Representations of Dialogue State for Domain and Task Independent Meta-Dialogue

David R. Traum and **Carl F. Andersen**
University of Maryland
A.V. Williams Building, College Park, MD 20742 USA
{traum,cfa}@cs.umd.edu

Abstract

We propose a representation of local dialogue context motivated by the need to react appropriately to meta-dialogue, such as various sorts of corrections to the sequence of an instruction and response action. Such context includes at least the following aspects: the words and linguistic structures uttered, the domain correlates of those linguistics structures, and plans and actions in response. Each of these is needed as part of the context in order to be able to correctly interpret the range of corrections. Partitioning knowledge of dialogue structure in this way may lead to an ability to represent generic dialogue structure (e.g., in the form of axioms), which can be particularized to the domain, topic and content of the dialogue.

1 Introduction

Many simple dialogue systems are constructed in a more or less holistic fashion, not making clear differentiations between linguistic, dialogue, and domain components or reasoning, treating everything other than speech input and output as the "dialogue component". Such architectures allow shortcuts in the design process and fine-tuning to the particular anticipated task and dialogue interaction, which can speed up both system implementation time and run-time. However, the resulting systems are not particularly portable to other domains, tasks within the same domain, or even very robust t in the face of different styles of interaction in accomplishing the task. Often, where the dialogue component is concerned, all that can be carried over into the next system is the experience gained by building such a system. Toolkits for constructing scripted dialogues, such as [Sutton *et al.*, 1996] make the construction process faster, but do not address the underlying problem of partitioning dialogue knowledge from linguistic and domain knowledge in order to reuse the same dialogue strategies.

Simply partitioning the knowledge sources is also not sufficient to achieve domain-independent reusable dialogue modules. A dialogue component (in the narrow sense) must have appropriate access to both linguistic and domain knowledge sources in order to act appropriately in dialogue, in a manner similar to a more holistic system. While there will always be a certain amount of work involved in adapting a generic dialogue module to particular linguistic processing components and domain knowledge sources and manipulators, there is still some room for generic dialogue function, abstracting away from the specific representations provided by other modules. The key is being able to represent aspects of the dialogue in a suitably abstract fashion, to allow reasoning about generalities without relying on peculiarities of interfaces to linguistic and domain modules. We maintain, agreeing with [McRoy *et al.*, 1997], that it is important to keep several different kinds of representations of an utterance available as context, in order to act appropriately in the face of meta-dialogue, such as corrections, as well as to be able to give the right kind of feedback about problems in the system's ability to interpret and act appropriately.

As an example of a simple dialogue episode which can motivate the kinds of representation we propose, consider the exchange schema in (1). In order to understand and respond to [3] properly, B must at least keep some context around of [1] and [2]. The question arises, however, as to how to represent this context in a compact and useful form.

(1) [1] A: do X.
[2] B: [does something]
[3] A: no, do ...

In the next section, we quickly review various structural proposals for representation of local exchanges like (1). Then in Section 3, we consider these proposals in the light of a suite of examples of different kinds of negative feedback. This leads us, in Section 4, to propose a representation based on considering not just the utterances themselves, but other intensional information associated with the utterances. These include, for a request produced by the user of a system, the literal request, an interpreted version, still at the level of natural language description, and a domain-specific version. For the reply, this also includes both the plan leading to its performance, as well as observed feedback. These various levels provide both a source for detecting potential or actual incoherence in dialogue, as well as serving as a source of potential repair requests. In Section 5, we illustrate these levels in action in a dialogue manager for the TRAINS-96 system [Allen *et al.*, 1996]. Finally, we conclude with some observations of more general applicability of these levels.

2 Representations of Local Dialogue Structure

There have been several proposals for the kind of dialogue unit represented in (1), using structural terms like *adjacency pair* [Schegloff and Sacks, 1973], *exchange* [Sinclair and Coulthard, 1975], *game* [Severinson Eklundh, 1983, Carletta *et al.*, 1997], *IR-unit* [Ahrenberg *et al.*, 1990] and *argumentation act* [Traum and Hinkelman, 1992]. At an abstract level, we need a unit which can contain three moves or acts, as indicated in (2).

(2)

[1] Initiative: Request(Act) [Instruct]
[2] Response: Do(Act)
[3] Feedback: Eval [+ Counter-Request(Act')]

There are several ways in which this unit can be structured. In Figure 1 we show several proposed structures for this or similar units for questions. (A) shows a flat structure containing all three acts, as proposed by [Sinclair and Coulthard, 1975]. Some authors prefer to allow only binary branching units, which leads to structures (B) through (D). (B) was proposed by [Wells *et al.*, 1981] (though with the unit names Solicit-Give and Give-Acknowledge), and is also used by [McRoy *et al.*, 1997]. (C) and (D) were both proposed in [Severinson Eklundh, 1983], the former for information-seeking questions, and the latter for exam-questions. (E) shows a finite automaton which could be induced from these structures, allowing multiple rejections and counter-requests before a final acceptance.

