Chapter 1

THE INFORMATION STATE APPROACH TO DIALOGUE MANAGEMENT

David R. Traum
University of Southern California Institute for Creative Technologies
13274 Fiji Way, Marina del Rey California 90292 USA
traum@ict.usc.edu

Staffan Larsson
Department of Linguistics, Gothenburg University, Box 200
SE-405 30 Göteborg, Sweden
sl@ling.gu.se

Abstract We introduce the information state approach to dialogue management, and show how it can be used to formalize theories of dialogue in a manner suitable for easy implementation. We also show how this approach can lead to better engineering of dialogue management components of dialogue systems, allowing for separate development of modular system fundamentals, dialogue theories, and domain-specific dialogue systems, in a manner where components can more easily be reused. TrindiKit is a tool instantiating the lowest level, and allowing straightforward implementation of dialogue theories formalized using the information state approach. We briefly describe several dialogue systems built using TrindiKit, and how components have been successfully further developed and reused in other projects.

Keywords: Dialogue Management, information state, TrindiKit, GoDiS, EDIS

1. Introduction

There are currently many quite different approaches to dialogue management in existing and proposed dialogue systems. It is difficult to compare these approaches, especially divorced from the systems and specific tasks for which these systems have been designed. A crucial issue for

progress in dialogue management is to be able to evaluate dialogue management approaches, bringing theoretical approaches closer to practical implementations, and having a better sense of the appropriate starting point for a new dialogue system: how much (of both theory and implementation) can be carried over from previous efforts, and what might need to be added/changed. This article describes an approach toward these issues, along with some development aids for producing dialogue management in dialogue systems, as developed in the TRINDI project.[1]

A first issue is a lack of universal agreement about what dialogue management *is*. We can trivially define dialogue management as the functions performed by a *dialogue manager* component of a dialogue system, but this, if anything, makes the situation even worse, since different systems break up the division of functions to software components very differently. Some systems include in a dialogue manager component more or less than others of such functions as contextual interpretation, domain reasoning and action, message routing, and natural language generation. Other systems, e.g., (Allen *et al.*, 2001; Blaylock *et al.*, 2002) have no component called a *dialogue manager*, assigning dialogue management functions to other modules. To be clear, we define dialogue management as the following functions within a dialogue system:

1 updating the dialogue context on the basis of interpreted communication (both that produced by the system and by other communicating agents, be they human "user" or other software agent)

2 providing context-dependent expectations for interpretation of observed signals as communicative behavior

3 interfacing with task/domain processing (e.g., database, planner, execution module, other back-end system), to coordinate dialogue and non-dialogue behavior and reasoning

4 deciding what content to express next and when to express it

Some of these functions are naturally closely-related to other functions, and so, e.g., one might want to have the same software component that manages the context updates actually perform context-based interpretation, alleviating the need for communications of expectations on one side, and interpretations on the other. In this case, from our point of view, we would say that this module (whether it is called a "Dialogue Manager" or some other name) is performing both dialogue management and other dialogue system functions (e.g., interpretation). Many of these decisions about which functions to allocate to which components (and

how many different components there should be) are decided purely locally to the development of a specific dialogue system for a particular domain and task. Sometimes these decisions are based on the data-flow for a particular task, and may yield efficiency and simplicity gains for a particular system, but often these decisions are made for external reasons, such as the expertise of particular team members, or the availability of existing software tools. Unfortunately, this somewhat arbitrary decision on assigning functions to software modules creates problems for reuse of components. The efficiency-based determinations for one system don't necessarily carry over to a next system, often requiring a more radical redesign for a new system than might be necessary with other boundaries. Also, there has not been much in the way of support for dialogue management reuse, since dialogue management has often been performed as part of system-specific and domain-specific modules.

We propose two contributions toward solving the problem of dialogue management re-use. First, a unifying view of dialogue management, that can help organize the relationship between dialogue theories and implementations, and secondly, software tools that can help to achieve reusable dialogue systems. The unifying view includes a proposal to formalize dialogue management functions in terms of *information state* update. Key to this approach is identifying the relevant aspects of information in dialogue, how they are updated, and how updating processes are controlled. This simple view can be used to compare a range of approaches and specific theories of dialogue management within the same framework (as well as facilitating hybrid approaches).

The term INFORMATION STATE of a dialogue represents the information necessary to distinguish it from other dialogues, representing the cumulative additions from previous actions in the dialogue, and motivating future action. For example, statements generally add propositional information; questions generally provide motivation for others to provide specific statements. Information state is also referred to by similar names, such as "conversational score", "discourse context" or "mental state". Generally, although not necessarily, we will also talk about the information state of participants of the dialogue, representing the information that those participants have at a particular point in the dialogue – what they brought with them to the dialogue, what they pick up, and how they are motivated to act in the (near) future.

In the next section, we present the information state approach, which allows specific theories of dialogue to be formalized, implemented, tested, compared, and iteratively reformulated. Key to this approach will be a notion of UPDATE of information state, with most updates related to the observation and performance of DIALOGUE MOVES. In Section 3, we

describe how the information state approach can be used to help provide reusable components for dialogue system design, separating out a basic software engineering layer, a dialogue theory layer, and a task/domain specific layer. We follow this with sections describing the bottom two layers: first, in Section 4, we describe TrindiKit, a tool that provides the basic software engineering glue that can be used to implement a dialogue manager at a level closer to linguistic theories than other existing toolkits. In Section 5, we illustrate some of the systems that have been built using TrindiKit to implement different theories of dialogue. Finally, in Section 6, we describe how the separation of architecture layers described in Section 3 has led to actual reuse in a number of dialogue systems beyond those described in Section 5.

2. The Information State Approach

Just as dialogue systems are largely incommensurable, so are dialogue theories. Often, when comparing different theories, functionally similar concepts are given different names, or the same name is used for quite different concepts. It can also be difficult to tease apart the relative contributions of the underlying formal tools vs the specific aspects of the dialogue theory. Moreover, there is often quite a gap between theories of dialogue that linguists or philosophers of language might devise and the theories directly implemented in dialogue systems. Dialogue systems can provide a great testbed for theories of dialogue, since they can straightforwardly manifest behavior of an implemented theory as the dialogue progresses, however, this is true only in so far as the system incorporates an accurate representation of the theory. To help in this regard, we present a method of specifying a dialogue theory that makes it straightforward to implement, and, as described in the following sections, tools to help implement a dialogue theory specified along these lines.

We view an information state-based theory of dialogue as consisting of:

- a description of the **informational components**, including aspects of common context as well as internal motivating factors (e.g., participants, common ground, linguistic and intentional structure, obligations and commitments, beliefs, intentions, user models, etc.).
- **formal representations** of the above components (e.g., as lists, sets, typed feature structures, records, Discourse Representation Structures (DRSs), propositions or modal operators within a logic, etc.).

- a set of **dialogue moves** that will trigger the update of the information state. These will generally also be correlated with externally performed actions, such as particular natural language utterances. A complete theory of dialogue behavior will also require rules for recognizing and realizing the performance of these moves, e.g., with traditional speech and natural language understanding and generation systems.
- a set of **update rules**, that govern the updating of the information state, given various conditions of the current information state and performed dialogue moves, including (in the case of participating in a dialogue rather than just monitoring one) a set of selection rules, that license choosing a particular dialogue move (or set of dialogue moves) to perform given conditions of the current information state.
- an **update strategy** for deciding which rule(s) to apply at a given point, from the set of applicable ones. This strategy can range from something as simple as "pick the first rule that applies" to more sophisticated arbitration mechanisms, based on game theory, utility theory, or statistical methods.

