = (MagnitudeRelation
            <relation equals>
            <magnitude (Quantity <value 10> <content minutes>)>
            <reference (00046)> )
[194-S]: [00055] = (ConclusionOf <argument (00049)> <conclusion (00054)>)
[195-S]: [00056] = (MagnitudeRelation
         <relation equals>
         <magnitude 0>
         <reference (Modifier <modifier our> <content resources>)> )
[196-S]: [00057] = (Desire <objective (00056)> <who opponent>)
[197-S]: [00058] = (Plan
            <objective (00056)>
            <who opponent>
            <method (Disjunction
               <first (Action <name train>)>
               <second (Action <name build>)> )>
            <confidence 95%>
[198-S]: [00059] = (Inference
            <if (Action <name train>)>
            <then (ChangeQuantity
               <type increase>
               <content (Modifier <modifier Action:team> <content resources>)> )>)
[199-S]: [00060] = (Inference
            <if (Action <name build>)>
            <then (ChangeQuantity
               <type increase>
               <content (Modifier <modifier Action:team> <content resources>)> )>)
[200-S]: [00061] = (ChangeQuantity
            <type increase>
            <content (Modifier <modifier their> <content resources>)>
            <confidence 95%> )
[201-S]: [00062] = (MagnitudeRelation
            <relation greater-than>
            <magnitude (Quantity <value 10> <content minutes>)>
            <reference (00046)> )
```

Table C2 Continued

```
[202-S]: [00063] = (MagnitudeRelation
            <relation greater-than>
            <magnitude (Quantity <value 10> <content minutes>)>
            <reference (00031)> )
[203-S]: [00064] = (Argument
            <assumefact (00057)>
            <assumefact (00058)>
            <givenrule (00059)>
            <givenrule (00060)>
            <assumefact (00061)>
            <assumefact (00062)>
            <conclusion (00063)> )
[204-M]: ($Reject <rejection (00055)> <justification (00064)>
[205-A]: >> “No.  They are most likely building their resources, it will take longer."
[206-M]: ($Notify <notification (Event <event (00029)> <type completion> )>)
[207-A]: >> “The training camp has been finished.”
[208-H]: "Show me."
[209-S]: [00065] = (Action <name present>)
[210-M]: (@Order <orders (00065)>)
[211-R]: (@Order <orders Action:?A>)
         => (#EvaluateAction <action ?A> <intent execute>)
[212-#]: (#EvaluateAction <action ?A> <intent execute>)
         => ($Query <query ?B>)
[213-S]: [00066] = (MissingParameter
            <content (00065)>
            <parameter Action:target> )
[214-M]: ++($Query <query (00066)>)++
[215-R]: {  ($Query <query (QueryParameter <content ?A> <parameter ?B>)>),
            (%ParameterMatchInFocus <content ?A> <parameter ?B>) -> ?C }
         => {  (%AddParameter <original ?A> <parameter ?B> <value ?C>) -> ?D ),
               (%GetIntent <content ?D>) -> ?E,
               (^HandleResponse <content ?D> <intent ?E>),
               --($Query <query (QueryParameter <content ?A> <parameter ?B>)>)-- }
[216-S]: [00067] = (Action <name present> <target (00029)>)
[217-R]: (^HandleResponse <content (Action:?A)> <intent ?B>)
         => (#EvaluateAction <action ?A> <intent ?B>)
[218-#]: (#EvaluateAction <action ?A> <intent execute>)
[219-D]: ‘’Training camp ‘camp1’ is shown on the screen and highlighted.’’
[220-M]: ($Notify <notification (Event <event (00030)> <type completion> )>)
[221-A]: >> “The second training camp has been finished.”
[222-H]: "Create a squad."
[223-S]: [00068] = (Action <name create> <target ‘a squad’>)
[224-M]: (@Order <orders (00068)>)
[225-R]: (@Order <orders Action:?A>)
         => (#EvaluateAction <action ?A> <intent execute>)
[226-#]: (#EvaluateAction <action ?A> <intent execute>)
         => ($Query <query ?B>)
[227-S]: [00069] = (Procedure <objective (00068)>)
[228-S]: [00070] = (QueryParameter <content (00069)> <parameter Procedure:procedure>)
[229-M]: ++($Query <query (00070)>)++
[230-A]: >> “How do I create a squad?"
[231-H]: "First, train six soldiers."
[232-S]: [00071] = (Action <name train> <target (Quantity <value 6> <content soldier>)>)
[233-S]: [00072] = (Modifier <modifier first> <content (00071)>)
[234-M]: (@Order <orders (00072)>)
[235-R]: {  ??($Query <query (QueryParameter:?A)>)??,
            (@Order <orders ?B>),
            (#QueryResponseMatch <query ?A> response <?B>) }
         => {  (^RewriteQuery <query ?A> <response ?B>) -> ?C,
            (%GetIntent <content ?C>) -> ?D,
            (^HandleResponse <content ?C> <intent ?D>),
            --($Query <query ?B>)-- }
[236-R]: (^RewriteQuery
            <query (QueryParameter <content ?A> <parameter ?B> )>
            <response ?C> )
         -> (%AddParameter <original ?A> <parameter ?B> <value ?C>)
[237-S]: [00073] = (Procedure <objective (00068)> <procedure (00072)> )
```

Table C2 Continued

```
[238-R]: (^HandleResponse <content (Procedure:?A)> <intent ?B>)
         => (#EvaluateAction <action ?A> <intent ?B>)
[239-#]: (#EvaluateAction <action ?A> <intent execute>)
         => {  ($Confirm <confirmation ?A>),
               ++(^LookForward <content (00072)> <key first>)++ }
[240-#]: ($Confirm <confirmation (00073)>)
[241-A]: >> “Ok"
[242-H]: "Then, group them together."
[243-S]: [00074] = (Action <name group> <content them>)
[244-S]: [00075] = (Modifier <modifier then> <content (00074)>)
[245-M]: (@Order <orders (00074)>)
[246-R]: {  (@Order <orders (Modifier <modifier then> <content ?B>) >),
            ??(^LookForward <content ?A> <key first>)?? }
         => {  (^MergeConcepts
                  <first ?A>
                  <second (Modifier <modifier then> <content ?B>)> ) -> ?C,
               (^HandleChange <original ?A> <new ?C>),
               --(^LookForward <content ?A> <key first>)-- }
[247-R]: (^MergeConcepts
            <first (Modifier <modifier first> <content (Action:?A)>
            <second (Modifier <modifier then> <content (Action:?B)> )
         -> (ActionSequence <first ?A> <second ?B>)
[248-S]: [00076] = (ActionSequence
            <first (00071)>
            <second (Action <name group> <target first:result> )> )
[249-R]: (^HandleChange <original ?A> <new ?B>)
         => {  (%GetRootConcept <concept ?A>) -> ?C,
               (%GetIntent <content ?C>) -> ?D,
               (%FindParameter <root ?C> <search ?A>) -> ?E
               (%ReplaceParameter <original ?C> <parameter ?E> <value ?B>) -> ?F
               (#EvaluateChangedConcept <original ?C> <new ?F> <intent ?D>) }
[250-S]: [00077] = (Procedure <objective (00068)> <procedure (00076)> )
[251-#]: (#EvaluateChangedConcept <original ?A> <new ?B> <intent execute>)
         => ($Confirm <confirmation ?C>)
[252-S]: [00078] = (Change <original (00073)> <new (00077)>)
[253-M]: ($Confirm <confirmation (00078)>)
[254-A]: >> “Ok"
[255-H]: "The group is called a squad."
[256-S]: [00079] = (Nomenclature <usage squad> <meaning ‘the group’>)
[257-M]: (@Tell <content (00079)>)
[258-R]: (@Tell <content (Nomenclature:?A>)
         => (#EvaluateKnowledge <knowledge ?A> <intent learn>)
[259-#]: (#EvaluateKnowledge <knowledge ?A> <intent learn>)
         => ($Confirm <confirmation ?B>)
[260-S]: [00080] = (Knowledge <knowledge (00079)>)
[261-M]: ($Confirm <confirmation (00080)>)
[262-A]: >> “I understand"
[263-H]: "Have the other camp create a squad as well."
[264-S]: [00081] = (Action
            <name create>
            <performer (Modifier <modifier other> <content camp>)>
            <target squad> )
[265-M]: (@Order <orders (00081)>)
[266-R]: (@Order <orders Action:?A>)
         => (#EvaluateAction <action ?A> <intent execute>)
[267-#]: (#EvaluateAction <action ?A> <intent execute>)
         => ($Confirm <confirmation ?A>)
[268-M]: ($Confirm <confirmation (00081)>)
[269-A]: >> “Ok"
[270-H]: "Make two more squads, one at each camp."
[271-S]: [00082] = (Reference
            <reference one>
            <content (Modifier <modifier each> <content camp> )> )
[272-S]: [00083] = (Action
            <name create>
            <target (Quantity <value 2> <content squad> )>
            <method (00082)> )
```

Table C2 Continued