There may be different motivations for these different types of structures, but for the present purposes, we will consider them strictly in terms of what kind of context is provided for the antecedent of the utterance [3]. In particular, what is the utterance of "no" referring to? A's initial utterance [1], B's reaction in [2], or some other construct? Structure (A) would predict a choice of [1] or [2], equally. Structures (B) and (C) would have a preference for [2] as the antecedent (with (C) allowing [1] as a dispreferred option, and (B) disallowing it), while (D) would see the unit of [1] and [2] combined as the most likely antecedent, i.e., not necessarily a rejection of [2] in and of itself, but of [1] and [2], together as the realization of the goal that inspired production of [1] (for reasons that might be due to problems with either of the utterances/actions themselves, or the coherence of the two).

3 Examples

In order to decide on which structure is most appropriate, as well as what kinds of representations are needed for the task, it will be helpful to examine a suite of instantiations of the exchange schema in (1). We draw our examples from the TRAINS-96 domain [Allen *et al.*, 1996], in which a user interacts with a dialogue system to provide routes for trains. Figure 2 illustrates an episode from this task, in which there are two trains of interest, Northstar, which is currently at Boston, and Metroliner, which started the task at Boston, but is now at Albany. Given this same context, consider the dialogues in (3) through (8).

```
(A)         Req+-Do
           /   |   \
          /    |    \
       [1]   [2]    [3]
```

```
(B)  Req-Do  Do-Eval
      /   \  /  \
     /     \/    \
   [1]    [2]    [3]
```

```
(C)       Quest-ANSW
            /  \
           /    \
        Quest  answer/ASS
         /       / \
        /       /   \
      [1]     [2]   [3]
```

```
(D)          Do-Eval
              /  \
             /    \
        Req-Do     \
        /    \      \
       /      \      \
     [1]      [2]   [3]
```

(E)

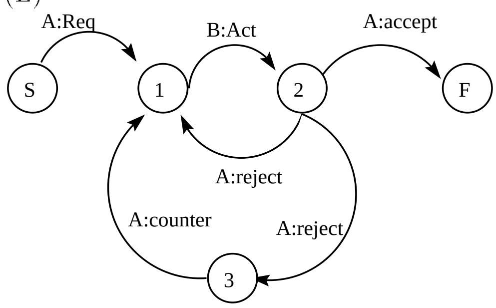

Figure 1: Proposed Structures for IRF unit

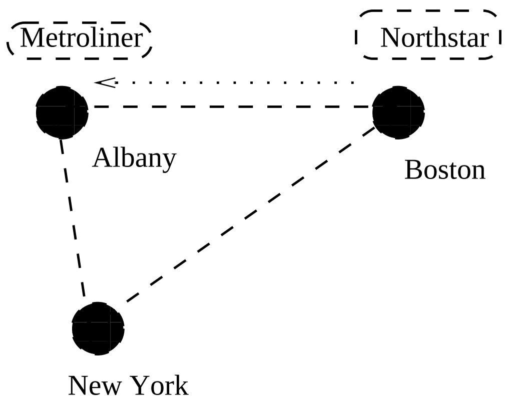

Figure 2: Trains Scenario

(3)
[1] A: “send Northstar to New York.”
[2] B: [sends Northstar to NY]
[3] A: “no, send Metroliner.”

(4)
[1] A: “send Metroliner to New York.”
[2] B: [sends Northstar to NY]
[3] A: “no, send Metroliner.”

(5)
[1] A: “send the Boston train to New York.”
[2] B: [sends Northstar to NY]
[3] A: “no, send Metroliner.”

(6)
[1] A: “send the Boston train to New York.”
[2] B: [sends Northstar to NY]
[3] A: “no, send the Boston train.”

(7)
[1] A: “send the Boston train to New York.”
[2] B: [sends Metroliner to NY]
[3] A: “no, send Metroliner.”

(8)
[1] A: “send Metroliner to New York.”
[2] B: [sends Metroliner to NY]
[3] A: “no, send Metroliner.”

(9)
[1] A: “send Northstar to New York.”
[2] B: [sends Metroliner to NY]
[3] A: “no, send Metroliner.”