It is important to distinguish information state approaches to dialogue modeling from other, structural, dialogue state approaches. These latter approaches conceive a "legal" dialogue as behaving according to some grammar, with the states representing the results of performing a dialogue move in some previous state, and each state licensing a set of allowable next dialogue moves. The "information" is thus implicit in the state itself and the relationship it plays to other states. It may be difficult to transform an information state view to a dialogue state view, since there is no necessary finiteness restriction on information states (depending on the type of information modeled), and the motivations for update and picking a next dialogue move (using update rules, and update strategy) may rely on only a part of the information available, rather than the whole state. On the other hand, it is very easy to model dialogue state as information state: the information is the dialogue state, itself. This is easily modeled as a register indicating the state number (for finite state models, or a stack for recursive transition networks). The dialogue moves will be the same moves that are used in the dialogue state theory, the update rules will be the transitions in the dialogue state theory, formulated as an update to a new state, given the previous state and performance of the action, and the update strategy will be much the same as in the transition network (i.e., deterministic or non-deterministic, etc.)

Structural dialogue state approaches have often been contrasted with plan-based approaches to dialogue modeling (e.g., by (Cohen, 1996; Sadek & De Mori, 1998)). Structure-based approaches are usually viewed as viable for simple, scripted dialogues, while plan-based approaches, though more complex and difficult to embed in practical dialogue systems, are seen as more amenable to flexible dialogue behavior. Plan-based approaches are also criticized as being more opaque, especially given the large amount of procedural processing and lack of a well-founded semantics for plan-related operations. An information-state approach allows one to fruitfully combine the two approaches, using the advantages of each. The information state may include aspects of dialogue state as well as more mentalistic notions such as beliefs, intentions, plans, etc. Moreover, casting the updates in terms of update rules and strategies that apply the rules under appropriate conditions provides for a more transparent, declarative representation of system behavior than most procedural programs, rendering the resulting dialogue manager easily amenable to experimentation with different dialogue strategies.

In the rest of this section, we will present the aspects of information state in a little more detail. To keep a degree of concreteness, we will make reference to an example theory of information state developed by Cooper and Larsson, described in more detail in (Cooper *et al.*, 1999; Traum *et al.*, 1999; Bohlin *et al.*, 1999).

2.1 Informational Components

Information state is usually not conceived of as a monolithic node in a transition network (as with dialogue state), but rather as consisting of several interacting components. There is a wide range of possibilities as to what kinds of components should be used to model dialogue. The first choice point comes as to whether to model the participants' internal state, or more external aspects of the dialogue. There are also many ways of modeling the internal state of a participant. One can choose to model the mental state of the agent (attitudes such as belief, desire, intention, along with social correlates such as mutual belief, joint intention, and obligation) (e.g., (Bretier & Sadek, 1996; Traum & Allen, 1994)), or one can take a more structural view of the dialogue, concentrating on the performance of actions and various sorts of accessibility relationships. (e.g., (Ahrenberg *et al.*, 1990)). It may also be useful to distinguish components of information state into *static* and *dynamic* aspects. The former are those aspects of information state that are not expected to change during the course of a dialogue, but are still very useful for modeling the progression of the dialogue. Examples of static

information state components could include things like domain knowledge, or knowledge of dialogue conventions. It depends on the type of dialogue being modeled, and the scope of the conversation as to which aspects will be assumed to be static vs. dynamic (e.g., contrasting a knowledge acquisition system vs. a question answering system – the former may want to treat domain knowledge as dynamic, while the latter would see it as static). Marking some information as static may have some advantages in efficient implementation, since various compilation shortcuts and efficient memory allocation techniques could be performed. It is still good practice to have declarative knowledge sources rather than implicit knowledge in program (or finite state automaton) control structure, to be able to reuse the same knowledge for different dialogue situations.

Our example information state is a simplified version of the dialogue game board which has been proposed by Ginzburg (Ginzburg, 1996a; Ginzburg, 1996b; Ginzburg, 1998). There is some information assumed to be private (including *beliefs*, and an *agenda* of actions to perform in the dialogue) and some that is assumed to be shared (propositions assumed to be shared *beliefs*, questions under discussion (*QUD*), and the latest dialogue move performed (*lm*)). This small set of informational elements was used to track the behavior of participants in information seeking dialogues, including asking and answering (potentially elliptical) questions and accumulating information (Cooper *et al.*, 1999; Poesio *et al.*, 1999).

2.2 Formal Representations

Given a choice of what aspects of the dialogue structure and the participants' internal state to model, the question then arises as to *how* to model them. There are a wide number of choices, from simple abstract data types, to more complex informational systems, such as logics (with associated inference systems) and statistical systems of various flavors. These choices will be related to the particular theory of accessibility of these elements, and will also affect other processing issues, such as comprehensiveness and efficiency. As an example, consider an aspect of information state such as actions to be performed in the dialogue (e.g., an agenda, plan, or other such bundle of intentions). There's a choice as to whether to represent tokens separately (e.g., with some sort of list) or just types, not distinguishing between multiple tokens of the same type (e.g., with a set). Given a choice of representing a list, there is still the question of accessibility – should it be a FIFO queue, a LIFO stack, or some more open structure, allowing access to the whole list? Likewise,

if an agent's beliefs are represented, should this be a set, some sort of ordered list, or a complete logical inference system, in which implicit beliefs are also said to hold given some configuration of explicit beliefs?

$$(1)\quad \left[\begin{array}{lcl} \textsc{private} & : & \left[\begin{array}{lcl} \textsc{bel} & : & \textsc{Set}(\textsc{Prop}) \\ \textsc{agenda} & : & \textsc{Stack}(\textsc{Action}) \end{array}\right] \\ \textsc{shared} & : & \left[\begin{array}{lcl} \textsc{bel} & : & \textsc{Set}(\textsc{Prop}) \\ \textsc{qud} & : & \textsc{Stack}(\textsc{Question}) \\ \textsc{lm} & : & \textsc{Move} \end{array}\right] \end{array}\right]$$

Our example information state is represented as a record (Cooper & Larsson, 1999), as shown in (1). Here private and shared information are represented as sub-records, each with several fields. Each field is either a value, a set or a stack, with the type of information (proposition, action, question or move) indicated.

2.3 Dialogue Moves

Dialogue moves are meant to serve as an abstraction between the large number of different possible messages that can be sent (especially in a natural language) and the types of update to be made on the basis of performed utterances. Dialogue moves can also provide an abstract level for content generation. As with information components, there are also a number of dialogue move taxonomies to choose from; some principles regarding this issue are outlined in (Traum, 2000). There must be at least sufficient types of dialogue moves to provide the different kinds of updates desired. The set of dialogue moves to choose is also influenced by the task of language interpretation – how easy will it be to (reliably) determine that one move vs. another has been performed? Another complicating issue is how to capture the inherent multi-functionality of utterances – with complex moves and move taxonomies, where each move has multiple functions, or with a set of more simple moves, one per function, in which case a single utterance will embody multiple moves. Dialogue moves are often conceived of as *speech-acts* in the sense of (Searle, 1969), but this is not a necessity, dialogue moves could be any mediating input, e.g., logical forms or even word-lattices augmented with likelihoods.

Our example information state theory uses only two moves, **ask** and **answer**.

2.4 Update Rules

Update rules formalize the way that information state is changed as the dialogue progresses. Each rule consists of a set of applicability conditions and a set of effects. The applicability conditions specify aspects

of the information state that must be present for the rule to be appropriate. Effects are changes that will be made to the information state when the rule has been applied (assuming that all conditions hold). Update rules are meant to encapsulate coherent bundles of change to the information state, given a particular theory of dialogue. While atomic conditions and effects are built from the set of possible operations on an abstract datatype, update rules specialize these operations further to be specific (potentially complex) building blocks of a dialogue theory.

Continuing our example information state theory, the rule for adding a question to QUD if an *ask* move has been performed is shown in (2). This rule has two conditions: that the latest move was of type *ask*, and that the top of the agenda was the action of raising a question, the effects are to pop this item from the agenda, and push onto QUD the question that is the content of both the *raise* agenda item and the *ask* dialogue move. Other update rules in our sample information state theory include rules to select an answer move on the basis of the top of the agenda (3), to integrate an answer (possibly elliptical) from the user relevant to the topmost question on QUD (4), and to remove the question from the QUD if a proposition resolving the question is in the shared beliefs (5).