```
[273-M]: (@Order <orders (00083)>)
[274-R]: (@Order <orders Action:?A>)
         => (#EvaluateAction <action ?A> <intent execute>)
[275-#]: (#EvaluateAction <action ?A> <intent execute>)
         => ($Confirm <confirmation ?A>)
[276-S]: [00084] = (Action <name crete> <performer camp1> <target squad> )
[277-S]: [00085] = (Action <name crete> <performer camp2> <target squad> )
[278-S]: [00086] = (Conjunction <first (00084)> <second (00085)>)
[279-M]: ($Confirm <confirmation (00086)>)
[280-A]: >> “Ok"
[281-H]: "Let's make missiles."
[282-S]: [00087] = (Action <name create> <target missle>)
[283-M]: (@Propose <proposal (00087)>)
[284-R]: (@Propose <proposal Action:?A>)
         => (#EvaluateProposal <proposal ?A> <intent evaluate>)
[285-#]: (#EvaluateProposal <proposal ?A> <intent evaluate>)
         => ($CounterPropose <original ?A> <proposal ?B>)
[286-S]: [00088] = (Action <name train> <target ‘upgraded soldiers’>)
[287-S]: [00089] = (Action <name build> <target hospital>)
[288-S]: [00090] = (ActionSequence <first (00088)> <second (00089)>)
[289-M]: ++($CounterPropose <original (00087)> <proposal (00090)>)++
[290-A]: >> “No, I think we should train upgraded soldiers, then build a hospital."
[291-H]: "Why?"
[292-S]: [00091] = (Explanation)
[293-M]: (@Query <query (00091)>)
[294-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[295-#]: (#EvaluateQuery <query ?A>) => ($Query <query ?B>)
[296-S]: [00092] = (MissingParameter
            <content (00091)>
            <parameter Explanation:content> )
[297-M]: ++($Query <query (00092)>)++
[298-R]: {  ($Query <query (QueryParameter <content ?A> <parameter ?B>)>),
            (%ParameterMatchInFocus <content ?A> <parameter ?B>) -> ?C }
         => {  (%AddParameter <original ?A> <parameter ?B> <value ?C>) -> ?D ),
               (%GetIntent <content ?D>) -> ?E,
               (^HandleResponse <content ?D> <intent ?E>),
               --($Query <query (QueryParameter <content ?A> <parameter ?B>)>)-- }
[299-R]: (^HandleResponse <content ?A> <intent query>) => (@Query <query ?A>)
[300-S]: [00093] = (Explanation
            <content ($CounterPropose <original (00087)> <proposal (00090)>)> )
[301-M]: (@Query <query (00093)>)
[302-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[303-#]: (#EvaluateQuery <query ?A>) =>
            ($Answer <content ?B>)
[304-S]: [00094] = (Plan <objective (00087)>)
[305-S]: [00095] = (Plan <objective (00090)>)
[306-S]: [00096] = (Timespan <value (00094):duration:minimum>)
[307-S]: [00097] = (Timespan <value (00095):duration:minimum>)
[308-S]: [00098] = (CompareRelation
            <relation greater-than>
            <reference1 (00096)>
            <reference2 (00097)> )
[309-S]: [00099] = (Quantity <value (00094):variation> )
[310-S]: [00100] = (Quantity <value (00095):variation> )
[311-S]: [00101] = (CompareRelation
            <relation greater-than>
            <reference1 (00099)>
            <reference2 (00100)> )
[312-S]: [00102] = (Conjunction <first (00098)> <second (00101)>)
[313-S]: [00103] = (Argument
            <givenfact (00096)>
            <givenfact (00097)>
            <assumefact (00098)>
            <givenfact (00099)>
            <givenfact (00100)>
            <assumefact (00101)>
            <conclude (00102)> )
```

Table C2 Continued

```
[314-S]: [00104] = (Explanation
            <content ($CounterPropose <original (00087)> <proposal (00090)>)>
            <explanation (00103)> )
[315-M]: ($Answer <content (00104)>)
[316-A]: >> “Because it is faster and easier than missiles."
[317-H]: "But, I really want missiles."
[318-S]: [00105] = (Desire <objective missle> <magnitude ‘really want’>)
[319-S]: [00106] = (Modifier <modifier but> <content (00105)>)
[320-M]: (@Tell <content (00106)>)
[321-R]: {  (@Tell <content (Modifier <modifier but> <content ?A>)>),
            [$:?B] }
         => (@Reject <rejection ?B> <justification ?A>)
[322-R]: (@Reject <rejection ($Answer <content ?A>)> <justification ?B>)
         => (@Reject <rejection ?A> <justification ?B>) }
[323-S]: [00107] = (@Reject <rejection (00104)> <justification (00105)>)
[324-R]: (@Reject
            <rejection (Explanation <content ($Propose ?A)>)>
            <justification ?B> <intent evaluate>)
         => (#EvaluateRejection <rejection ?B> <justification ?B> <intent evaluate>)
[325-#]: (#EvaluateRejection <proposal ?A> <justification ?B> <intent evaluate>)
         => {  ($CounterPropose <original ?A> <proposal ?C>),
               --($CounterPropose <original ?D> <proposal ?A> )-- }
[326-S]: [00108] = (ActionSequence <first (00088)> <second (00087)>)
[327-M]:  ++($CounterPropose <original (00090)> <proposal (00108)>)++
[328-A]: >> “How about we train upgraded soldiers then build missiles?"
[329-H]: "Alright, what do we need to do for upgraded soldiers?"
[330-S]: [00109] = (Procedure <objective ‘upgraded soldiers’>)
[331-M]: (Conjunction <first (@Accept)> <second (@Query <query (00109)>)>)
[332-R]: (Conjunction <first ?A> <second ?B>)
         => {  (?A):resolve#->?C,
               (?B):resolve#->?D,
               (^ResolveConjunction <first ?C> <second ?D>) }
[333-M]: (@Accept)
[334-R]: {  (@Accept !<acceptance ?A>),
            ??($Propose <proposal ?B>)?? }
         => {  (@Accept <acceptance ($Propose <proposal ?B>)>),
               --($Propose <proposal ?B>)-- }
[335-M]: (@Accept <acceptance ($Propose <proposal (00108)>)>)
[336-R]: (@Accept <acceptance ($Propose <proposal (Action:?A)>)>)
         => {  (%GetIntent <content ?A>) -> ?B,
               (#EvaluateAction <action ?A> <intent ?B>) }
[337-M]: (@Query <query (00109)>)
[338-R]: (@Query <query (Procedure <objective ?A>)>)
         => (#EvaluateQuery <query (Procedure <objective ?A>)>)
[339-M]: (^ResolveConjunction
            <first (#EvaluateAction <action (00108) <intent adopt>)>
            <second (#EvaluateQuery <query (00109)>)> )
[340-R]: {  (^ResolveConjunction
               <first (#EvaluateAction <action ?A> <intent adopt>)>
               <second (#EvaluateQuery <query (Procedure <objective ?B>)>)> ),
            (%ContainsConcept <content ?A> <contains ?B>) }
         => {  (Plan <objective ?A>) -> ?C,
               (%FindParameter <root ?C> <search ?B>) -> ?D,
               (#EvaluateAction <action ?C> <intent adopt>),
               (#EvaluateAction <action ?C> <intent plan> <focus ?D>) }
[341-S]: [00110] = (Plan <objective (00108)>)
[342-#]: (#EvaluateAction <action ?A> <intent adopt>) => {}
[343-#]: (#EvaluateAction <action ?A> <intent plan> <focus ?B>)
         => ($Plan <content ?C> <focus ?D>)
[344-S]: [00111] = (Action <name build> <target ‘research lab’>)
[345-S]: [00112] = (Action <name research> <target explosives>)
[346-S]: [00113] = (ActionSequence <first (00111)> <second (00112)>)
[347-S]: [00114] = (Action <name research> <target missiles>)
[348-S]: [00115] = (Action <name build> <target ‘missle silo’>)
[349-S]: [00116] = (Action <name build> <target ‘missle’>)
[350-S]: [00117] = (ActionSequence <first (00114)> <second (00115)> <third (00116)>)
[351-S]: [00118] = (Plan <objective (00088)> <procedure (00113)>)
```

Table C2 Continued