In each of these, the semantic structure of utterance [3] is something like (10). Context is used to determine what “X” refers to, and also to construe “Y” to be appropriately coherent, if possible.

(10) `Don'tDo(X) & Do(Y)`

The coherence of Dialogue (3) but lack of coherence of (9) indicates a problem with (A): it seems that after [2], [1] is no longer a possible antecedent in the same way. The contrast between (3) and (4) shows that the problem with [2] can be either a lack of coherence with [1], or a change in intention. This would seem to be a problem for (B), which does not preserve [1] as part of the context for [3]. Likewise, in (5) and (6), the source of the problem is likely the interpretation of the referring expression, “the Boston train”. This is important for interpreting [3] coherently in (6), and recognizing (7) as incoherent. It is less easy to see how this information can be retrieved from (C) as opposed to (D). In general, for these examples, the source of the correction in [3] can be anywhere in the space including what A actually said in [1] (true 3rd turn repair), B's interpretation of that, or B's response in [2] (2nd turn repair). (D) seems to be the most useful candidate representation, since it provides the complex of act of [1] and [2] together as a likely antecedent for [3]. (E) captures the move sequences correctly, but does not help much with the referential dependencies.

The interesting issue for these examples is how to respond to [3] in each case. For examples (3), (4) and (5), B can just undo the action performed in [2] and proceed to do the one mentioned in (3). In fact, this is just what the Rochester TRAINS-96 system [Allen *et al.*, 1996] will do. For (6), the situation is a bit more complex. B must recognize that the previous choice of anchor for the referring expression “the Boston train” was likely to be wrong, and choose a different candidate, given A's response. For (7), (8) and (9), there is no obvious strategy to make [3] coherent, so some sort of repair would be warranted, to overcome the incoherence. In order to be able to engage in fruitful dialogue rather than just respond to a sequence of commands, the important thing to realize, is that [3] is a complex command with structure like (10), rather than unrelated `cancel` and `request` acts. Likewise, realizing a coherent interpretation when possible, and noting the incoherence, when not, is important for engaging in natural dialogues.

4 Our Approach: Internal Representations

Our approach to this problem is to represent not just the moves [1], [2], and [3], themselves, as part of the IRF unit, but also, like [McRoy *et al.*, 1997], some associated internal structures, which can help provide likely candidates for resolving any seeming incoherence. Thus, our counterpart of the Req-Do sub+structure in (D) would include not just the two acts, but each of the following components:

1. **L-req** (for “literal”, or “locutionary”) the actual words said.
2. **I-req** (for “interpreted”, “intentional” or “illocutionary”) the direct logical interpretation. This level maintains all ambiguity present in the original, such as which train is the “Boston train”.

3. **D-req** (for "disambiguated" or "domain") a precisification of the I-req that actually represents a specific request for an action that can be performed by the domain module. For simple, unambiguous requests, in which the representation output by the language module and used by the domain module are the same, D-req can be just about identical with I-req, for cases with ambiguity or divergences in representation, it may involve several operations to get from I-req to D-req. D-Req represents *what* should be done in a manner that the domain reasoner can understand.
4. **P-act** (for Plan) a specification of *how* to do the requested act in D-req. A plan suitable for execution, which, if carried out will satisfy the original request
5. **E-Act** (for execution) the action the system actually takes in fulfilling the request.
6. **O-Act** (for Observation) concerns monitoring or observation of the system's act. Even if the system performed the act correctly, it might not have evidence of this fact. For linguistic actions, this is related to *grounding* [Clark and Schaefer, 1989, Traum, 1994].

As said above, the interpretation of D-req from I-req could involve several intermediate actions. In the case of dialogues (5), (6), and (7), it would involve construction of a new query (corresponding to "which engine is the Boston engine"), acting on the basis of this calculating the answer (perhaps using messages to a domain reasoner), and then fitting this answer into the D-req for the main request (replacing the more indirect information present in the I-req) I.e., in (7), in I-req we have "train associated with Boston", but in D-req we have "Metroliner").

In the example dialogues, utterance [1] has the same L-req and I-req in (5), (6), and (7) (though different from (3) and (8)). For (5) and (6), the D-req is the same as in (3), while in (7), the D-req is the same as (8), depending on the interpretation of "the Boston train" as Northstar or Metroliner, respectively.