(2)
U-RULE: **integrateSysAsk**
PRE: { val(SHARED.LM, ask(usr,Q))
fst(PRIVATE.AGENDA, raise(Q)) }
EFF: { push(SHARED.QUD, Q)
pop(PRIVATE.AGENDA) }

(3)
U-RULE: **selectAsk**
PRE: { fst(PRIVATE.AGENDA, raise(Q)) }
EFF: { set(NEXT_MOVE, ask(Q)) }

(4)
U-RULE: **integrateUserAnswer**
PRE: { val(SHARED.LM, answer(usr,A)),
fst(SHARED.QUD, Q)
DOMAIN :: relevant(A, Q)
DOMAIN :: reduce(Q, A, P) }
EFF: { add(SHARED.BEL, P) }

(5)
U-RULE: **downdateQUD**
PRE: { fst(SHARED.QUD, Q)
in(SHARED.BEL, P)
DOMAIN :: resolves(P, Q) }
EFF: { pop(SHARED.QUD) }

2.5 Update Strategy

Along with the set of update rules, a strategy for how to apply the rules is needed. This is, in many cases, going to be crucial for the design

of the rules themselves. Given a particular update strategy, one may need to make adaptations to the rules, and perhaps also aspects of the information state itself, in order to guarantee a particular sequence of rule applications. There's also a question of whether to have separate strategies for choosing different types of update rules (i.e., for observation and selection of dialogue moves), and whether these processes can be ordered or asynchronously applied. Some of the types of update strategies we have considered include:

1 Take the first rule that applies (iteratively until no rules apply)

2 Apply each rule (if applicable) in sequence

3 Apply rules according to class

4 Choose among applicable rules using probabilistic information

5 Present choices to user to decide (for development modes)

For our sample information state, we use algorithm 1, above.

2.6 Discussion

We have presented a five component model of the formalization of dialogue theories as information state update. There is a synergy between choices for the components of information state: the conceptual notions, formal representations, dialogue moves, update rules, and update strategy. A complete theory of dialogue update will need to include a smoothly interacting combination of these aspects. However, it is still very possible to hold some of them constant while trying out different possibilities for the others. E.g., different formal representations for a concept, different rules for doing updates, or different update strategies for applying rules.

Given the sample informational components described above, we can track information states in simple information-seeking dialogue. A short question-answer exchange is illustrated in Figure 1.1. Before the exchange, the system has an agenda item to raise the question about the user's destination. This meets the conditions for update rule (3). After the system utterance, the update algorithm will apply rule (2). After the user utterance, rule (4) will check that the answer matches the question topmost on QUD, and if so, will integrate the answer into the shared beliefs, and then rule (5) will pop the question off the QUD.

$$\begin{bmatrix} \text{PRIVATE} = \begin{bmatrix} \text{BEL} = \{\} \\ \text{AGENDA} = \langle \text{raise}(?x.\text{dest-city}(x)),\ \text{raise}(\ldots),\ \ldots \rangle \end{bmatrix} \\ \text{SHARED} = \begin{bmatrix} \text{BEL} = \{\} \\ \text{QUD} = \langle\rangle \\ \text{LM} = \ldots \end{bmatrix} \end{bmatrix}$$

U-RULE: **selectAsk**

EFF: { set(NEXT_MOVE, ask(?x.dest-city(x)))

`Sys: Where do you want to go?`

U-RULE: **integrateSysAsk**

EFF: { push(SHARED.QUD, ?x.dest-city(x))
pop(PRIVATE.AGENDA)

$$\begin{bmatrix} \text{PRIVATE} = \begin{bmatrix} \text{BEL} = \{\} \\ \text{AGENDA} = \langle raise(?\text{x}.depart-city(\text{x})), \ldots \rangle \end{bmatrix} \\ \text{SHARED} = \begin{bmatrix} \text{BEL} = \{\} \\ \text{QUD} = \langle\ ?x.\text{dest-city}(x) \rangle \\ \text{LM} = \text{ask}(?x.\text{dest-city}(x)) \end{bmatrix} \end{bmatrix}$$

`Usr: Malvern`

U-RULE: **integrateUsrAnswer**

EFF: { add(SHARED.BEL, dest-city(malvern))

U-RULE: **downdateQUD**

EFF: { pop(SHARED.QUD)

$$\begin{bmatrix} \text{PRIVATE} = \begin{bmatrix} \text{BEL} = \{\} \\ \text{AGENDA} = \langle\ \text{raise}(?x.\text{depart-city}(x)),\ \ldots \rangle \end{bmatrix} \\ \text{SHARED} = \begin{bmatrix} \text{BEL} = \{\text{dest-city(malvern)}\} \\ \text{QUD} = \langle\rangle \\ \text{LM} = \text{answer(malvern)} \end{bmatrix} \end{bmatrix}$$

Figure 1.1. Example Dialogue and associated information state and updates

3. A multi-level architecture for reusable dialogue management

Building theoretically adequate dialogue managers for specific dialogue tasks is a difficult endeavor because multiple types of expertise are required. At a basic level, one must handle the tasks involved with managing a complex software system, generally involving multiple components, inter-component communication, and complex data manipulation. Another requirement is expertise in the dialogue theory itself and intricacies of how it should be applied in different circumstances. Finally, there is the issue of engineering a generic system to a particular task and domain, requiring domain expertise and domain-related processing (e.g.,

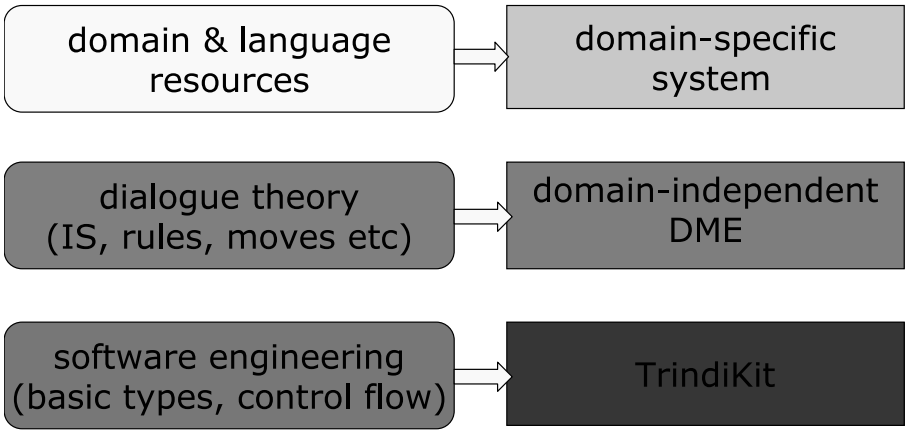


Figure 1.2. Multi-level architecture

database creation and maintenance). It is rare that a single individual has sufficient expertise in all three areas to quickly build a state of the art dialogue system. Thus, separating these three levels into different software components can be a big win, allowing individuals to concentrate on only a subset of the issues, more easily collaborating to build a complete system. Moreover, separating these levels into distinct components can make it easier to reuse appropriate parts in multiple systems and allow parallel development. Figure 1.2 shows our multi-level architecture for dialogue management.

At the lowest level of the architecture shown in Figure 1.2, the issues are those of software engineering to support easy implementation of dialogue theories. As opposed to those toolkits built for impoverished theoretical constructs, such as finite state dialogue transition networks (Sutton & Cole, 1998), we provide tools for directly implementing a dialogue theory formalized using the information state approach outlined in the previous section. As described in the next section, these tools include: extensible abstract data types for implementing the information components, methods for specifying dialogue moves, writing update rules and strategies, and running the system in conjunction with other modules (e.g., language interpretation and generation) of a dialogue system.

Section 5 describes some of the implemented dialogue systems that have been built using this toolkit, implementing different dialogue theories formulated as information state. Parts or all of these systems have

been successfully carried over to different domains and tasks, by the uses of different resources and, where necessary, adapting the theory for specific genres of dialogue, as discussed in Section 6.