```
[352-S]: [00119] = (Plan <objective (00087)> <procedure (00117)>)
[353-S]: [00120] = (Plan
               <objective (00108)>
               <procedure (Sequence <first (00118)> <second (00119)>)> )
[354-M]: ++($Plan <content (00120)> <focus Plan:procedure:first:procedure:first>)++
[355-A]: >> “First, we should build a research lab."
[356-H]: "How about just north of the vault?"
[357-S]: [00121] = (RelativeLocation
            <direction north>
            <distance ‘just’>
            <reference vault> )
[358-M]: (@Propose <proposal (00121)>)
[359-R]: {  (@Propose <proposal ?A>),
            ??($Plan <content ?B> <focus ?C>)??,
            (!(%Collision <first ?C> <second ?A>)) }
         => {  (%GetIntent <content ?B>) -> ?D,
               (#EvaluateProposal
                  <proposal (Refinement <original ?C> <refinement ?A>)>
                  <intent ?D> ) }
[360-#]: (#EvalutaeProposal <proposal (?A)> <intent plan>)
         => {  ($Answer <content ?B>),
               ($Plan <content ?C> <focus ?D>),
               --($Plan <content ?E> <focus ?F>)-- }
[361-M]: ($Accept <acceptance (00121)>)
[362-A]: >> “Ok"
[363-S]: [00122] = (Action <name build> <target ‘research lab’> <location (00121)>)
[364-S]: [00123] = (ActionSequence <first (00122)> <second (00112)>)
[365-S]: [00124] = (Plan <objective (00088)> <procedure (00123)>)
[366-S]: [00125] = (Plan
               <objective (00108)>
               <procedure (Sequence <first (00124)> <second (00119)>)> )
[367-M]: ++($Plan <content (00125)> <focus Plan:procedure:first:procedure:second>)++
[368-A]: >> “Then, we'll need to research explosives."
[369-H]: "Go attack the enemy."
[370-S]: [00126] = (Action <name attack> <target opponent>)
[371-M]: (@Order <orders (00126)>)
[372-R]: (@Order <orders Action:?A>)
         => (#EvaluateAction <action ?A> <intent execute>)
[373-#]: (#EvaluateAction <action ?A> <intent execute>)
         => {  ($Query <query ?B>)
[374-S]: [00127] = (Action
            <name attack>
            <performer (Quantity <content soldier>)
            <target opponent> )
[375-S]: [00128] = (QueryParameter
            <content (00127)>
            <parameter Action:performer:Quantity:value> )
[376-M]: ++($Query <query (00128)>)++
[377-A]: >> “How many soldiers should go?"
[378-H]: "Everyone."
[379-S]: [00129] = (Quantity <value all> <content person>)
[380-M]: (@Tell <content (00129)>)
[381-R]: {  (@Tell <content ?A>),
            ??($Query <query ?B>)??,
            (#QueryResponseMatch <query ?B> <response ?A>) }
         => {  (^RewriteQuery <query ?B> <response ?A>) -> ?C,
               (%GetIntent <content ?C>) -> ?D,
               (^HandleResponse <content ?C> <intent ?D>,
               --($Query <query ?B>)-- }
[382-R]: (^RewriteQuery
            <query (MissingParameter <content ?A> <parameter ?B> )>
            <response ?C> )
         -> (%AddParameter <original ?A> <parameter ?B> <value ?C>)
[383-R]: (^HandleResponse <Action:?A> <intent ?B>)
         => (#EvaluateAction <action ?A> <intent ?B>)
[384-#]: (#EvaluateAction <action ?A> <intent execute>) =>
            ($Confirm <confirmation ?A>)
```

Table C2 Continued

```
[385-S]: [00130] = (Action
            <name attack>
            <performer (Quantity <value all> <content soldier>)
            <target opponent> )
[386-M]: ($Confirm <confirmation (00130)>)
[387-A]: >> “Ok"
[388-H]: "Make two more squads at each camp."
[389-S]: [00131] = (Quantity <value 2> <content squad>)
[390-S]: [00132] = (Modifier <modifier each> <content camp>)
[391-S]: [00133] = (Action <name create> <target (00131)> <location (00132)>)
[392-M]: (@Order <orders (00133)>)
[393-R]: (@Order <orders Action:?A>)
         => (#EvaluateAction <action ?A> <intent execute>)
[394-#]: (#EvaluateAction <action ?A> <intent execute>)
         => ($Confirm <confirmation ?A>)
[395-S]: [00134] = (Action <name crete> <performer camp1> <target (00131)> )
[396-S]: [00135] = (Action <name crete> <performer camp2> <target (00131)> )
[397-S]: [00136] = (Conjunction <first (00134)> <second (00135)>)
[398-M]: ($Confirm <confirmation (00136)>)
[399-A]: >> “Ok"
[400-H]: "Where were we?"
[401-M]: (@Query <query (Focus)>)
[402-R]: {  (@Query <query (Focus)>),
            ??(?A)?? }
         => {  (%GetIntent <content ?A>) -> ?B,
               (^HandleFocus <content ?A> <intent ?B>),
               (?A) }
[403-R]: (^HandleFocus <content ?A> <intent plan>)
         => (@Tell <content ?A>)
[404-M]: ($Tell <content (00125)>)
[405-A]: >> “We were planning to build a research lab just north of the vault."
[406-M]: ++($Plan <content (00125)> <focus Plan:procedure:first:procedure:second>)++
[407-A]: >> “We then need to research explosives with it."
[408-H]: "Sounds good.  How long will it take?"
[409-S]: [00137] = (Timespan <begin now> <end (Event <event it> <type completion>)>)
[410-S]: [00138] = (QueryParameter <content (00137)> <parameter Timespan:value>)
[411-M]: (Conjunction <first (@Accept)> <second (@Query <query (00138)>)>)
[412-R]: (Conjunction <first ?A> <second ?B>)
         => {  (?A):resolve#->?C,
               (?B):resolve#->?D,
               (^ResolveConjunction <first ?C> <second ?D>) }
[413-M]: (@Approve)
[414-R]: {  (@Approve !<content ?A>),
            ??($Plan <content ?B> <focus ?C>)??,
            (!(%Collision <first ?C> <second ?A>)) }
         => {  (@Approve <content ?C>),
               --($Plan <content ?B> <focus ?C>)-- }
[415-M]: (@Approve <content (00112)>)
[416-R]: (@Approve <content (Action:?A)>)
         => {  (%GetIntent <content ?A>) -> ?B,
               (#EvaluateAction <action ?A> <intent ?B>) }
[417-M]: (@Query <query (00138)>)
[418-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[419-M]: (^ResolveConjunction
            <first (#EvaluateAction <action (00112) <intent adopt>)>
            <second (#EvaluateQuery <query (00138)>)> )
[420-R]: (^ResolveConjunction <first (#:?A)> <second (#:?B)>) => {  (?A), (?B) }
[421-#]: (#EvaluateAction <action ?A> <intent adopt>) => {}
[422-#]: (#EvaluateQuery <query ?A>) => ($Answer <content ?B>)
[423-S]: [00139] = (Quantity <value 1630> <content cycles>)
[424-S]: [00140] = (Timespan
            <begin now>
            <end (Event <event (00124)> <type completion>)>
            <value (00139)>)
[425-M]: ($Answer <content (00140)>)
[426-A]: >> “1630 cycles."
[427-H]: "Ok, do it."
```

Table C2 Continued

```
[428-M]: (Conjunction <first (@Acknowledge)> <second (@Execute <content it>)>)
[429-R]: (Conjunction <first ?A> <second ?B>)
         => {  (?A):resolve#->?C,
               (?B):resolve#->?D,
               (^ResolveConjunction <first ?C> <second ?D>) }
[430-M]: (@Acknowledge)
[431-R]: (@Acknowledge) => {}
[432-M]: (@Execute <content it>)
[433-R]: { (@Execute !<content ?A>),
           [(Action|Plan|Procedure:?B)] }
         => (#EvaluateAction <action ?B> <intent execute>)
[434-M]: (^ResolveConjunction <first ({})> <second ?A>) => (?A)
[435-#]: (#EvaluateAction <action ?A> <intent execute>) => (Confirm <confirmation ?B>)
[436-M]: ($Confirm <confirmation (00124)>)
[437-A]: >> “Ok"
[438-H]: "Then what do we have to do to build missiles?"
[439-S]: [00141] = (Action <name build> <target missle>)
[440-S]: [00142] = (Procedure <objective (00141)>)
[441-S]: [00143] = (Modifier <modifier then> <content (00142)>)
[442-M]: (@Query <query (00143)>)
[443-R]: (@Query <query (Modifier <modifier then> <content ?A>)
         => (@Query <query ?A>)
[444-M]: (@Query <query (00142)>)
[445-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[446-#]: (#EvaluateQuery <query ?A>) => ($Answer <content ?B>)
[447-M]: ($Answer <content (00117)>)
[448-A]: >> “First, we have to research missiles in the research lab."
[449-A]: >> “Then, we have to build a missle silo."
[450-A]: >> “Finally, we have to build a missle."
[451-H]: "How long until we can start building a silo?"
[452-S]: [00144] = (Event <event (Action <name build> <target silo>)>) <type capable>)
[453-S]: [00145] = (Timespan <begin now> <end (00144)> <type completed>)
[454-S]: [00146] = (QueryParameter <content (00145)> <parameter Timespan:value>)
[455-M]: (@Query <query (00146)>)
[456-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[457-#]: (#EvaluateQuery <query ?A>) => ($Answer <content ?B>)
[458-S]: [00147] = (Quantity <value 3500> <name cycles> <accuracy approx>)
[459-S]: [00148] = (Timespan <begin now> <end (00144)> <type completed> <value (00147)>)
[460-M]: ($Answer <content (00148)>)
[461-A]: >> “Approximately 3500 cycles."
[462-H]: "Forget it!"
[463-M]: (@Abandon <content it>)
[464-R]: { (@Abandon !<content ?A>),
           [(Action|Plan|Procedure:?B)] }
         => (#EvaluateAction <action ?B> <intent abandon>)
[465-S]: [00149] = ($Abandon <content (00117)>)
[466-M]: ($Confirm <confirmation (00149)>)
[467-H]: "How long does it take to build a squad?"
[468-S]: [00150] = (Action <name build> <target squad>)
[469-S]: [00151] = (Procedure <objective (00150)>)
[470-S]: [00152] = (QueryParameter <content (00151)> <parameter Procedure:duration>)
[471-M]: (@Query <query (00152)>)
[472-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[473-#]: (#EvaluateQuery <query ?A>) => ($Answer <content ?B>)
[474-S]: [00153] = (Quantity <value 860> <name cycles> <accuracy approx>)
[475-S]: [00154] = (Procedure <objective (00150)> <duration (00153)>)
[476-M]: ($Answer <content (00154)>)
[477-A]: >> “Approximately 860 cycles."
[478-H]: "That's not good enough!"
[479-M]: (@Assert <content dissatisfaction>)
[480-R]: {  (@Assert <content dissatisfaction>),
            [$:?A] }
         => (^HandleFeedback <content ?A> <feedback dissatisfied> )
[481-R]: (^HandleFeedback <content ?A> <feedback dissatisfied>)
         => {  (%GetFocus ?A) -> ?B,
               (Problem <content ?A> <focus ?B> <type dissatisfied>) -> ?C,
               (#EvaluateProblem <content ?C> <intent fix>) }
```

Table C2 Continued