Using this more fine-grained notion of the Req-Do unit, we can re-examine the likely sources for the correction in [3] in each of the cases. For (3), the obvious interpretation is that there was a problem with I-req, (either A mis-spoke in [1], or changed his mind, or B misheard). For (4), B must have misheard (or somehow made a mistake in execution). For (5), the most natural interpretation is that there was a problem at the D-req level, and A meant Metroliner rather than Northstar. For (6), things are a bit more subtle. Probably the problem is the same as for (5), but less information is provided by A about the correction — B must use the information that Metroliner is probably not the correct choice when interpreting the repair. For (7), (8), or (9), the problem is most likely with P-act or E-act, or L-req (i.e., in the speech recognizer, but then with L-req for [1] or [3]?) or some unresolvable contradiction. With luck, the confusion can be cleared up using a subdialogue with the user. While it is not always crucial to identify the exact source of the problem, it is important to recognize these situations of incoherence when they occur, and not just undo the previous act and redo the very same thing.

4.1 Repair at various levels

In addition to being able to repair when faced with an unresolvable contradiction, as in dialogues (7), (8) and (9), repair is also an option whenever there is difficulty computing any of these components of the representation, or when the system is insufficiently confident of its computation. For example,

L-req: "what was the third word?"

I-req: "is Metroliner an engine?"

D-req: "which train did you mean when you said 'Boston train'?"

P-act: "is going through Albany an appropriate way to send Metroliner to NY?"

E-Act: "should I do that now or after I send Bullet through?"

O-Act: "is it there now?"

5 Implementation

We have begun to implement this approach to dialogue representation within a new dialogue manager, using Active Logic [Gurney *et al.*, 1997, Elgot-Drapkin and Perlis, 1990]. The dialogue manager and reasoner are relatively domain and system independent, relying however on translation actions to convert between the format used by external language and domain modules and the internal logic. The new implementation of active logic (described briefly in [Purang *et al.*, 1999]) combines logical reasoning in time with an ability to perform and monitor the progress of external actions. The goal of the project is to achieve a better degree of *conversational adequacy* [Perlis *et al.*, 1998] than current dialogue systems.

5.1 Maryland version of TRAINS-96

As our initial testbed, we are using components from the TRAINS-96 system from University of Rochester [Allen *et al.*, 1996]. The TRAINS-96 system consists of a set of heterogeneous modules communicating through a central hub using messages in KQML[Group, 1993]. This architecture is thus well suited for swapping in different components to do the same or a similar job and assessing the results. As well as the architecture itself, we have been using the parser, domain problem solver, and display modules, replacing the discourse manager component with our own dialogue manager and multi-modal generator. The functions of the modules in the Maryland version of the system are summarized in (11).

(11) **Parser:** produces interpretation of sentence input, as shown in Figure 3(A) (source for I-req).

Problem Solver: answers queries for problem state, also does planning requests (helps produce P-act from D-req).

Display Manager: shows objects on screen.

Dialogue manager: uses Active Logic to maintain a logical representation of dialog state and act appropriately to fulfill dialog obligations [Traum and Allen, 1994].

Output Manager: provides multimodal presentations of system output, including calls to display manager, printed text, and speech.

Figures 3 and 4 show some examples of the kinds of representations used in the system.

Figure 3(A) shows the input KQML message the parser will send the dialogue manager for a user input (typed or spoken) of "Send the Boston Engine to New York." From this, the L-req (1) and I-req (2) are computed directly, interpreting the information provided as a set of propositions about a Davidsonian event [Davidson, 1967] (in this case kqml15) and associated objects. Note the proposition `assoc-with`, indicating underspecification at the I-level about the identity of this engine or the exact relation with Boston.

Conversion to D-req involves reinterpreting the useful information of the I-req in the ontology of whatever domain reasoner is used for computing plans of action to satisfy the request. This may involve disambiguation techniques for underspecified information that the domain reasoner needs to have in order to compute a correct plan. In this example, it means resolving the desired meaning of `assoc-with`, and computing which engine is "the Boston engine".

In converting from I-req (2) to D-req (3), the system needs to resolve `:assoc-with` to find an actual engine for the bindings in the D-REQ. While this is a general contextual resolution process, involving at least the previous dialogue history, the visual map, and plans and expectations [Poesio, 1994], currently, reference resolution is performed solely by queries to the domain problem solver. In this case, the DM constructs a query conforming to "which engine is at Boston", sending a KQML message, with reply of "answer is Northstar". These two messages are added to the `assoc_msgs` for kqml15.