4. TrindiKit: A Dialogue Move Engine Toolkit

Developing a dialogue theory using the information-state approach described in the previous section yields a computational theory of dialogue that naturally lends itself to implementation. We call the implementation of such a theory of dialogue dynamics a *Dialogue Move Engine* (DME), since its main functions are updating information state based on the observance of moves and selecting moves to be performed. This DME, together with some connective material, forms the dialogue management and discourse tracking aspects of a dialogue system. A complete dialogue system would need modules (or sets of modules) to perform at least the following additional functions:

- user interface – to receive input from and present output to the user.
- interpretation – to calculate from the input which dialogue moves have been performed, adding these to a special *latest move* part of the information state.
- generation – to take the contents of the special *next move* part of the information state, and produce the output.
- control – to wire together the other modules, either serially or in parallel.

As part of the TRINDI project and its sequel SIRIDUS, and described more fully in (Larsson *et al.*, 1999), we have developed a DME toolkit called TrindiKit, which provides the basic architecture as well as facilities for implementing theories of information state. The general architecture of TrindiKit is shown schematically in Figure 1.3. Dialogue management is handled by the control module, the DME and the information state.[2] The DME can consist of one or multiple UPDATE MODULES, each containing a different set of rules and potentially a different algorithm. Typically the DME contains an update module and a selection module, encapsulating the functions of integrating observed dialogue moves, and selecting new ones for the system to say. Under an architecture of this sort, it is up to the control module as to how to interleave these two functions.

The components of the architecture are the the total information state (TIS), consisting of the information state proper (IS), as well as interface

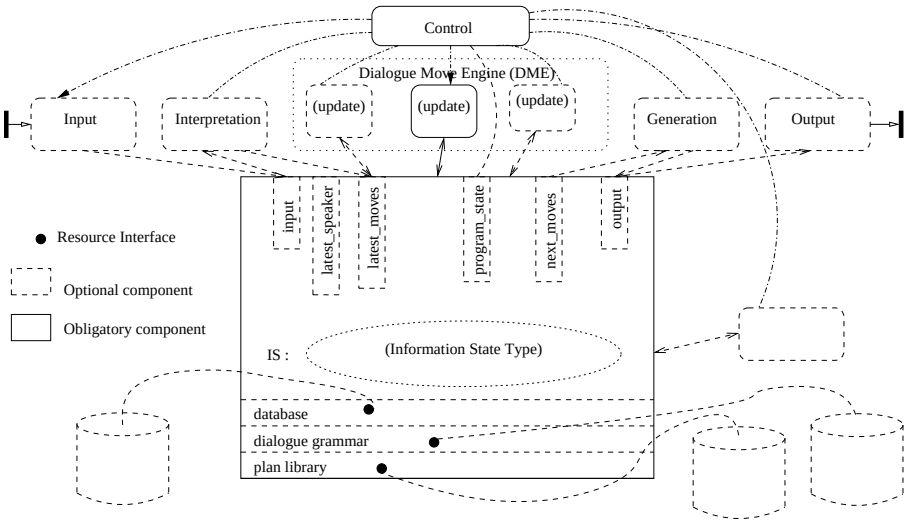


Figure 1.3. The TRINDI DME Architecture

variables for communicating with language processing modules and non-linguistic resources; the Dialogue Move Engine, consisting of one or more DME modules; other dialogue system modules (DME-external); and a control module, wiring together the other modules, either in sequence or through some asynchronous mechanism. The IS is specified using abstract data types, each permitting a specific set of queries to inspect the type and operations to change it. These are the building blocks of update rules, which can be used by other modules to inspect and change the information state in coherent ways.

Some of the components are obligatory, and others are optional or user-defined. To build a system, one must minimally supply an information state type, at least one DME module – consisting of TIS update rules and an algorithm, and a control module, operating according to a control algorithm. Any useful system is also likely to need additional modules, e.g. for getting input from the user, interpreting this input, generating system utterances, and providing output for the user. The other modules will generally communicate with the rest of the system through designated interface variables. Other resources, such as databases, plan libraries, etc., can also be integrated into the system, using an interface that allows the same kinds of queries and operations as for the IS proper, allowing update rules to be oblivious as to whether the components are part of the information state or external resources. Note that the spe-

cific set of modules shown in Figure 1.3 is just an example. TrindiKit provides methods for defining any number of both DME-modules and DME-external modules, with associated interface variables.

Apart from the general architecture defined above, TrindiKit provides definitions of datatypes (for use in TIS variable definitions), a language and format for specifying TIS update rules, methods for accessing the TIS, an algorithm definition language for DME and control modules, default modules for input, interpretation, generation and output, methods for converting items from one type to another, methods for visually inspecting the TIS, and debugging facilities. TrindiKit does not itself specify any particular theory of dialogue - instead, it provides the building blocks and infrastructure for implementing different dialogue theories in terms of IS type, update rules, dialogue moves, etc.

Starting with version 2.0, TrindiKit has been adapted to allow multiple means of interacting with OAA (Open Agent Architecture, (Martin *et al.*, 1999)). The simplest technique is to run a TrindiKit-based system as an OAA agent. It is also possible to run TrindiKit modules and resources as OAA agents, and one can even configure TrindiKit to use OAA as its internal communication protocol. This allows the use of a wide selection of existing software together with TrindiKit. It also makes it easier to write TrindiKit system components in other programming languages such as Java, C, and C++, even though the TrindiKit implementation is currently written in Sicstus Prolog.

Work on TrindiKit continues as part of the SIRIDUS project[3]. TrindiKit 3.0 (Larsson *et al.*, 2002), to be released in December 2002, will feature, among other things, improved methods for accessing the information state, input and output modules for several common speech recognizers and synthesizers (e.g. Nuance), improved debugging facilities, and a graphical user interface.

5. Implementations using TrindiKit

A number of systems have been developed using TrindiKit, implementing dialogue managers for different theories of dialogue, using the information state approach. We will look at two of them in some detail: GoDiS, developed at Gothenburg University by Cooper, Larsson and Bohlin (Bohlin *et al.*, 1999), which uses an extension of the information state theory used as an example in Section 2, and the EDIS system (Matheson *et al.*, 2000), developed at University of Edinburgh, which uses a notion of information state based on (Poesio & Traum, 1998). More details of these and other TrindiKit systems can be found in (Bos *et al.*, 1999).

5.1 GoDiS

GoDiS is an experimental dialogue system built using TrindiKit, which is being used to explore and develop a theory of issue-based dialogue management (Larsson, 2002). It uses fairly simple algorithms for control, update and selection modules, keyword-based interpretation and template-based generation. The notion of information state used in GoDiS is an extension of that illustrated in Section 2, and is currently able to handle multiple simultaneous topics, inquiry-oriented and action-oriented dialogue, grounding, and *question accommodation*, which, among other things, allows users to answer unasked but contextually salient questions. The GoDiS system currently distinguishes 6 "core" dialogue move types: *ask, answer, request, confirm, greet* and *quit*. In addition, it distinguishes an extensive set of dialogue moves related to grounding (Traum, 1994).

The main division in the information state is between information that is PRIVATE to the system and that which is assumed to be SHARED between the dialogue participants. What we mean by shared information here is that which has been explicitly established during the conversation.[4]

The SHARED field is divided into several subfields. The first subfield (COM) is a set of propositions which the system assumes that the dialogue participants are jointly committed to. The second subfield, ISSUES, represents all questions which have been raised in a dialogue (explicitly or implicitly) but not yet resolved. It thus contains a collection of current, or "live" issues. The data structure used is an open stack, i.e. a stack where non-topmost elements can be accessed. This allows a non-rigid modeling of current issues and task-related dialogue structure. For handling action-oriented dialogue, an additional field containing requested but not yet completed actions is included. The data structure used is the same as for global issues, for the same reasons. Another subfield contains a more local stack of questions under discussion (QUD), which can be used to resolve elliptical answers. The final two SHARED sub-fields, PU and LU contain information about the previous and latest utterances, respectively. This information includes speaker and dialogue moves (type and content).