```
[482-S]: [00155] = (Problem
            <content (00154)>
            <focus Procedure:duration>
            <type dissatisfied>
[483-#]: (#EvaluateProblem <content (Problem ?A)> <intent fix>)
         => ($Propose <proposal ?B>)
[484-S]: [00156] = (Action
            <name build>
            <target (Modifier <modifier more> <content 'training camp'>)>
[485-S]: [00157] = (Solution <problem (00155)> <solution (00156)>)
[486-M]: ($Propose <proposal (00157)>)
[487-A]: >> "We could build more training camps."
[488-H]: "Take care of it."
[489-M]: (@ShiftAutonomy <content it> <responsible agent>)
[490-R]: {  (@ShiftAutonomy <content it> <responsible ?A>),
            [(Action|Plan|Procedure:?B)] }
         => (#EvaluateAutonomicShift <content ?B> <responsible ?A>)
[491-R]: (#EvaluateAutonomicShift <content ?A> <responsible ?B>)
         => ($Confirm <confirmation ?C>)
[492-S]: [00158] = (AutonomicShift <content (00157)> <resonsible agent>)
[493-M]: ($Confirm (00158))
[494-A]: >> "Ok"
[495-#]: (#EvaluateAction <content ?A> <intent execute>)
         => ($Warn <content ?A>)
[496-S]: [00159] = (RestrictionQuantity <content resources> <polarity insufficient>)
[497-S]: [00160] = (ConsequenceOf <content (00156)> <consequence (00159)>)
[498-M]: ($Warn <content (00160)>)
[499-A]: >> "We are going to run low on resources if we build more troops."
[500-H]: "Then train more engineers."
[501-S]: [00161] = (Action
            <name train>
            <target (Modifier <modifier more> <content engineer>)> )
[502-S]: [00162] = (Modifier <modifier then> <content (00161)>)
[503-M]: (@Order <orders (00162)>)
[504-R]: {  (@Order <orders (Modifier <modifier then> <content ?A>)>),
            [$Warn:?B],
            (%GetIntent <content ?B>) -> ?C }
         => (#EvaluateAction <action ?A> <intent ?C>)
[505-#]: (#EvaluateAction <action ?A> <intent execute>)
         => ($Query <query ?B>)
[506-S]: [00163] = (Action <name mine>)
[507-S]: [00164] = (Action
            <name train>
            <target (Quantity <content engineer>)>
            <intent (00163)> )
[508-S]: [00165] = (MissingParameter
            <content (00164)>
            <parameter Action:target:Quantity:value>)
[509-S]: [00166] = (MissingParameter <content (00163)> <parameter Action:target>)
[510-S]: [00167] = (Conjunction <first (00165)> <second (00166)>)
[511-M]: ++($Query <query (00167)>)++
[512-A]: >> "How many?  What should they mine?"
[513-H]: "You decide."
[514-M]: (@ShiftAutonomy <content answer> <responsible agent>)
[515-R]: {  (@ShiftAutonomy <content answer <responsible ?A>),
            ??($Query <query ?B>)?? }
         => {  (#EvaluateAutonomicShift <content ?B> <responsible ?A>),
               --($Query <query ?B>)-- }
[516-#]: (#EvaluateAutonomicShift <content ?A> <responsible ?B>)
         => ($Confirm <confirmation ?C>)
[517-S]: [00168] = (AutonomicShift <content (00167)> <responsible agent>)
[518-M]: ($Confirm <confirmation (00168)>)
[519-A]: >> "Ok"
[520-H]: "Whenever a squad is ready, have it attack the enemy."
[521-S]: [00169] = (Event <event squad> <type created>)
[522-S]: [00170] = (Action <name attack> <target opponent> <performer it>)
[523-S]: [00171] = (ContinualAction <condition (00169)> <action (00170)>)
```

Table C2 Continued

```
[524-M]: (@Order <orders (00171)>)
[525-R]: (@Order <orders Action:?A>)
         => (#EvaluateAction <action ?A> <intent execute>)
[526-#]: (#EvaluateAction <action ?A> <intent execute>)
         => ($Confirm <confirmation ?A>)
[527-M]: ($Confirm <confirmation (00171)>)
[528-A]: >> “Ok"
[529-S]: [00172] = (Event <event (00157)> <type completion>)
[530-M]: ($Notify <notification (00172)>)
[531-A]: >> “We can now make squads twice as fast.”
[532-S]: [00173] = (Event <event (00124)> <type completion>)
[533-M]: ($Notify <notification (00173)>)
[534-A]: >> “We have finished researching explosives.”
[535-S]: [00174] = (Action
            <name train>
            <target (Quantity <value 6> <content ‘upgraded soldier’>)> )
[536-S]: [00175] = (ActionSequence
            <first (00174)>
            <second (Action <name group> target first:result>)> )
[537-S]: [00176] = (Procedure <objective (00068)> <procedure (00175)>)
[538-S]: [00177] = (Modification <original (00073)> <modified (00176)>)
[539-M]: ++($Propose <proposal (00177)>)++
[540-A]: >> “Shall we incorporate upgraded soldiers in new squads?
[541-H]: "Yes"
[542-M]: (@Answer <polarity affirmative>)
[543-R]: {  (@Answer <polarity affirmative>),
            ??($Propose <proposal ?A>)?? }
         => {  (@Accept <acceptance ($Propose <proposal ?A>)>),
               --($Propose <proposal ?A>)—- }
[544-M]: (@Accept <acceptance ($Propose <proposal (00177)>)>)
[545-R]: (@Accept <acceptance ($Propose
            <proposal (Modification <original ?A> <modified ?B> )> )>)
         => {  (%GetIntent <content ?A>) -> ?C,
               (#EvaluateAction <action ?B> <intent ?C>) }
[546-#]: (#EvaluateAction <action ?A> <intent procedural>)
         => ($Confirm <confirmation ?B>)
[547-M]: ($Confirm <confirmation (00177)>)
[548-A]: >> “Ok"
[549-H]: "Make squads as necessary."
[550-S]: [00178] = (Action <name create> <target squad>)
[551-S]: [00179] = (ContinualAction <condition whenever-necessary> <action (00178)>)
[552-M]: (@Order <orders (00179)>)
[553-R]: (@Order <orders Action:?A>) => (#EvaluateAction <action ?A> <intent execute>)
[554-#]: (#EvaluateAction <action ?A> <intent execute>) => ($Confirm <confirmation ?A>)
[555-M]: ($Confirm <confirmation (00179)>)
[556-A]: >> “Ok"
[557-H]: "How many training camps do we have?"
[558-S]: [00180] = (Modifier <modifier our> <content ‘training camp’>)
[559-S]: [00181] = (Quantity <content (00180)>)
[560-S]: [00182] = (QueryParameter <content (00181)> <parameter Quantity:value>)
[561-M]: (@Query <query (00182)>)
[562-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[563-#]: (#EvaluateQuery <query ?A>) => ($Answer <content ?B>)
[564-S]: [00183] = (Quantity <value 4> <content (00180)>)
[565-M]: ($Answer <content (00183)> )
[566-A]: >> “4"
[567-H]: "But there were two before."
[568-S]: [00184] = (Quantity <value 2>)
[569-S]: [00185] = (StateInTime <time before> <content (00184)>)
[570-S]: [00186] = (Modifier <modifier but> <content (00185)>)
[571-M]: (@Tell <content (00186)>)
[572-R]: {  (@Tell <content (Modifier <modifier but> <content ?A>)>),
            [$:?B] }
         => (@Reject <rejection ?B> <justification ?A>)
[573-R]: (@Reject <rejection ($Answer <content ?A>)> <justification ?B>)
         => (@Reject <rejection ?A> <justification ?B>) }
[574-S]: [00187] = (@Reject <rejection (00183)> <justification (00185)>)
```

Table C2 Continued