Once a complete D-req is built, this is used to construct a query to the problem solver to find a way to send the engine to its destination. The resulting reply message is used (along with the D-req) to construct the P-act level. A decision to execute the plan along with messages to Problem solver and output manager completes the E-act level. The O-act level is completed with responses from these modules indicating successful performance. Not shown in this level is an O-act correlate of "act4", which would come as a result of seeing this communication successfully grounded by positive feedback from the user.

```
(A) Input: Parser Message
(TELL
  :CONTENT
   (SA-REQUEST
     :FOCUS :V11621
     :OBJECTS
      ((:DESCRIPTION (:STATUS :NAME) (:VAR :V11573)
          (:CLASS :CITY) (:LEX :BOSTON) (:SORT :INDIVIDUAL))
       (:DESCRIPTION (:STATUS :DEFINITE) (:VAR :V11584)
          (:CLASS :ENGINE) (:SORT :INDIVIDUAL)
          (:CONSTRAINT (:ASSOC-WITH :V11584 :V11573)))
       (:DESCRIPTION (:STATUS :NAME) (:VAR :V11621)
       (:CLASS :CITY) (:LEX :NEW˙YORK) (:SORT :INDIVIDUAL)))
     :PATHS ((:PATH (:VAR :V11613)
               (:CONSTRAINT (:TO :V11613 :V11621))))
     :DEFS NIL
     :SEMANTICS
       (:PROP (:VAR :V11560) (:CLASS :MOVE)
         (:CONSTRAINT
         (:AND (:LSUBJ :V11560 :*YOU*) (:LOBJ :V11560 :V11584)
               (:LCOMP :V11560 :V11613))))
     :NOISE NIL
     :SOCIAL-CONTEXT NIL
     :RELIABILITY 100
     :MODE KEYBOARD
     :SYNTAX ((:SUBJECT . :*YOU*) (:OBJECT . :V11584))
     :SETTING NIL
     :INPUT (SEND THE BOSTON ENGINE TO NEW˙YORK
           PUNC-PERIOD))
     :RE 1)
```

```
(1) L-REQ
% the literal utterance

utterance(kqml15)
lreq(kqml15,[send, the, boston, engine, to, new, york, punc-period])
```

```
(2) I-REQ
% the basic logical representation of the initial
%(possibly ambiguous) utterance
ireq(kqml15, type(kqml15, sa-request))
ireq(kqml15, obj(kqml15, v11621)),
ireq(kqml15, class(v11621, city)),
ireq(kqml15, status(v11621, name)),
ireq(kqml15, lex(v11621, new˙york)),
ireq(kqml15, sort-of(v11621, individual))
ireq(kqml15, obj(kqml15, v11573)),
ireq(kqml15, sort-of(v11573, individual)),
ireq(kqml15, lex(v11573, boston)),
ireq(kqml15, class(v11573, city)),
ireq(kqml15, status(v11573, name)),
ireq(kqml15, obj(kqml15, v11584)),
ireq(kqml15, sort-of(v11584, individual)),
ireq(kqml15, class(v11584, engine)),
ireq(kqml15, status(v11584, definite)),
ireq(kqml15, assoc-with(v11584, v11573))
ireq(kqml15, path(kqml15, v11613), to(v11613, v11621))

ireq(kqml15, focus(kqml15, v11621))

ireq(kqml15, contexts(kqml15, [plan1]))
ireq(kqml15, assoc˙msgs(kqml15, [kqml15]))

ireq(kqml15, sem(kqml15, v11560))
ireq(kqml15, lf(v11560, [move, v11584, v11621]))
ireq(kqml15, class(v11560, move))
ireq(kqml15, lsubj(v11560, *you*))
ireq(kqml15, lobj(v11560, v11584))
ireq(kqml15, lcomp(v11560, v11613))
```

Figure 3: Examples of levels: L-req, I-req

5.2 Dialogue Management

The current dialogue management algorithm is very simple, consisting of a sequence of forward chaining inferences to compute the levels, one from the next, executing actions when necessary to manipulate and convert external representations and communicate with other modules. The basic outline is shown in (12).