The PRIVATE field contains five subfields. The BEL field contains propositions that the system holds to be true. The AGENDA field contains the system's short term intentions for the next turn. The PLAN field is a list of actions that are longer-term dialogue goals. This plan can, however, be changed during the course of the conversation. We also have a field TMP that mirrors the shared fields. This field keeps track of

the shared information prior to the integration of the latest utterance. This makes it easy to delete information which the agent has optimistically assumed to have become shared if it should turn out that the other dialogue participant does not perceive, understand or accept it. If a low reliability score is assigned to an utterance, the system will use a *cautious* rather than *optimistic* strategy with respect to grounding. The final private subfield is a queue of as yet non-integrated moves (NIM), possibly including moves performed in utterances previous to the latest one.

The update rules in GoDiS include an ability to perform several kinds of *accommodation* when the appropriate expected structures needed to fully integrate a new utterance are not on ISSUES, QUD or PLAN. In real human-human dialogues, dialogue participants can address questions that have not been explicitly raised in the dialogue. However, the question is still needed, to allow interpretation of elliptical utterances. An example from a travel agency dialogue[5] Is shown in (6).

(6) J: what month do you want to go

P: well around 3rd 4th april / some time there

P: as cheap as possible

The strategy we adopt for interpreting elliptical utterances is to think of them as short answers to questions on QUD (in the sense of (Ginzburg, 1996a; Ginzburg, 1996b; Ginzburg, 1998)). A suitable question here is *What class (economy, business, or first class) does P want for the ticket?*. This question is not under discussion at the point when the customer says "as cheap as possible", but it can be inferred from the context. In fact J will have as part of his plan the action of raising this question. On our analysis it is this fact which enables J to interpret the ellipsis. He finds the matching question on his plan, accommodates by placing it on QUD and then continues with the integration of the information expressed by *as cheap as possible* as normal.

A similar situation arises when there is no current plan to use as context for interpretation. For example, if a travel agent discovers that his customer wishes to get information about a flight he will adopt a plan to ask her where she wants to go, when she wants to go, what price class she wants and so on. In cases where the customer does not state her errand explicitly, but rather answers some question(s) (e.g. about destination and means of transport), the agent must infer what the task is. We call this process *task accommodation*, and it is closely related to question accommodation.

A sample dialogue with GoDiS (as "Sys") is shown in (7).

(7) Sys: Welcome to the travel agency!
Usr: flights to paris
Sys: Okay, you want to know about price. A flight. To Paris. Let's see. What city do you want to go from?
Usr: london
Sys: Okay, from London. When do you want to travel?
Usr: in april, as cheap as possible
Sys: Okay, in April, economy class. What day do you want to travel?
Usr: the fifth
Sys: Okay. The price is $300

The first user utterance ("flights to paris") is interpreted by the system as realizing two answer-moves, potentially resolving two questions (means of transport and destination city). However, no task or plan has yet been established and no questions have been raised. To be able to integrate the utterance, the system must find a task (in the form of an issue) associated with a plan which includes the raising of questions which match the answers given. Once this issue (in this case the issue of the price of a trip) has been accommodated and the plan entered into the PLAN field, the questions matching the user's answers can be accommodated and the answers integrated. As a consequence of this process, the information state now contains a plan which guides the system behavior. The system explicitly indicates acceptance (Okay), accommodation (you want to know about price), integration (A flight. To Paris.)[6], loading a new plan (Let's see.), and then proceeds to ask the next question in the plan (What city do you want to go from). The resulting information state is shown (partially) in (9).

(8)
$$\left[\begin{array}{lcl} & & \left[\begin{array}{lcl} \text{BEL} & = & \{\} \\ \text{AGENDA} & = & \langle\rangle \end{array}\right] \\ \text{PRIVATE} & = & \left[\begin{array}{lcl} \text{PLAN} & = & \left\langle\begin{array}{l} \text{findout(?x.dept-month(x))}, \\ \text{findout(?x.dept-day(x))}, \\ \text{findout(?x.class(x))}, \\ \text{consultDB(?x.price(x))} \end{array}\right\rangle \\ \text{TMP} & = & \ldots \\ \text{NIM} & = & \ldots \end{array}\right] \\ \text{SHARED} & = & \left[\begin{array}{lcl} \text{COM} & = & \{\text{dest-city(paris),how(plane)}\} \\ \text{ISSUES} & = & \langle ?x.\text{dept-city}(x),\ ?x.\text{price}(x)\ \rangle \\ \text{QUD} & = & \langle ?x.\text{dept-city}(x)\rangle \\ \text{PU} & = & \ldots \\ \text{LU} & = & \langle \text{ask(sys, } ?x.\text{dept-city}(x)), \ldots\rangle \end{array}\right] \end{array}\right]$$

Given the kind of information state illustrated in (8), we can provide update rules which accommodate questions. A formalization of

the **accommodateQuestion** move is given in (9)[7]. When interpreting the latest utterance by the other participant, the system makes the assumption that it was an **answer** move with content A. This assumption requires accommodating some question Q such that A is a relevant answer to Q. The condition "relevant(A, Q)" is true if A is a relevant answer to Q given the current information state, according to some (possibly domain-dependent) definition of question-answer relevance.

(9) U-RULE: **accommodateQuestion**(Q,A)

PRE: { in(SHARED.LU, answer(usr,A)), in(PRIVATE.PLAN, findout(Q)), domain :: relevant(A, Q)

EFF: { del(PRIVATE.PLAN, findout(Q)), push(SHARED.QUD, Q)

GoDiS uses an update algorithm where different types of rules are applied at different stages of the update process. There are currently 7 rule types and 49 rules in the most complex version of the system:

- *grounding*: handles optimistic or cautious grounding (1 rule)
- *integrate*: integrates the effects of the latest move (19 rules)
- *accommodate*: handles question (and task) accommodation (7 rules)
- *downdate_issues*: removes resolved issues and actions (4 rules)
- *downdate_qud*: removes questions from QUD (1 rule)
- *load_plan*: loads plans for dealing with issues and actions (2 rules)
- *exec_plan*: executes dialogue plans (8 rules)
- in addition, there are 7 rules for miscellaneous purposes, with no class assigned

A short outline of the update algorithm goes as follows: First, any dialogue moves that result from interpretation are incorporated into the NIM queue in the information state proper. The system then goes into an "integration loop" where it tries to integrate the effects of these moves in the order they were performed, while keeping track of new and obsolete issues, and possibly loading a plan for dealing with some new issue. If any moves are still left after this loop, accommodation is tried. If any accommodation rule succeeds, the integration loop is entered again to see if accommodation has enabled any remaining moves to be integrated. This procedure is repeated until no moves can be integrated and no accommodation rules can trigger. After this, the current plan is executed,

which typically leads to one or several actions being put on the agenda. Finally, questions which are judged to be no longer available for ellipsis resolution are removed from QUD.

The basic strategy for the selection algorithm is that not more than one issue should be raised by each system utterance. The selection algorithm first checks if some question-raising action is already on the agenda; if not, it tries to select a new action. After this, it selects dialogue moves (including grounding moves) based on the actions on the agenda repeatedly until no more moves can be selected.

The current control algorithm in GoDiS simply calls each module in turn in a serial fashion.

GoDiS has been adapted to several domains including travel agency, autoroute and VCR interface, and there are lexicon resources for both English and Swedish. GoDiS is also being used in teaching and student projects, including adaptations to domains such as cinema ticket booking, handheld computer agenda, and mobile phone interface.

5.2 EDIS

The EDIS system (Matheson *et al.*, 2000), uses a notion of information state based on (Poesio & Traum, 1997; Poesio & Traum, 1998), using the record representation used for coding information states in (Cooper *et al.*, 1999; Poesio *et al.*, 1999). Like GoDiS, the informational components of EDIS consist of a common ground part, a semi-public part, and a private part. The common part includes four types of information: obligations of dialogue participants to perform actions (OBL), social commitments that participants have that propositions hold (SCP), a dialogue history of acts that have been performed (DH), and conditional statements that will establish obligations or commitments, given the performance of appropriately typed dialogue acts (COND). The semi-public part, analogous to TMP in GoDiS, is a collection of *discourse units* (DUs) (Traum & Hinkelman, 1992), which represent coherent bundles of information that are *grounded* (added to the common ground (Clark & Schaefer, 1987)) together. Private information includes the intentions of the agent being modeled. The formal representations of EDIS are shown in (10), where PT-R is a record containing the type of information contained in common ground, shown to the right. G represents the common ground. Two DU pointers are represented, CDU for current, and PDU, the previous one. UDUs is a list of the DUs (which may include the ones identified by PDU and/or CDU) that are not grounded.

