A Reliable Natural Language Interface To Household Appliances

Alexander Yates
University of Washington
Computer Science
Seattle, WA 98105 USA
206-616-1844
ayates@cs.washington.edu

Oren Etzioni
University of Washington
Computer Science
Seattle, WA 98105USA
206-685-3035
etzioni@cs.washington.edu

Daniel Weld
University of Washington
Computer Science
Seattle, WA 98105 USA
206-543-9196
weld@cs.washington.edu

> "*I have always wished that my computer would be as easy to use as my telephone. My wish has come true. I no longer know how to use my telephone*."
>
> – Bjarne Stroustrop (originator of C++)

ABSTRACT

As household appliances grow in complexity and sophistication, they become harder and harder to use, particularly because of their tiny display screens and limited keyboards. This paper describes a strategy for building natural language interfaces to appliances that circumvents these problems. Our approach leverages decades of research on planning and natural language interfaces to databases by reducing the appliance problem to the database problem; the reduction provably maintains desirable properties of the database interface. The paper goes on to describe the implementation and evaluation of the EXACT interface to appliances, which is based on this reduction. EXACT maps each English user request to an SQL query, which is transformed to create a PDDL goal, and uses the Blackbox planner [13] to map the planning problem to a sequence of appliance commands that satisfy the original request. Both theoretical arguments and experimental evaluation show that EXACT is highly reliable.

Categories and Subject Descriptors

I.2.7 [Natural Language Processing]: Language Parsing and Understanding.

General Terms

Reliability, Human Factors.

Keywords

Natural language interface, database, appliance, planner.

1. INTRODUCTION AND MOTIVATION

The exponential drop in microprocessor cost over time has enabled appliance manufacturers to pack increasingly complex feature sets into appliances such as phones, TVs, microwave ovens, MP3 players, and more. Networked homes of the future will allow even more complex functionality. Yet even today, consumers are typically unable or unwilling to decipher increasingly thick and all-too-often incomprehensible user manuals. As a result, they often limit themselves to only a small fraction of their appliances' capabilities. Humorist Dave Barry captured this sentiment when he wrote:

> "*I have a feature-packed telephone with 43 buttons, at least 20 of which I am afraid to touch. This phone probably can communicate with the dead, but I don't know how to operate it, just as I don't know how to operate my TV, which has features out the wazooty and requires THREE remote controls...*" [5]

Most appliances have a very small screen and a limited set of buttons compared with a personal computer. Thus, standard Graphical User Interface (GUI) techniques such as browsing, menu trees, and online help are far less appealing for appliances. It is unlikely that these form factors will change because they are dictated by the desired size, weight, and "look" of the appliance. As the TV example illustrates, the problem of appliance interfaces grows more acute when multiple devices[1] interact – a scenario that is becoming increasingly common as the era of pervasive computing approaches.

Clearly, a conversational interface to appliances is worth investigating. In addition to circumventing the form factor issues of a GUI, a conversational interface would allow remote, hands free operation of appliances. Imagine walking into your home and saying, "Phone: any messages? Thermostat: raise the temperature 5 degrees... VCR: record Seinfeld." As devices become networked, one could even go to the living room and say "Lights: flicker when the microwave is done.''

The field of speech recognition has made great strides in the last 10 years and continues to do so. However, many speech interfaces are still limited to single word commands. We posit that a conversational speech interface would be far more desirable. While natural language interfaces have been studied extensively in AI, particularly in the context of databases [4], natural language interfaces to household appliances have received scant attention to date. Our paper raises the simple question: how do we build a Natural Language Interface to Appliances (NLIA)?

An NLIA maps an English sentence (e.g., "defrost 2 pounds of corn") to a sequence of appliance commands that aims to satisfy

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
IUI'03, January 12–15, 2003, Miami, Florida, USA.
Copyright 2003 ACM 1-58113-586-6/03/0001…$5.00.

[1] We use the terms "appliance" and "device" interchangeably.

the original request.[2] While NLIAs are largely unexplored, there has been more than thirty years of research on Natural Language Interfaces to DataBases (NLIDBs). An NLIDB maps an English sentence to a corresponding SQL statement.

Can we leverage the body of knowledge accumulated in decades of studying NLIDBs to help in designing and building NLIAs? As it turns out, we are able to make a strong claim in this regard, which is our central insight: *for a broad class of devices, which appears to include all household appliances, the problem of NLIA is provably reducible to the NLIDB problem.* That is, given an NLIDB, a planner, and a model of an appliance, we are able to automatically generate an NLIA for the appliance. We substantiate our claim both formally and experimentally.

The remainder of the paper is organized as follows. Section 2 discusses the framework for our problem, including design requirements and significant hurdles. Section 3 presents an example that we will refer to and develop throughout the paper, and it also discusses some simplifying assumptions. In section 4, we illustrate the reduction method and state the relevant theoretical results and arguments. Section 5 describes the EXACT implementation – a working NLIA based on the reduction method – and section 6 reports on experiments measuring EXACT's performance. Sections 7 and 8 present a discussion of directions for future research and related work in the areas of NLIDB, planning, and dialogue interfaces.

2. GUIDING PRINCIPLES

Four principles underlie our approach to designing a reliable NLIA. We insist that our NLIA be predictable, process high level goals correctly, respond appropriately to impossible requests, and generate safe plans. We explain these principles in more detail below.