```
[575-R]: (@Reject <rejection ?A> <justification ?B>)
         => (#EvaluateRejection <rejection ?A> <justification ?B>)
[576-#]: (#EvaluateRejection <rejection ?A> <justification ?B>)
         => ($Answer <content ?C>)
[577-S]: [00188] = (StateInTime <time now> <content (00183)>)
[578-S]: [00189] = (Quantity <value 2> <content (00180)>)
[579-S]: [00190] = (StateInTime <time before> <content (00189)>)
[580-S]: [00191] = (StateChange <from (00190)> <to (00188)>)
[581-M]: ($Answer <content (00191)>)
[582-A]: >> “Now there are four."
[583-H]: "What happened?"
[584-S]: [00192] = (Explanation)
[585-M]: (@Query <query (00192)>)
[586-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[587-#]: (#EvaluateQuery <query ?A>) => ($Query <query ?B>)
[588-S]: [00193] = (MissingParameter
            <content (00192)>
            <parameter Explanation:content> )
[589-M]: ++($Query <query (00038)>)++
[590-R]: {  ($Query <query (QueryParameter <content ?A> <parameter ?B>)>),
            (%ParameterMatchInFocus <content ?A> <parameter ?B>) -> ?C }
         => {  (%AddParameter <original ?A> <parameter ?B> <value ?C>) -> ?D ),
               (%GetIntent <content ?D>) -> ?E,
               (^HandleResponse <content ?D> <intent ?E>),
               --($Query <query (QueryParameter <content ?A> <parameter ?B>)>)-- }
[591-R]: (^HandleResponse <content ?A> <intent query>)
         => (@Query <query ?A>)
[592-S]: [00194] = (Explanation
            <content ($Answer <content (00191)>)> )
[593-M]: (@Query <query (00194)>)
[594-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[595-#]: (#EvaluateQuery <query ?A>) => ($Answer <content ?B>)
[596-S]: [00195] = (Explanation
            <content ($Answer <content (00191)>
            <explanation (00157)> )
[597-M]: ($Answer <content (00195)>)
[598-A]: >> “I built two."
[599-H]: "Why?"
[600-S]: [00196] = (Explanation)
[601-M]: (@Query <query (00196)>)
[602-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[603-#]: (#EvaluateQuery <query ?A>) => ($Query <query ?B>)
[604-S]: [00197] = (MissingParameter
            <content (00196)>
            <parameter Explanation:content> )
[605-M]: ++($Query <query (00038)>)++
[606-R]: {  ($Query <query (QueryParameter <content ?A> <parameter ?B>)>),
            (%ParameterMatchInFocus <content ?A> <parameter ?B>) -> ?C }
         => {  (%AddParameter <original ?A> <parameter ?B> <value ?C>) -> ?D ),
               (%GetIntent <content ?D>) -> ?E,
               (^HandleResponse <content ?D> <intent ?E>),
               --($Query <query (QueryParameter <content ?A> <parameter ?B>)>)-- }
[607-R]: (^HandleResponse <content ?A> <intent query>) => (@Query <query ?A>)
[608-S]: [00198] = (Explanation
            <content ($Answer <content (00195)>)> )
[609-M]: (@Query <query (00194)>)
[610-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[611-#]: (#EvaluateQuery <query ?A>) => ($Answer <content ?B>)
[612-S]: [00199] = (Explanation
            <content (00195)>
            <explanation (00155)> )
[613-M]: ($Answer <content (00199)>)
[614-A]: >> “In order to build squads faster."
[615-H]: "Who told you that you could?"
[616-S]: [00200] = (Permission <responsible agent>)
[617-S]: [00201] = (QueryParameter <content (00200)> <parameter Permission:authority>)
[618-M]: (@Query <query (00201)>)
```

Table C2 Continued

```
[619-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[620-#]: (#EvaluateQuery <query ?A>) => ($Query <query ?B>)
[621-S]: [00202] = (MissingParameter
            <content (00201)>
            <parameter QueryParameter:content:Permission:content> )
[622-M]: ++($Query <query (00202)>)++
[623-R]: {  ($Query <query (QueryParameter <content ?A> <parameter ?B>)>),
            (%ParameterMatchInFocus <content ?A> <parameter ?B>) -> ?C }
         => {  (%AddParameter <original ?A> <parameter ?B> <value ?C>) -> ?D ),
               (%GetIntent <content ?D>) -> ?E,
               (^HandleResponse <content ?D> <intent ?E>),
               --($Query <query (QueryParameter <content ?A> <parameter ?B>)>)-- }
[624-R]: (^HandleResponse <content ?A> <intent query>)
         => (@Query <query ?A>)
[625-S]: [00203] = (Permission
            <content (00156)>
            <responsible agent> )
[626-S]: [00204] = (QueryParameter <content (00203)> <parameter Permission:authority>)
[627-M]: (@Query <query (00204)>)
[628-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[629-#]: (#EvaluateQuery <query ?A>) => ($Answer <content ?B>)
[630-S]: [00205] = (Permission
            <content (00156)>
            <responsible agent>
            <authority human> )
[631-M]: ($Answer <content (00205)>)
[632-A]: >> “You did."
[633-H]: "No, I didn't."
[634-M]: (@Answer <polarity negative>)
[635-R]: {  (@Answer <polarity negative>),
            [$:?B] }
         => (@Reject <rejection ?B>)
[636-R]: (@Reject <rejection ($Answer <content ?A>)>)
         => (@Reject <rejection ?A>) }
[637-S]: [00206] = (@Reject <rejection (00205)>)
[638-R]: (@Reject <rejection ?A>)
         => (#EvaluateRejection <rejection ?A>)
[639-#]: (#EvaluateRejection <rejection ?A>)
         => ($Answer <content ?C>)
[640-S]: [00207] = (Interpretation
            <content (00158)>
[641-S]: [00208] = (Explanation
            <content (00205)>
            <explanation (00207)>
[642-M]: ($Answer <content (00208)>)
[643-A]: >> “Oh, I thought that's what 'Take care of it.' meant."
[644-H]: "Yes, you are right."
[645-S]: [00209] = (Affirmation <type agree>)
[646-M]: (@Assert <content (00209)>)
[647-R]: {  (@Assert <content (Affirmation <type correctness>)>),
            [$Answer:?B] }
         => (@Accept <acceptance ?B>)
[648-R]: (@Accept <acceptance ($Answer <content ?A>)>)
         => (@Accept <acceptance ?A>) }
[649-S]: [00210] = (@Accept <acceptance (00210)>)
[650-R]: (@Accept <acceptance ?A>) => (#EvaluateAcceptance <acceptance ?A>)
[651-#]: (#EvaluateAcceptance <acceptance ?A>) => {}
[652-H]: "How many engineers do we have?"
[653-S]: [00211] = (Modifier <modifier our> <content engineer>)
[654-S]: [00212] = (Quantity <content (00211)>)
[655-S]: [00213] = (QueryParameter <content (00212)> <parameter Quantity:value>)
[656-M]: (@Query <query (00213)>)
[657-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[658-#]: (#EvaluateQuery <query ?A>) => ($Answer <content ?B>)
[659-S]: [00214] = (Quantity <value 23> <content (00211)>)
[660-M]: ($Answer <content (00214)> )
[661-A]: >> “23."
```

Table C2 Continued