(12) 1. input → compute L-req, I-req
2. I-req → Compute D-req (using additional queries for disambiguation, if necessary).
3. D-req → Compute P-act (usually call to problem solver for domain acts, but could be DM-internal for meta-requests)
4. P-act → Execute (E-act), (calls to problem solver and output manager)
5. Done(E-act)→ Expect feedback on O-act

```
(3) D-REQ
dreq(kqml15, at-loc(v11584, v11573))
dreq(kqml15, assoc˙msgs(kqml15, [kqml15,kqml16,kqml17]))

dreq(kqml15,
    psConstraint(kqml15, [class, v11621, city], [type, new˙york, city]))
dreq(kqml15, psConstraint(kqml15, [class, v11584, engine],
                          [type, v11584, engine]))
dreq(kqml15,
    psConstraint(kqml15, [class, v11573, city], [type, boston, city]))
dreq(kqml15, psConstraint(kqml15, [at-loc, v11584, v11573],
                          [at-loc, v11584, boston]))
dreq(kqml15,
    bindings(kqml15, [[v11573, [boston]], [v11584, [northstar]],
                      [v11621, [new˙york]]]))
dreq(kqml15, goal(kqml15,go1))
dreq(kqml15, agent(go1,northstar))
dreq(kqml15, to(go1,new˙york)))
```

```
(4) P-ACT
pact(kqml15, psstate(kqml15,pss567))
pact(kqml15, plan(kqml15,plan566))
pact(kqml15, goal(plan566,go1))
pact(kqml15, type(go1,go))
pact(kqml15, from(go1,boston))
pact(kqml15, to(go1,new˙york))
pact(kqml15, agent(plan566, northstar))
pact(kqml15, actions(plan566,[go564]))
pact(kqml15, from(go564,boston))
pact(kqml15, to(go564,new˙york))
pact(kqml15, track(go564, boston-new˙york))
pact(kqml15, status(plan566,unimplemented))
```

```
(5) E-ACT
eact(kqml15, status(plan566,implemented))
eact(kqml15, type(act4,sendmsg))
eact(kqml15, receiver(act4, outmgr))
eact(kqml15, content(act4, kqml31))
eact(kqml15, type(kqml31,move-engine-along-path))
eact(kqml15, path(kqml31,go564))
eact(kqml15, status(act4, implemented))
```

```
(6) O-ACT
oact(kqml15, status(plan566,pss-update-success))
```

Figure 4: Examples of levels: D-req, P-act, E-act, O-act

In the event of missing but necessary information, the system can construct an embedded exchange structure, in a manner similar to [Smith *et al.*, 1995]. This exchange structure can be for communication with other modules or with the user (starting from the P-level and working around to the D-level). In addition, similar actions can be undertaken when faced with contradictions, using Active Logic's capabilities for detecting and resolving contradictions [Elgot-Drapkin and Perlis, 1990]. (14) shows the rule used to trigger disambiguation of any objects not given a proper name in the I-req when converting to D-req.

```
(13) fif(and(compute_dreq(ID),ireq(ID,obj(ID,Obj)),
          ireq(ID,lex(Obj,null))),
       conclusion(dreq(ID,disambiguate(ID,Obj)))).
```

Although the algorithm presented here is very simple, the representation levels are compatible with more complex agent-oriented approaches to dialogue management, e.g., [Traum, 1996, Bretier and Sadek, 1996]. Embedding this representation in such an agent will allow choices, e.g., of whether or not to adopt an intention to execute a computed P-act.

5.3 Solving the problem

In this section we present a sketch of the resources needed to react appropriately to utterance [3] in the examples. Figure 5 shows a slightly abbreviated version of the parser message for the utterance, "No, send Metroliner to New York." The I-level for the main message will contain the predicates shown in (14), with kqml18 representing the compound act.

```
(14) ireq(kqml18, type(kqml18,compound-c-act))
     ireq(kqml18, subact(kqml18, kqml19))
     ireq(kqml18, subact(kqml18, kqml20))
     ireq(kqml18, type(kqml19, sa-reject))
     ireq(kqml18, type(kqml20, sa-request))
```

Computation of D-level for kqml19 needs to find an action to reject (identified as kqml15, using context). For kqml18 to be coherent, its sub-actions must be mutually consistent. For a request to be consistent with a rejection, (at least) E-levels must differ between the new request (kqml20) and object of rejection (kqml15). Reasoning about determinism of the plan executor, given the same situation, leads to the conclusion that P-act must be different between these two acts, as well. In this case, there is no problem, since Metroliner differs from Northstar. For the other examples, reasoning about determinism of the problem solver (or actually calling the problem solver) pushes the problem back to D-req level. This leads to recomputation of D-req (if possible/necessary), or query to user, when not (or when unable to find a sufficiently satisfactory candidate).