(10)
$$\begin{bmatrix} \text{G} & : & PT\text{-}R \\ \text{CDU} & : & \begin{bmatrix} \text{C} & : & PT\text{-}R \\ \text{ID} & : & \text{DU-ID} \end{bmatrix} \\ \text{PDU} & : & \begin{bmatrix} \text{C} & : & PT\text{-}R \\ \text{ID} & : & \text{DU-ID} \end{bmatrix} \\ \text{UDUs} & : & \text{List(DU-ID)} \\ \text{INT} & : & \text{List(Action)} \end{bmatrix} \qquad PT\text{-}R : \begin{bmatrix} \text{DH} & : & \text{List(Action)} \\ \text{OBL} & : & \text{List(Action)} \\ \text{SCP} & : & \text{List(Prop)} \\ \text{COND} & : & \text{List(Action)} \end{bmatrix}$$

EDIS uses a modified version of the dialog acts from the DRI coding scheme (Discourse Resource Initiative, 1997), giving them precise effect conditions according to aspects of the information state in (10). A summary of the main effects of dialogue acts is shown in (11). Act observations are represented as an ID, a confidence level (1 for partially understood, 2 for well understood), and an act schema, (including a discourse participant (DP) as speaker and other content fields depending on the act: either previous dialogue acts, propositions, questions, acts (not necessarily dialogue acts), or discourse units), For each act observation the effects on the information state are shown. Several shorthand functions are used: o(DP) means the other dialogue participant in the interaction, Q(ID) and P(ID) mean the question and proposition part (respectively) of the content of dialogue act ID.

(11)

act	ID:2, **accept**(DP,ID2)
effect	*accomplished via rule resolution*
act	ID:2, **ack**(DP,DU1)
effect	peRec(G,DU1.C)
effect	remove(DU1,UDUS)
act	ID:2, **agree**(DP,ID2)
effect	push(SCP,**scp**(DP,**P**(ID2)))
act	ID:2, **answer**(DP,ID2,ID3)
effect	push(SCP,**ans**(DP,**Q**(ID2),**P**(ID2)))
act	ID:2, **assert**(DP,PROP)
effect	push(SCP,**scp**(DP,PROP))
effect	push(COND,**accept**(**o**(DP),ID)→ **scp**(**o**(DP),PROP))
act	ID:1, **assert**(DP,PROP)
effect	push(COND,**accept**(**o**(DP),ID)→ **scp**(**o**(DP),PROP))
act	ID:2, **check**(DP,PROP)
effect	push(OBL,**address**(**o**(DP),ID))
effect	push(COND,**agree**(**o**(DP),ID) → **scp**(DP,PROP))
act	ID:2, **direct**(DP,Act)
effect	push(OBL,**address**(**o**(DP),ID))
effect	push(COND,**accept**(**o**(DP),ID) → **obl**(**o**(DP),Act))
act	ID:2, **info_request**(DP,Q)
effect	push(OBL,**address**(**o**(DP),ID))

EDIS uses the same general pipelining of modules as GoDiS, however the update algorithm is a bit different. Whenever a set of dialogue acts

are placed in `latest_moves`, the algorithm in (12) is applied, where each step includes the application of a set of update rules.

(12) a. Create a new DU and push it on top of UDUs (and point CDU to this one, while updating the PDU pointer to the old value of CDU).

b. Perform updates on the basis of backwards grounding acts, such as merging the contents of PDU.C into G for an `acknowledgement`.

c. If any other type of act is observed, record it in the dialogue history in CDU and apply the update rules for this kind of act (invoking the effects shown in (11)).

d. Apply update rules to all parts of the IS which contain newly added acts.

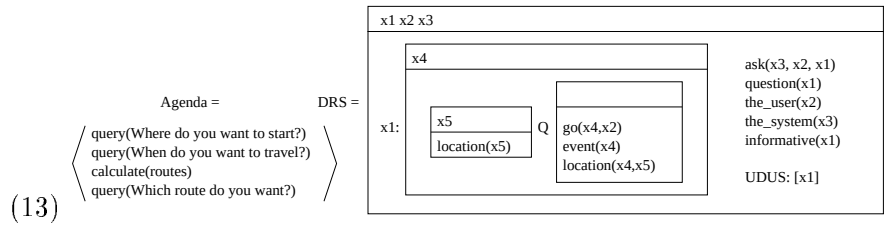
There is also a deliberation step, applied for each system turn, which leads to the system developing new intentions on the basis of obligations, potential obligations that would result from conditions (in the COND field of G or CDU) if an intended act were performed, as well as insufficiently understood dialogue acts and intentions to perform complex acts. Following deliberation, dialogue acts are selected to fulfill any intentions, and placed in the `next_moves` interface variable, for the generation module to act on.

5.3 Other TrindiKit Systems

We briefly mention two other TrindiKit systems that were developed as part of the TRINDI project. More details on the theories of information state underlying these systems can be found in (Traum *et al.*, 1999), while details of the systems themselves can be found in (Bos *et al.*, 1999).

The MIDAS system uses the DRS structures of DRT (Kamp & Reyle, 1993) as a major component of its information state. As part of the root DRS will be subordinate DRSes representing events mentioned in the dialogue, as well as tracking of grounding, using a simplified version of the theory proposed in (Poesio & Traum, 1998). Multiple theorem provers are used both for pragmatic aspects of dialogue act interpretation and to implement some of the conditional tests on the update rules. An example of a MIDAS information state can be seen in (13), in which the DRS represents a question the system has asked. This is one of a set of questions, with other future questions remaining on an agenda.

(13)



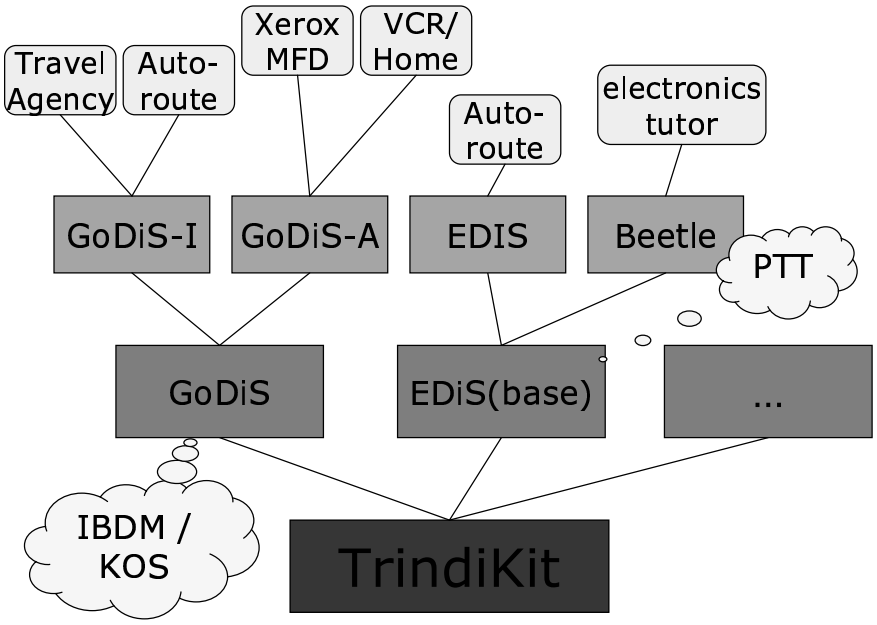
Another system focuses on conversational game theory, recasting a previous Autoroute system (Lewin, 1998) in the current framework. Conversational games are formalized as recursive transition networks. The top-level information state type is a record, as shown in (14). The top level distinguishes a dialogue participant's role as rational agent from that of conversational game-player, with the intuition that the details of the latter should not be major factors in the former. Actions and agenda-items are themselves records with multiple fields.