```
[662-H]: "What are they doing?"
[663-S]: [00215] = (Action <performer they>)
[664-S]: [00216] = (QueryParameter <content (00215)> <parameter Action:name>)
[665-M]: (@Query <query (00216)>)
[666-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[667-#]: (#EvaluateQuery <query ?A>) => ($Answer <content ?B>)
[668-S]: [00217] = (Quantity <value 13> <content engineer>)
[669-S]: [00218] = (Quantity <value 7> <content engineer>)
[670-S]: [00219] = (Quantity <value 3> <content engineer>)
[671-S]: [00220] = (Action <name mine> <target crystal> <performer (00217)>)
[672-S]: [00221] = (Action <name mine> <target titanium> <performer (00218)>)
[673-S]: [00222] = (Action <name stand> <performer (00219)>)
[674-S]: [00223] = (Conjunction <first (00220)> <second (00221)>)
[675-S]: [00224] = (Conjunction <first (00223)> <second (00222)>)
[676-M]: ($Answer <content (00224)> )
[677-A]: >> “13 are mining crystal, 7 are mining titanium and 3 are standing."
[678-H]: "Show me the ones that are standing."
[679-S]: [00225] = (Subset
            <superset (Reference <reference ones>)>
            <restriction (Action <name standing>)> )
[680-S]: [00226] = (Action <name present> <target (00225)>)
[681-M]: (@Orders <order (00226)>)
[682-R]: (@Order <orders Action:?A>)
         => (#EvaluateAction <action ?A> <intent execute>) }
[683-#]: (#EvaluateAction <action ?A> <intent execute>)
         => ($Query <query ?B> )
[684-S]: [00227] = (MissingParameter
            <content (00225)>
            <parameter Subset:superset>
[685-M]: ++($Query <query (00034)>)++
[686-R]: {  ($Query <query (QueryParameter <content ?A> <parameter ?B>)>),
            (%ParameterMatchInFocus <content ?A> <parameter ?B>) -> ?C }
         => {  (%AddParameter <original ?A> <parameter ?B> <value ?C>) -> ?D ),
               (%GetIntent <content ?D>) -> ?E,
               (^HandleResponse <content ?D> <intent ?E>),
               --($Query <query (QueryParameter <content ?A> <parameter ?B>)>)-- }
[687-R]: (^HandleResponse <content ?A> <intent execute>)
         => (@Order <orders ?A>)
[688-S]: [00228] = (Subset
            <superset (00224)>
            <restriction (Action <name standing>)> )
[689-M]: (@Order <orders (Action <name present> <target (00228)>)>)
[690-R]: (@Order <orders Action:?A>)
         => (#EvaluateAction <action ?A> <intent execute>)
[691-#]: (#EvaluateAction <action ?A> <intent execute>)
         => ($Confirm <confirmation ?A>)
[692-S]: [00229] = (EnumeratedSet
            <first engineer16>
            <second engineer8>
            <third engineer9>)
[693-S]: [00230] = (Action <name present> <target (00229)> <focus EnumeratedSet:first>)
[694-M]: ($Confirm <confirmation (00230)>)
[695-D]: ‘’Engineer ‘engineer16’ is shown on the screen and highlighted.’’
[696-A]: >> “Here is the first."
[697-H]: "What is its history?"
[698-S]: [00231] = (QueryParameter <content it> <parameter it:history>)
[699-M]: (@Query <query (00231)>)
[700-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[701-#]: (#EvaluateQuery <query ?A>) => ($Answer <content ?B>)
[702-S]: [00232] = (Object <name engineer16> <history (00015)>
[703-M]: ($Answer <content (00232)>)
[704-A]: >> “It has built 2 generators."
[705-H]: "Mine titanium"
[706-S]: [00233] = (Action <name mine> <target titanium>)
[707-M]: (@Orders <order (00233)>)
[708-R]: (@Order <orders Action:?A>)
         => (#EvaluateAction <action ?A> <intent execute>)
```

Table C2 Continued

```
[709-#]: (#EvaluateAction <action ?A> <intent execute>)
         => ($Query <query ?B>)
[710-S]: [00234] = (MissingParameter
            <content (00233)>
            <parameter Action:performer> )
[711-M]: ++($Query <query (00066)>)++
[712-R]: {  ($Query <query (QueryParameter <content ?A> <parameter ?B>)>),
            (%ParameterMatchInFocus <content ?A> <parameter ?B>) -> ?C }
         => {  (%AddParameter <original ?A> <parameter ?B> <value ?C>) -> ?D ),
               (%GetIntent <content ?D>) -> ?E,
               (^HandleResponse <content ?D> <intent ?E>),
               --($Query <query (QueryParameter <content ?A> <parameter ?B>)>)-- }
[713-S]: [00235] = (Action <name mine> <target titanium> <performer engineer16>)
[714-R]: (^HandleResponse <content (Action:?A)> <intent ?B>)
         => (#EvaluateAction <action ?A> <intent ?B>)
[715-#]: (#EvaluateAction <action ?A> <intent execute>)
         => {  ($Reject <rejection ?A> <justification ?B>),
               ($Propose
                  <proposal (Modification
                     <original ?A> <modified ?C>)> ) }
[716-S]: [00236] = (RestrictionQuantity
            <content energy>
            <polarity insufficient> )
[717-M]: ($Reject <rejection (00235)> <justification (00236)>)
[718-A]: >> “But we are almost out of energy."
[719-S]: [00237] = (ActionSequence
            <first (Action
               <name build>
               <target (Quantity <value 2> <content generator>)>
               <performer engineer16> )>
            <second (00235)> )
[720-S]: [00238] = (Modification
            <original (00235)>
            <modified (00237)> )
[721-M]: ++($Propose <proposal (00238)>)++
[722-A]: >> “We should have the engineer build two more first."
[723-H]: "Ok"
[724-M]: (@Acknowledge)
[725-R]: {  (@Acknowledge),
            ??($Propose <proposal ?A>)?? }
         => {  (@Accept <content ($Propose <proposal ?A>)>),
               --($Propose <proposal ?A>)-- }
[726-M]: (@Accept <content ($Propose <proposal (00004)>)>)
[727-R]: (@Accept <content ($Propose
            <proposal (Modification <original ?A> <modified ?B> )> )>)
         => {  (%GetIntent <content ?A>) -> ?C,
               (#EvaluateAction <action ?B> <intent ?C>) }
[728-#]: (#EvaluateAction <action ?A> <intent execute>) => {}
[729-H]: "Next"
[730-M]: (@Order <orders (Action <name advance>)>)
[731-R]: {  (@Order <orders (Action <name advance>)>),
            [(Action:?A <name present>] }
         => (^Advance <content ?A>)
[732-R]: (^Advance <content (Action <name present> <target ?A> <focus ?B>)
         => {  (%GetNextInSet <set ?A> <focus ?B>) -> ?C,
               (@Order <orders (Action <name present> <target ?A> <focus ?C>)>) }
[733-R]: (@Order <orders Action:?A>)
         => (#EvaluateAction <action ?A> <intent execute>)
[734-#]: (#EvaluateAction <action ?A> <intent execute>)
         => ($Confirm <confirmation ?A>)
[735-S]: [00239] = (Action <name present> <target (00229)> <focus EnumeratedSet:second>)
[736-M]: ($Confirm <confirmation (00239)>)
[737-D]: ‘’Engineer ‘engineer8’ is shown on the screen and highlighted.’’
[738-H]: "This history?"
[739-S]: [00240] = (QueryParameter <content this> <parameter this:history>)
[740-M]: (@Query <query (00240)>)
[741-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
```

Table C2 Continued

```
[742-#]: (#EvaluateQuery <query ?A>) => ($Answer <content ?B>)
[743-S]: [00241] = (Proximity <reference camp0> <distance near>)
[744-S]: [00242] = (Action <name build> <target 'training camp'> <location (00241)> )
[745-S]: [00243] = (ActionSequence <first (00022)> <second (00122)> <third (00242)> )
[746-S]: [00244] = (Object <name engineer8> <history (00243)>)
[747-M]: ($Answer <content (00244)>)
[748-A]: >> "First, this engineer built a training camp in the northeast."
[749-A]: >> "Then, it built the research lab just north of the vault"
[750-A]: >> "Finally, it built another training camp near the first."
[751-H]: "How many are left?"
[752-S]: [00245] = (Subset <restriction remaining>)
[753-S]: [00246] = (QueryParameter
            <content (00245)>
            <parameter Subset:cardinality> )
[754-M]: (@Query <query (00246)>)
[755-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[756-#]: (#EvaluateQuery <query ?A>) => ($Query <query ?B>)
[757-S]: [00247] = (MissingParameter
            <content (00246)>
            <parameter content:Subset:superset> )
[758-M]: ++($Query <query (00034)>)++
[759-R]: {  ($Query <query (QueryParameter <content ?A> <parameter ?B>)>),
            (%ParameterMatchInFocus <content ?A> <parameter ?B>) -> ?C }
         => {  (%AddParameter <original ?A> <parameter ?B> <value ?C>) -> ?D ),
               (%GetIntent <content ?D>) -> ?E,
               (^HandleResponse <content ?D> <intent ?E>),
               --($Query <query (QueryParameter <content ?A> <parameter ?B>)>)-- }
[760-R]: (^HandleResponse <content ?A> <intent query>)
         => (@Query <query ?A>)
[761-S]: [00248] = (Subset
            <superset (00229)>
            <restriction remaining> )
[762-S]: [00249] = (QueryParameter
            <content (00248)>
            <parameter Subset:cardinality> )
[763-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[764-#]: (#EvaluateQuery <query ?A>) => ($Answer <content ?B>)
[765-S]: [00250] = (Subset
            <superset (00229)>
            <restriction remaining>
            <cardinality 1> )
[766-M]: ($Answer <content (00250)>)
[767-A]: >> "One more."
[768-H]: "Show me"
[769-S]: [00251] = (Action <name present>)
[770-M]: (@Order <orders (00251)>)
[771-R]: (@Order <orders Action:?A>)
         => (#EvaluateAction <action ?A> <intent execute>)
[772-#]: (#EvaluateAction <action ?A> <intent execute>)
         => ($Query <query ?B>)
[773-S]: [00252] = (MissingParameter
            <content (00251)>
            <parameter Action:target> )
[774-M]: ++($Query <query (00034)>)++
[775-R]: {  ($Query <query (QueryParameter <content ?A> <parameter ?B>)>),
            (%ParameterMatchInFocus <content ?A> <parameter ?B>) -> ?C }
         => {  (%AddParameter <original ?A> <parameter ?B> <value ?C>) -> ?D ),
               (%GetIntent <content ?D>) -> ?E,
               (^HandleResponse <content ?D> <intent ?E>),
               --($Query <query (QueryParameter <content ?A> <parameter ?B>)>)-- }
[776-R]: (^HandleResponse <content ?A> <intent execute>)
         => (@Order <orders ?A>)
[777-S]: [00253] = (Action
            <name present>
            <target (00229):third> )
[778-R]: (@Order <orders Action:?A>)
         => (#EvaluateAction <action ?A> <intent execute>)
```

Table C2 Continued