6 Discussion

Many systems compute something like the six levels presented here, as part of their process of engaging in dialogue. Where we are different from most researchers is in claiming the utility of keeping these levels as distinct

```
(A') Input: Parser Message
(TELL
 :CONTENT
  (COMPOUND-COMMUNICATIONS-ACT
   :ACTS
    ((SA-REJECT
      :SEMANTICS :NO
      ....
      :MODE KEYBOARD
      :SYNTAX ((:SUBJECT) (:OBJECT))
      :INPUT (NO))
     (SA-REQUEST
      :OBJECTS
       ((:DESCRIPTION (:STATUS :NAME) (:VAR :V11683)
                (:CLASS :ENGINE) (:LEX :METROLINER)
                (:SORT :INDIVIDUAL))
        (:DESCRIPTION (:STATUS :NAME) (:VAR :V11704)
                (:CLASS :CITY) (:LEX :NEW`YORK)
                (:SORT :INDIVIDUAL)))
      :PATHS ((:PATH (:VAR :V11696)
                (:CONSTRAINT (:TO :V11696 :V11704))))
      :SEMANTICS
        (:PROP (:VAR :V11676) (:CLASS :MOVE)
          (:CONSTRAINT
            (:AND (:LSUBJ :V11676 :*YOU*)
                (:LOBJ :V11676 :V11683)
                (:LCOMP :V11676 :V11696))))
       :SYNTAX ((:SUBJECT . :*YOU*) (:OBJECT . :V11683))
       :SETTING NIL
       :INPUT (SEND METROLINER TO NEW`YORK
              PUNC-PERIOD))))
 :RE 2)
```

Figure 5: Parser message for Utt. 3: "No, send Metroliner to New York"

representations for use as context in processing further utterances. Something like this is clearly necessary to deal appropriately with the examples we presented in Section 3. The Rochester system would do the same thing in each case: undo the previous action and interpret the second request in the restored context before the original request was fulfilled, with whatever train it decided upon for "the Boston Train" in (6). The ability to use the incoherence as a resource for recomputing a referential anchor or repairing is not available, nor is there an option of complaining about the seeming incoherence itself.

Keeping the I-level and D-level distinct is also important for sending appropriate messages back to the user. The I-level should be close to the linguistic structure of the user interaction, while the D-level should be close to what domain reasoners actually use. Conflating the two can lead to an inability to provide comprehensible feedback to the user. For example, the MIT Galaxy system [Seneff *et al.*, 1996] has several domain specialists, each used for a different kind of task. These domain reasoners use different ontologies, and thus, in their discourse representation (essentially the D-level), "Boston" is ambiguous between a `TOWN` in the CityGuide domain and a `CITY` in the AirTravel domain. The system may not be able to resolve which ontology object is being referred to, but surely a user not intimately familiar with the system internals would be very confused by a disambiguating query such as, "Do you mean Boston the city, or Boston the town". Fleshing this out with descriptions of the ontology types, such as "Boston the geographical area or Boston the point location" is not likely to help. Here, at the ontology of natural conversation (I-level), "Boston" is unambiguously the kind of entity that one could fly to or from, and which can contain restaurants, so any query would have to attack a different avenue for disambiguation, relating to the activities such as restaurant finding or flight booking, rather than to the kind of entity.

The approach that we are closest to, is perhaps [McRoy *et al.*, 1997], who also exploit the utility of maintaining multiple levels of representation as context. While there are some differences in the particular levels and type of structure assumed, a larger difference in approach is the uniformity of the representation language. McRoy, Haller, and Ali use a uniform approach, representing all aspects of processing in the same representation language, SNePS [Shapiro, 1979]. This does allow uniform reasoning and very powerful access to all parts of the representation, but also places limits on the kinds of language and domain subsystems that can be easily added to the system. Our approach is rather to treat the internals of the other subsystems more or less as black-boxes, interpreting only the final products within the logic.

Acknowledgments

This work was supported in part by NSF grant IIS-9724937. This work is part of the dialogue effort by the University of Maryland Active Logic Group, other members of this group who have contributed to the work presented here include Don Perlis, K. Purang, and Darsana Purushothaman. We would also like to thank James Allen and George Ferguson from University of Rochester for allowing the use of the TRAINS-96 system as a platform in which to embed these ideas. Also, we would like to thank members of the TRINDI project and the Linguistics department of University of Gothenburg for helpful comments on previous versions of this material.

References

[Ahrenberg *et al.*, 1990] Lars Ahrenberg, Nils Dahlbäck, and Arne Jönsson. 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*, 1990.

[Allen *et al.*, 1996] James F. Allen, Bradford W. Miller, Eric K. Ringger, and Teresa Sikorski. A robust system for natural spoken dialogue. In *Proceedings ACL-96*, pages 62–70, 1996.

[Bretier and Sadek, 1996] P. Bretier and M. D. Sadek. A rational agent as the kernel of a cooperative spoken dialogue system: Implementing a logical theory of interaction. In J. P. Müller, M. J. Wooldridge, and N. R. Jennings, editors, *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, 1996.

[Carletta *et al.*, 1997] Jean Carletta, Amy Isard, Stephen Isard, Jacqueline C. Kowtko, Gwynewth Doherty-Sneddon, and Anne H. Anderson. The reliability of a dialogue structure coding scheme. *Computational Linguistics*, 23(1):13–31, 1997.