(14)
$$\left[\begin{array}{lcl} \textsc{rational\_agent} & : & \left[\begin{array}{lcl} \textsc{plan} & : & \textsc{Stack(Action)} \\ \textsc{scoreboard} & : & \textsc{Set(Prop)} \end{array}\right] \\ \textsc{game\_player} & : & \left[\begin{array}{lcl} \textsc{agendaitem} & : & \textsc{Stack(Agendaitem)} \\ \textsc{currtoken} & : & \textsc{Token} \\ \textsc{alltokens} & : & \textsc{Stack(Set(Prop))} \end{array}\right] \end{array}\right]$$

6. Reusing dialogue management components

The previous section illustrated that TrindiKit has been successfully used as a foundation for implementing several quite different theories of dialogue. In addition, the implementations of the dialogue theories have also successfully been re-used in the design of alternate versions of the systems to deal with different tasks and domains. Figure 6 shows a sort of "family tree" of some of the re-use of implementations over the past few years, relating systems to their components and theories, using the layers described in Section 3.

Using TrindiKit (which implements the information state approach), two different domain-independent dialogue move engines have been developed (GoDiS and EDIS(base)); the former implementing Issue-Based Dialogue Management (which was developed starting from Ginzburg's KOS framework), and the latter implementing the "Poesio-Traum Theory" (PTT), as described in the previous section. From these basic systems, genre-specific variants have been developed (more or less incrementally) for dealing with inquiry-oriented dialogue (GoDiS-I, EDIS), action-oriented dialogue (GoDiS-A), and tutorial dialogue (Beetle) (Core *et al.*, 2000). Finally, various domain-specific dialogue systems have been



developed by adding specific resources (databases, plans, lexicons) to these systems.

This demonstrates several cases of reuse of implementations. Firstly, all systems use TrindiKit to deal with low-level implementation details related to the information state approach, and many also use modules supplied with TrindiKit (e.g. for input and output). Second, the same basic system can be used in several genres, given the appropriate additions and modifications. Thirdly, the same genre-specific system can be used for several different applications given the appropriate resources. In fact, the case of GoDiS and EDIS demonstrates a further type of reuse, as some basic building blocks from GoDiS was used in the initial development of EDIS. We also anticipate that domain specific resources could be reused by systems embodying different information state based theories of dialogue, merely by adding some wrappers to access the resources according to the datatypes of the new theory.

In addition to the SIRIDUS project, which involves directly extending TrindiKit and the GoDiS system, a number of other projects aside from TRINDI have built systems using TrindiKit and the information state approach to do dialogue management for a variety of application areas, including tutoring (Core *et al.*, 2000), embodied agents (Pelachaud *et al.*, 2002), and PDA control of robots (Burke *et al.*, 2002). There

are as of this writing, over seventy registered downloaders of TrindiKit, representing a heterogeneous population of users, from those who just learned about dialogue systems in class and wanted to get some hands on experience, to those slightly modifying existing systems like GoDiS for their own purposes, to those actually using TrindiKit in new research project implementations. While we have received explicit feedback from only a small percentage of these downloaders, we can still make some tentative conclusions about the utility of TrindiKit to the general user population. The time to learn how to modify and build dialogue systems using TrindiKit is reasonably short, ("a couple of weeks", "a month or two of five-ten hours a week"). Many users said that having an example system distributed (GoDiS) was very helpful for learning how to use TrindiKit. The ability to rapidly develop prototype systems was generally confirmed and appreciated, as was the "absolute freedom in defining system behaviour". However, some users reported that implementing a new dialogue manager was not so easy. While this may be partly due to the fact that dialogue management in itself is a fairly tricky business, we are striving for further improvements regarding the usability of TrindiKit for implementing new systems, both in terms of implementation and documentation. A GUI, improved debugging facilities, and a tutorial would probably be useful additions. Some users also had problems with the fact that TrindiKit requires SICStus Prolog, which is not freeware. Generally, the use of Prolog was (unsurprisingly) beneficial to Prolog programmers; however, a long-term goal is to provide an implementation formalism which is less specific to Prolog or other programming languages.

There have also been a number of projects who, for various reasons, did not use the TrindiKit software, but used the information state approach as the guiding principle. These projects use other software as the bottom implementation layer, while maintaining the notions of information state, dialogue moves, and updates. For example, the Mission Rehearsal Project at University of Southern California's Institute for Creative Technologies includes a dialogue manager built using SOAR (Laird *et al.*, 1987), but contains at its core many of the same dialogue moves and update rules as the EDIS system, with minor syntactic variations (Traum & Rickel, 2002). Similarly the MATCH system for multimodal access to city help uses an information state design for its dialogue manager (Johnston *et al.*, 2002).

We see the prospect for continued advancement at all three levels of the conceptual architecture: tools for implementing information state dialogue managers such as new versions of TrindiKit and similar tools for implementing information state theories, better and different informa-

tion state-based theories of dialogue (in general and for specific genres), and domain theory "plug-ins" to allow existing systems to participate in new tasks. This multi-level scheme can allow for much more rapid progress in dialogue management development, since it will be possible for a development team to fruitfully focus on only a part of the entire problem, re-using other parts developed elsewhere.

Acknowledgments

This work was supported by the TRINDI (Task Oriented Instructional Dialogue) project, EU TELEMATICS APPLICATIONS Programme, Language Engineering Project LE4-8314. Other participants in the TRINDI project were instrumental in developing the ideas and systems described here. Peter Ljunglöf, Alexander Berman, and Johan Bos helped develop the TrindiKit distribution. GoDiS was developed by the second author and Ljunglöf, Robin Cooper and Elisabet Engdahl. EDIS was developed by Colin Matheson, Massimo Poesio, and the first author. The MIDAS system was developed by Bos. Ian Lewin developed the SRI conversational game theory system. In addition, all of the above and other TRINDI participants have contributed to the development of the framework presented here. As usual, all mistakes and misrepresentations are due to the authors of this paper. TrindiKit and GoDiS have been further developed as part of the SIRIDUS Project EC Project IST-1999-10516. The first author was supported during the writing of this article by the Department of the Army under contract number DAAD 19-99-D-0046. Any opinions, findings and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the Department of the Army.

Notes

1. TRINDI (Task Oriented Instructional Dialogue) EU TELEMATICS APPLICATIONS Programme, Language Engineering Project LE4-8314

2. Information State is used both to denote the components of the theoretical approach to dialogue modeling, and the specific blackboard-like system module that allows inspection and update (via update rules) of the current state.

3. SIRIDUS (Specification, Interaction and Reconfiguration in Dialogue Understanding Systems), EC Project IST-1999-10516

4. akin to the "conversational scoreboard" in (Lewis, 1979).

5. This dialogue, collected by the University of Lund, has been translated from Swedish.

6. If the user detects a system misunderstanding during this feedback phase, she may e.g. respond "no" immediately after the feedback utterance from the system, thus prompting a re-raising of the corresponding question. See Chapter 3 of (Larsson, 2002) for details.

7. The current GoDiS implementation uses a development version of TrindiKit 3.0, which means that the actual syntax of the rule as imlemented is slightly different from that shown here (using the 2.0 syntax).

References

Ahrenberg, Lars, Dahlbäck, Nils, & Jönsson, Arne. 1990. Discourse representation and discourse management for a natural language dialogue system. *In: Proceedings of the second nordic conference on text comprehension in man and machine.*

Allen, James F., Ferguson, George, & Stent, Amanda. 2001. An architecture for more realistic conversational systems. *Pages 1–8 of: Proceedings intelligent user interfaces 2001 (iui 01).*

Blaylock, Nate, Allen, James, & Ferguson, George. 2002. Synchronization in an asynchronous agent-based architecture for dialogue systems. *Pages 1–10 of: Proceedings of the 3rd sigdial workshop on discourse and dialogue.* Philadelphia: Association for Computational Linguistics.