```
[779-#]: (#EvaluateAction <action ?A> <intent execute>)
         => {  ($Confirm <confirmation ?B>),
               ($Tell <content ?C>) }
[780-S]: [00254] = (Action <name present> <target (00229)> <focus EnumeratedSet:third>)
[781-M]: ($Confirm <confirmation (00254)>)
[782-D]: ''Engineer 'engineer9' is shown on the screen and highlighted.''
[783-S]: [00255] = (Proximity <reference camp1> <distance near>)
[784-S]: [00256] = (Action <name build> <target 'training camp'> <location (00255)>)
[785-S]: [00257] = (ActionSequence <first (00027)> <second (00256)>)
[786-M]: ($Tell <content (00257)>)
[787-A]: >> "This engineer built a training camp in the south."
[788-A]: >> "Then, this engineer built another training camp near the first."
[789-H]: "How many engineers are left?"
[790-S]: [00258] = (Subset
            <superset (Reference <reference engineers>)>
            <restriction remaining> )
[791-S]: [00259] = (QueryParameter
            <content (00258)>
            <parameter Set:cardinality> )
[792-M]: (@Query <query (00259)>)
[793-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[794-#]: (#EvaluateQuery <query ?A>) => ($Query <query ?B>)
[795-S]: [00260] = (MissingParameter
            <content (00259)>
            <parameter content:Subset:superset> )
[796-M]: ++($Query <query (00034)>)++
[797-R]: {  ($Query <query (QueryParameter <content ?A> <parameter ?B>)>),
            (%ParameterMatchInFocus <content ?A> <parameter ?B>) -> ?C }
         => {  (%AddParameter <original ?A> <parameter ?B> <value ?C>) -> ?D ),
               (%GetIntent <content ?D>) -> ?E,
               (^HandleResponse <content ?D> <intent ?E>),
               --($Query <query (QueryParameter <content ?A> <parameter ?B>)>)-- }
[798-R]: (^HandleResponse <content ?A> <intent query>) => (@Query <query ?A>)
[799-S]: [00261] = (Subset <superset (00229)> <restriction remaining> )
[800-S]: [00262] = (QueryParameter
            <content (00261)>
            <parameter Subset:cardinality> )
[801-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[802-#]: (#EvaluateQuery <query ?A>) => ($Answer <content ?B>)
[803-S]: [00263] = (Subset
            <superset (00229)>
            <restriction remaining>
            <cardinality 0> )
[804-M]: ($Answer <content (00263)>)
[805-A]: >> "You have seen all of the engineers that are standing."
[806-H]: "No, I mean in the game."
[807-S]: [00264] = (CompareRelation <relationship in> <reference2 game>)
[808-S]: [00265] = (Meaning <meaning (00264)>)
[809-M]: (Conjunction
            <first (@Answer <polarity negative>)>
            <second (@Assert <content (00265)>)> )
[810-R]: (Conjunction <first ?A> <second ?B>)
         => {  (?A):resolve#->?C,
               (?B):resolve#->?D,
               (^ResolveConjunction <first ?C> <second ?D>) }
[811-M]: (@Answer <polarity negative>)
[812-R]: {  (@Answer <polarity negative>),
            [$:?B] }
         => (@Reject <rejection ?B>)
[813-R]: (@Reject <rejection ($Answer <content ?A>)>)
         => (@Reject <rejection ?A>) }
[814-S]: [00266] = (@Reject <rejection (00263)>)
[815-R]: (@Reject <rejection ?A>) => (#EvaluateRejection <rejection ?A>)
[816-M]: (@Assert <content (00265)>)
[817-R]: {  (@Assert <content (Meaning <meaning ?A> !<content ?B>)>),
            [@:?C] }
         => (#ApplyMeaning <content ?C> <meaning ?A>)
```

Table C2 Continued