[Clark and Schaefer, 1989] Herbert H. Clark and Edward F. Schaefer. Contributing to discourse. *Cognitive Science*, 13:259–294, 1989.

[Davidson, 1967] Donald Davidson. The logical form of action sentences. In N. Rescher, editor, *The Logic of Decision and Action.* University of Pittsburgh Press, Pittsburgh, PA, 1967.

[Elgot-Drapkin and Perlis, 1990] J. Elgot-Drapkin and D. Perlis. Reasoning situated in time I: Basic concepts. *Journal of Experimental and Theoretical Artificial Intelligence*, 2(1):75–98, 1990.

[Group, 1993] External Interfaces Working Group. Draft specification of the kqml agent-communication language. available through the WWW at: http://www.cs.umbc.edu/kqml/papers/, 1993.

[Gurney *et al.*, 1997] John Gurney, Donald Perlis, and Khemdut Purang. Interpreting presuppositions using active logic: from contexts to utterances. *Computational Intelligence*, 13:391–413, 1997.

[McRoy *et al.*, 1997] Susan W. McRoy, Susan Haller, and Syed Ali. Uniform knowledge representation for language processing in the b2 system. *Journal of Natural Language Engineering*, 3(2/3):123–145, 1997.

[Perlis *et al.*, 1998] D. Perlis, K. Purang, and C. Andersen. Convesational adequacy: mistakes are the essence. *Int. J. Human-Computer Studies*, 48:553–575, 1998.

[Poesio, 1994] Massimo Poesio. *Discourse Interpretation and the Scope of Operators.* PhD thesis, University of Rochester, 1994. Also available as TR 518, Department of Computer Science, University of Rochester.

[Purang *et al.*, 1999] K. Purang, D Purushothaman, D. Traum, C. Andersen, and D. Perlis. Practical reasoning and plan execution with active logic. In *IJCAI-99 Workshop on Practical Reasoning and Rationality*, 1999.

[Schegloff and Sacks, 1973] Emmanuel A. Schegloff and H. Sacks. Opening up closings. *Semiotica*, 7:289–327, 1973.

[Seneff *et al.*, 1996] S. Seneff, D. Goddeau, C. Pao, and J. Polifroni. Multimodal discourse modelling in a multi-user multi-domain environment. In Proceedings 4th International Conference on Spoken Language Processing (ICSLP-96), 1996.

[Severinson Eklundh, 1983] Kerstin Severinson Eklundh. The notion of language game – a natural unit of dialogue and discourse. Technical Report SIC 5, University of Linköping, Studies in Communication, 1983.

[Shapiro, 1979] Stuart C. Shapiro. The SNePS semantics network processing system. In Nicholas V. Findler, editor, *Associative Networks: Representation and Use of Knowledge by Computers*, pages 179–203. Academic Press, 1979.

[Sinclair and Coulthard, 1975] J. M. Sinclair and R. M. Coulthard. *Towards an analysis of Discourse: The English used by teachers and pupils.* Oxford University Press, 1975.

[Smith *et al.*, 1995] Ronnie W. Smith, D. Richard Hipp, and Alan W. Biermann. An architecture for voice dialogue systems based on prolog-style theorem proving. *Computational Linguistics*, 21(3):281–320, 1995.

[Sutton *et al.*, 1996] S. Sutton, D. G. Novick, R. A. Cole, and M. Fanty. Building 10,000 spoken-dialogue systems. In Proceedings 4th International Conference on Spoken Language Processing (ICSLP-96), 1996.

[Traum and Allen, 1994] David R. Traum and James F. Allen. Discourse obligations in dialogue processing. In *Proceedings of the 32^{th} Annual Meeting of the Association for Computational Linguistics*, pages 1–8, 1994.

[Traum and Hinkelman, 1992] 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.

[Traum, 1994] David R. Traum. *A Computational Theory of Grounding in Natural Language Conversation.* PhD thesis, Department of Computer Science, University of Rochester, 1994. Also available as TR 545, Department of Computer Science, University of Rochester.

[Traum, 1996] David R. Traum. Conversational agency: The trains-93 dialogue manager. In *Proceedings of the Twente Workshop on Langauge Technology: Dialogue Management in Natural Language Systems (TWLT 11)*, pages 1–11, 1996.

[Wells *et al.*, 1981] Gordon Wells, Margaret MacLure, and Martin Montgomery. Some strategies for sustaining conversation. In Paul Werth, editor, *Conversation and Discourse.* Croon Helm, 1981.