Bohlin, Peter, Cooper, Robin, Engdahl, Elisabet, & Larsson, Staffan. 1999. Information states and dialogue move engines. *Pages 25–31 of: Proceedings of the ijcai99 workshop: Knowledge and reasoning in practical dialogue systems.*

Bos, Johan, Bohlin, Peter, Larsson, Staffan, Lewin, Ian, & Matheson, Colin. 1999. *Evaluation of the model with respect to restricted dialogue systems.* Tech. rept. Deliverable D3.2. Trindi.

Bretier, P., & Sadek, M. D. 1996. A rational agent as the kernel of a cooperative spoken dialogue system: Implementing a logical theory of interaction. *In:* Müller, J. P., Wooldridge, M. J., & Jennings, N. R. (eds), *Intelligent agents iii — proceedings of the third international workshop on agent theories, architectures, and languages (atal-96).* Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg.

Burke, Carl, Harper, Lisa, & Loehr, Dan. 2002 (June). A dialogue architecture for multimodal control of robots. *In: Proceedings*

of the international class workshop on natural, intelligent and effective interaction in multimodal dialogue systems.

Clark, Herbert H., & Schaefer, Edward F. 1987. Collaborating on contributions to conversation. *Language and cognitive processes*, **2**, 1–23.

Cohen, Phil. 1996. Dialogue modeling. *Chap. 6.3 of:* Cole, Ron, Mariani, Joseph, Uszkoreit, Hans, Zaenen, Annie, & Zue, Victor (eds), *Survey of the state of the art of human language technology.* Cambridge, MA: Cambridge University Press.

Cooper, R., Larsson, S., Matheson, C., Poesio, M., & Traum, D. 1999. *Coding instructional dialogue for information states.* Deliverable D1.1. Trindi Project.

Cooper, Robin, & Larsson, Staffan. 1999. Dialogue moves and information states. *Pages 398–400 of:* Bunt, H.C., & Thijsse, E. C. G. (eds), *Proceedings of the third international workshop on computational semantics.*

Core, M., Moore, J., & Zinn, C. 2000. Supporting constructive learning with a feedback planner. *In: Papers from the AAAI fall symposium on building dialogue systems for tutorial applications.* Technical Report FS-00-01, American Association for Articial Intelligence, 445 Burgess Drive, Menlo Park CA 94025.

Discourse Resource Initiative. 1997. *Standards for dialogue coding in natural language processing.* Report no. 167. Dagstuhl-Seminar.

Ginzburg, J. 1996a. Dynamics and the semantics of dialogue. *Pages 221–237 of:* Seligman, Jerry, & Westerståhl, Dag (eds), *Logic, language and computation, vol. 1.* CSLI Publications.

Ginzburg, J. 1996b. Interrogatives: Questions, facts and dialogue. *Pages 385–422 of:* Lappin, Shalom (ed), *The handbook of contemporary semantic theory.* Blackwell, Oxford.

Ginzburg, J. 1998. Clarifying utterances. *Pages 11–30 of:* Hulstijn, J., & Niholt, A. (eds), *Proc. of the twente workshop on the formal semantics and pragmatics of dialogues.* Universiteit Twente, Faculteit Informatica, Enschede.

Johnston, Michael, Bangalore, Srinivas, Vasireddy, Gunaranjan, Stent, Amanda, Ehlen, Patrick, Walker, Marilyn,

WHITTAKER, STEVE, & MALOOR, PREETAM. 2002. Match: An architecture for multimodal dialogue systems. *Pages 376–383 of: Proceedings of the 40th annual meeting of the association for computational linguistics (acl).*

KAMP, H., & REYLE, U. 1993. *From discourse to logic.* Dordrecht: D. Reidel.

LAIRD, J. E., NEWELL, A., & ROSENBLOOM, P. S. 1987. SOAR: an architecture for general intelligence. *Artificial intelligence*, **33**(1), 1–64.

LARSSON, STAFFAN. 2002. *Issue-based dialogue management.* Ph.D. thesis, Göteborg University.

LARSSON, STAFFAN, BOHLIN, PETER, BOS, JOHAN, & TRAUM, DAVID. 1999. *Trindikit manual.* Tech. rept. Deliverable D2.2 - Manual. Trindi.

LARSSON, STAFFAN, BERMAN, ALEXANDER, LJUNGLF, PETER, & TRAUM, DAVID. 2002. *Implemented siridus system architecture (trindikit 3.0 manual).* Tech. rept. Deliverable D6.4 - Manual. SIRIDUS.

LEWIN, IAN. 1998. *The autoroute dialogue demonstrator.* Tech. rept. CRC-073. SRI Cambridge Computer Science Research Centre.

LEWIS, DAVID K. 1979. Scorekeeping in a language game. *Journal of philosophical logic*, **8**(3), 339–359.

MARTIN, DAVID L., CHEYER, ADAM J., & MORAN, DOUGLAS B. 1999. The open agent architecture: A framework for building distributed software systems. *Applied artificial intelligence*, **13**(1-2), 91–128.

MATHESON, COLIN, POESIO, MASSIMO, & TRAUM, DAVID. 2000. Modelling grounding and discourse obligations using update rules. *In: Proceedings of the first conference of the north american chapter of the association for computational linguistics.*

PELACHAUD, CATHERINE, CAROFIGLIO, VALERIA, CAROLIS, BERADINA DE, DE ROSIS, FIORELLA, & POGGI, ISABELLA. 2002. Embodied contextual agent in information delivering application. *Pages 758–765 of: Proceedings of the first international joint conference on autonomous agents and multiagent systems.*

POESIO, M., COOPER, R., LARSSON, S., MATHESON, C., & TRAUM., D. 1999. Annotating conversations for information state update. *In:*

Proceedings of amstelogue'99 workshop on the semantics and pragmatics of dialogue.

POESIO, MASSIMO, & TRAUM, DAVID R. 1997. Conversational actions and discourse situations. *Computational intelligence*, **13**(3).

POESIO, MASSIMO, & TRAUM, DAVID R. 1998. Towards an axiomatization of dialogue acts. *Pages 207–222 of: Proceedings of twendial'98, 13th twente workshop on language technology: Formal semantics and pragmatics of dialogue.*

SADEK, DAVID, & DE MORI, RENATO. 1998. Dialogue systems. *In:* MORI, R. DE (ed), *Spoken dialogues with computers.* Academic Press.

SEARLE, JOHN R. 1969. *Speech acts.* New York: Cambridge University Press.

SUTTON, S., & COLE, R. 1998. Universal speech tools: the cslu toolkit. *Pages 3221–3224 of: Proceedings of the international conference on spoken language processing (icslp).*

TRAUM, DAVID, BOS, JOHAN, COOPER, ROBIN, LARSSON, STAFFAN, LEWIN, IAN, MATHESON, COLIN, & POESIO, MASSIMO. 1999. *A model of dialogue moves and information state revision.* Tech. rept. Deliverable D2.1. Trindi.

TRAUM, DAVID R. 1994. *A computational theory of grounding in natural language conversation.* Ph.D. thesis, Department of Computer Science, University of Rochester. Also available as TR 545, Department of Computer Science, University of Rochester.

TRAUM, DAVID R. 2000. 20 questions for dialogue act taxonomies. *Journal of semantics*, **17**(1), 7–30.

TRAUM, DAVID R., & ALLEN, JAMES F. 1994. Discourse obligations in dialogue processing. *Pages 1–8 of: Proceedings of the 32nd annual meeting of the association for computational linguistics.*

TRAUM, DAVID R., & HINKELMAN, ELIZABETH A. 1992. Conversation acts in task-oriented spoken dialogue. *Computational intelligence*, **8**(3), 575–599. Special Issue on Non-literal language.

TRAUM, DAVID R., & RICKEL, JEFF. 2002. Embodied agents for multi-party dialogue in immersive virtual worlds. *Pages 766–773 of: Proceedings of the first international joint conference on autonomous agents and multiagent systems.*