```
[818-M]: (^ResolveConjunction
            <first (#EvaluateRejection <rejection (00263)>)>
            <second (#ApplyMeaning
               <content (@Query <query (00259)>))>
               <meaning (00264)>)> )
[819-R]: {  (^ResolveConjunction
               <first (#EvaluateRejection <rejection ?A>)>
               <second (#ApplyMeaning <content ?B> <meaning ?C>)> ),
            (%GetGenerator <content ?A>) -> ?D,
            (= <first ?B> <second ?D> ) }
         => {  (%GetIntent <content ?B>) -> ?E,
               (%Merge <first ?B> <second ?C>) -> ?F,
               (^HandleReinterpretation <content ?F> <intent ?E>) }
[820-S]: [00267] = (Subset <superset engineers> <restriction (00264)>)
[821-S]: [00268] = (Subset <superset (00278)> <restriction remaining>)
[822-S]: [00269] = (QueryParameter <content (00268)> <parameter Subset:cardinality>)
[823-R]: (^HandleReinterpretation <content ?A> <intent query>)
         => (@Query <query ?A>)
[824-M]: (@Query <query (00269)>)
[825-R]: (@Query <query ?A>) => (#EvaluateQuery <query ?A>)
[826-#]: (#EvaluateQuery <query ?A>)
         => ($Answer <content ?B>)
[827-S]: [00270] = (CompareRelation
            <relationship in>
            <reference1 thing>
            <reference2 game>)
[828-S]: [00271] = (Subset
            <superset engineers>
            <restriction (00264)>
            <cardinality 23>)
[829-M]: ($Answer <content (00271)>)
[830-A]: >> “There are 23 engineers alive."
```

REFERENCES

[1] James Allen, Donna Byron, Myroslava Dzikovska, George Ferguson, Lucian Galescu, and Amanda Stent. "An architecture for a generic dialogue shell." *Natural Language Engineering*, 6(3-4):213-228, December 2000.

[2] James Allen, George Ferguson, and Amanda Stent. "An architecture for more realistic conversational systems." In *Proceedings of the 6th International Conference on Intelligent User Interfaces*, Sante Fe, New Mexico, U.S.A., January 2001, pp. 1-8.

[3] Salah Aly. "Protocol Verification and Analysis Using Colored Petri Nets." *Technical Report, DePaul University, TR04-003.* July 2003.

[4] AUML: Agent Unified Markup Language Website. http://www.auml.org/

[5] J. L. Austin. *How to Do Things with Words.* Harvard University Press, Cambridge, Massachusetts, U.S.A.

[6] R. Bindiganavale, W. Schuler, J. Allbeck, N. Badler, A. Joshi, and M. Palmer. "Dynamically Altering Agent Behaviors Using Natural Language Instructions." *4th International Conference on Autonomous Agents*. Pg. 293-300, 2000.

[7] Nate Blaylock, John Dowding, and James Allen. "A dialogue model for interaction with planers, schedulers and executives." In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, Houston, Texas, U.S.A., October 27-29, 2002.

[8] D. G. Bobrow, R. M. Kaplan, M. Kay, D. A. Norman, H. Thomspon, and T. Winograd. "Gus, a frame driven dialog system." *Artificial Intelligence* 8:155-173.

[9] Gregor v. Bochmann and Alexandre Petrenko. "Protocol Testing: Review of Methods and Relevance for Software Testing." Universite de Montreal.

[10] J. M. Bradshaw et al. KAoS: Toward an industrial-strength open agent architecture. In J.M. Bradshaw, editor, *Software Agents*. AAAI/MIT Press, 1997.

[11] Robbert L. Brak, Jacques D. Fleuriot and Jarred McGinnis. "Theorem proving for protocol languages." In *Proceedings of the Second European Workshop on Multi-Agent Systems*. Barcelona, Spain 2004.

[12] L. Braubach, A. Pokahr, D. Moldt, and W. Lamersdorf. "Goal Representation for BDI Agent Systems." In *Proceedings of The Second International Workshop on Programming Multiagent Systems*. PROMAS-2004 at ACTIONMAS04.

[13] J. Carletta, A. Isard, S. Isard, J. Kowtko, and G. Doherty-Sneddon. "HCRC Dialogue Structure Coding Manual." Technical Report HCRC/TR-82.

[14] A. Chella, M. Cossentino, L. Sabatucci and V. Seidita. "Agile PASSI: An Agile Process for Designing Agents." *International Journal of Computer Systems Science & Engineering. Special issue on 'Software Engineering for Multi-Agent Systems.'* May 2006.

[15] Herbert H. Clark, and Susan E. Brennen. "Grounding in Communication." In L. B. Reskick, J. M. Levine, and S. D. Teasley (Editors), Perspectives on Socially Shared Cognition (pp. 127-149). Washington, DC, U.S.A.: American Psychological Association, 1991.

[16] Philip R. Cohen and Hector J. Levesque. "Intention is choice with commitment." *Artificial Intelligence*, 42(2-3):213-292, 1990.

[17] Philip R. Cohen, and Hector J. Levesque. "Communicative Actions for Artificial Agents." In J.M. Bradshaw (Editor), Software Agents (pp. 419-436). Cambridge, Massachusetts U.S.A.: MIT Press.

[18] Philip R. Cohen, and Hector J. Levesque. "ACL for Teamwork." In *Nous*. Vol. 25, No. 4. (pp. 487-512) 1991.

[19] Robin Cohen, Michael Y. K. Cheng, and Michael W. Fleming. "Why bother about bother: Is it worth it to ask the user?" In *Proceedings of the AAAI Fall Symposium Series on Mixed Initiative Problem Solving Assistants*, Arlington, Virginia, U.S.A., November 3-6, 2005.

[20] R. A. Cole, J. Mariani, H. Uszkoreit, A. Zaenen and V. Zue. "Survey of the State of the Art in Human Language Technology." Center for Spoken Language Understanding CSLU, Canegie Mellon University, Pittsburgh, Pennsylvania, U.S.A., 1995.

[21] M. G. Core, and James F. Allen. "Coding Dialogues with the DAMSL Annotation Scheme." In *Working Notes of the AAAI Fall Symposium on Communicative Action in Humans and Machines*. 1997.

[22] N. Dalbäck, and A. Jönsson. "A Coding Manual for the Linköping Dialogue Model." 1998. Linköping University.

[23] D. J. Duke, P. J. Barnard, D. A. Duce, I. Herman, and J. May. "Human-Computer Protocols." In *Proceedings of the Workshop on Continuity in Human Computer Interaction*, Scheveningen, The Netherlands, April 2-3, 2000.

[24] Carlos A. Estombelo Montesco, and Dilvan de Abreu Moreira. "UCL – Universal Communication Language." *Journal of Research on Computer Science*, ISSN 1665-9899.

[25] B. Eugenio. "An Action Representation Formalism to Interpret Natural Language Instructions."

[26] Jacques Ferber. "Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence." Addison-Wesley Professional, 1999.

[27] FIPA: Foundation for Intelligent Physical Agents. http://www.fipa.org/

[28] FIPA ACL: Agent Communication Language. http://www.fipa.org/specs/fipa00061/

[29] Galaxy Communicator Homepage. http://communicator.sourceforge.net/

[30] M. Georgeff, B. Pell, M. Pollack, M. Tambe and M. Woolridge. "The Belief-Desire-Intention Model of Agency." In Jörg Müller, Munindar P. Singh and Anand S. Rao (Editors), Proceedings of the 5th International Workshop on Intelligent Agents {V} : Agent Theories, Architectures, and Languages ({ATAL}-98). 1555:1-10, Springer-Verlag Publishers, Heidelberg, Germany, 1999.

[31] Göran Goldkuhl. "Action and Media in Interorganizational Interaction. Coordinating the Role of IT with Business Processes." *Communications of the ACM*. Vol. 49, No. 5. May 2006.

[32] Barbara J. Grosz. "The structure of task oriented dialogues." In *Proceedings of the IEEE Symposium on Speech Recognition*. Pittsburgh, Pennsylvania, U.S.A., April 1974.

[33] Barbara J. Grosz, and Candace L. Sidner. "Attention, intentions, and the structure of discourse." *Computational Linguistics*. 12(3):175-204. July-September 1986. MIT Press, Cambridge, Massachusetts, U.S.A.

[34] Gerald J. Holzmann. "Protocol Design: Redefining The State Of The 'Art'." *IEEE Software*. Vol. 9, No. 1. (pp. 17-22). January 1992.

[35] G. Juhas, R. Lorenz and T. Singilar. "Petri Net Semantics." In *Proceedings of the 24th International Conference on Application and Theory of Petri Nets*. Eindhoven, The Netherlands, June 23-27, 2003.

[36] Yves Kodratoff, Adrian Dimulescu and Ahmed Amrani. "Man-Machine Cooperation in Retrieving Knowledge from Technical Tests." In *Proceedings of the AAAI Fall Symposium Series on Mixed Initiative Problem Solving Assistants*, Arlington, Virginia, U.S.A., November 3-6, 2005.

[37] John R. Lee, and Andrew B. Williams. "Behavior Development through Natural Language Discourse." *Computer Animation and Virtual Worlds*, Special Issue: The Very Best Papers from CASA 2004, 15(3-4):327-337.

[38] John R. Lee, and Andrew B. Williams. "Towards a Theoretical Framework for the Integration of Dialogue Models into Human-Agent Interaction." In *Proceedings of the AAAI Fall Symposium Series on Mixed Initiative Problem Solving Assistants*, Arlington, Virginia, U.S.A., November 3-6, 2005.

[39] Jürgen Lind. "Specifying Agent Interaction Protocols with Standard UML." In *Proceedings of the Second International Workshop on Agent-Oriented Software Engineering*. Montreal Canada, May 29, 2001.

[40] Olivia C. March. *Natural Language as an Agent Communication Language*. Honours thesis, Department of Computer Science and Software Engineering, University of Melbourne.

[41] Michael F. McTear. "Spoken Dialogue Technology: Enabling the Conversational User Interface." In *ACM Computing Surveys*, 2002.

[42] Steve Miller. *A Model Theoretic Approach to the Design and Verification of Distributed Systems*. Ph.D. Thesis, University of Iowa. Iowa City, Iowa, U.S.A., 1992

[43] John Mylopoulos, Jaelson Castro and Manuel Kolp. "Tropos: Towards Agent-Oriented Information Systems Engineering." *The Second International Bi-Conference Workshop on Agent-Oriented Information Systems, AOIS2000*. Stockholm, Sweden, June 5-6, 2000.

[44] Nursebot Project: Robotic Assistants for the Elderly. http://www-2.cs.cmu.edu/~nursebot/

[45] OAA: Open Agent Architecture. http://www.ai.sri.com/~oaa/

[46] Lin Padgham and Michael Winikoff. "Prometheus: A Practical Agent-Oriented Methodology." In B. Henderson-Sellers and P. Giorgini (Editors), Agent-Oriented Methodologies. Idea Group, 2005.

[47] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, second edition, 2003.

[48] David Sadek and R. De Mori. "Dialogue systems." In R. De Mori (Editor) *Spoken Dialogue with Computers*, Academic Press, 1998.

[49] David Sadek. "Design Considerations on Dialogue Systems: From Theory to Technology, The Case of Artimis." In *Proceedings of the ESCA Workshop on Interactive Dialogue in Multi-Modal Systems*, Kloster Irsee, (pp. 173-188), 1999.

[50] John R. Searle. *Expression and Meaning: Studies in the Theory of Speech Acts*. Cambridge University Press, 1979.

[51] Candice L. Sidner. "An artificial discourse language for collaborative negotiation." In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, MIT Press, Cambridge, Massachusetts, U.S.A., 1:814-819, 1994.

[52] Candice L. Sidner. "Building Spoken Language Collaborative Interface Agents." *Technical Report, Mitsubishi Electric Research Laboratories, TR2002-038*. August 2002.

[53] Ira A. Smith, Philip R. Cohen, Jeffery M. Bradshaw, Mark Greaves and Heather Holmback. "Designing conversational policies using joint intention theory." In *Proceedings of the International Conference on Multi Agent Systems*, Paris, France, (pp. 269-276), July 3-7 1998.

[54] I. A. Smith and P. R. Cohen. "Toward a semantics for an agent communication language based on speech-acts." In *Proceedings of the National Conference on Artificial Intelligence*. AAAI Press, 1996.

[55] Stratagus, A Real Time Strategy Game. SourceForge Website. http://stratagus.sourceforge.net/

[56] David R. Traum, and Elizabeth A. Hinkelman. "Conversation acts in task-oriented spoken dialogue." *Computational Intelligence* 8(3):575-599. 1992. Special Issue on Non-literal language.

[57] David R. Traum, and James F. Allen. "Discourse Obligations in Dialogue Processing." In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, Las Cruces, New Mexico, U.S.A., (pp. 1-8), 1994.

[58] David R. Traum. "Coding Schemas for Spoken Dialogue Structure." 1996 Unpublished manuscript.

[59] Richard S. Wallace. "Don't Read Me: A. L. I. C. E. and AIML Documentation." http://alicebot.org/articles/dont.html

[60] Peter Wegner. "Why Interaction is More Powerful Than Algorithms." *Communications of the ACM*. Vol. 40, No. 5. May 1997.

[61] J. Weizenbaum. "ELIZA – A computer program for the study of natural language communication between man and machine." *Communications of the ACM*, 9(1):36-45.

[62] Andrew B. Williams, George Thomas, Michael Grassi, and John R. Lee. "IDOCS: Collaboratively Describing and Classifying the Features of AMD with Machine Learning." Invest Ophthalmol Vis Sci, ARVO, Fort Lauderdale, Florida, U.S.A., April 30-May 4, 2006.

[63] F. Zambonelli, N. R. Jennings and M. Woolridge. "Multiagent systems as computational organizations: the Gaia methodology." In B. Henderson-Sellers and P. Giorgini (Editors), Agent-Oriented Methodologies. 136-171, Idea Group Publishers, 2005.