2.1 Predictability

As Norman and Schneiderman have argued [17, 24], predictability is an essential feature of a user interface; without it, users will lose the essential feeling of control. Norman and Schneiderman have taken their arguments as an indictment of the intelligent user interface paradigm. However, we can take the need for predictable interfaces to heart without giving up on intelligent user interfaces. Let's consider the issue in the context of NLIAs.

There are two main aspects to an NLIA: understanding what the user wants, and carrying out her request. If an NLIA can reliably achieve these two tasks, then the user will feel in control. As we explain below, EXACT uses a sound and complete NLIDB to understand the user's goal, and a sound and complete planner to guide action. As section 4.2 will show, the combination of guarantees on the NLIDB and the planner yields a sound and complete NLIA. Of course, these guarantees, while helpful, are no panacea: if the request is ambiguous, some sort of clarification dialog will be necessary. However, our experimental results (section 6) provide preliminary evidence that ambiguities are rare and that EXACT is reliable and predictable in practice.

[2] Our NLIA is not a full-blown conversational interface; the NLIA focuses on the task of reliably understanding single sentences. It is best viewed as a powerful *module* to be integrated into a full-blown dialog system.

2.2 Complex And Impossible Requests

An NLIA insulates the user from the peculiarities of an appliance's command language and obviates the unpleasant task of reading and re-reading appliance manuals. As a result, there may be some mismatch between the user's mental model of appliance capabilities and the appliance's exact command set. This mismatch can come in at least two flavors. First, the user's request may require a *sequence* of commands to satisfy it, instead of a single command. We refer to such requests as *goals*. Second, the user may issue a request that is impossible to satisfy. We refer to such requests as *impossible requests.* For example, the KX-TC1040W phone does not have a 'mute' button, so a user's request to mute a call has to be declined. In general, when the user is not looking at the device, she misses the physical cues that suggest how the device can be used. We expect our NLIA to handle both goals and impossible requests appropriately. In the case of goals, the user need not be aware that her request translates into a sequence of appliance commands – the NLIA ought to make that transparent to the user. In the case of impossible requests, the NLIA ought to respond in a way that makes clear that the request was understood, but cannot be satisfied (e.g., as in HAL's infamous line "I can't do that, Dave").

2.3 Safety

When processing a complex request with a search-based planner, one must confront the possibility that the resulting plan, while achieving a user's goal, may have unintended and harmful consequences. Consider, for example, the goal "delete my old messages"; because our answering machine has a single command for deleting all messages (both old *and* new), a simple plan might have surprising and unintended consequences. Since a powerful NLIA can cause considerable havoc if not restrained, some mechanism for controlling side-effects is crucial.

In 1994, Weld and Etzioni [28] introduced the ideas of safety and tidiness in planning, ideas that were meant to restrain intelligent agents from harming people or their property. *Safety constraints* tell the planner never to violate certain conditions. For example, "*dont-disturb(written.to.tape(f) or isa(f, file))*" tells a planner that no file can be deleted unless it is already written to tape. Unfortunately, these kinds of constraints are often too strict for our domain. We do not want to tell the planner that it can never delete a message. Rather, we want to constrain the planner so that it never deletes a message unless the user tells it to.

Intuitively, one would like to tell the planner to *minimize* side-effects that were not requested by the user, but Weld and Etzioni recognized that obeying such a constraint is intractable in the worst case, since it requires reasoning over the (infinite) space of all plans to find the minimum. Instead, they proposed *tidiness constraints*, which tell the agent to restore the state of the world as much as possible to the way it was in the state before the plan started executing; "*restore(compressed(file))*", for example, tells the agent's planner to recompress as many files as possible after achieving the user's main goal. When actions are neatly reversible, tidiness constraints often cause the planner to achieve the intuitively desirable minimization. Unfortunately, many appliances have irreversible actions, like deleting messages on an answering machine, or cooking a dish in a microwave. Thus tidiness is not well suited for our domain. Section 4.1 explains how we define and enforce a new kind of safety constraint that is a good fit for the appliance domain.

3. EXAMPLE AND SIMPLIFYING ASSUMPTIONS

In order to illustrate the ideas underlying our reduction of an NLIA to an NLIDB, we use the Panasonic KX-TC1040W telephone/answering machine as a running example throughout the paper. Our model of the phone system includes 37 actions and involves a database schema with 5 relations. The relations contain anywhere from two attributes for the answering machine messages (pictured below in table 1) to eleven attributes for the phone state. The actions include phone commands such as changing the ringer volume as well as answering machine commands and commands to access phone company services such as call forwarding.

Note that while we have fully implemented the EXACT NLIA, we have tested it on a *simulation* of the Panasonic device, rather than the actual appliance hardware; we have not done the wiring and tinkering that would be required to actually drive the appliance. Nevertheless, our command set was taken directly from the manual for the appliance [1], so we are confident that our simulation is realistic.

Our expertise is not in speech; hence the focus of this paper is on developing an expressive NLIA as a step towards the ultimate goal of linking it to a speech recognizer.

In contrast with the work of Moore, Allen, and Walker [30, 3, 26], we have not built a full-blown dialog system. Instead, we focus on the core capability of understanding single-sentence appliance commands such as "Cook my corn for 5 minutes" as well as goals such as "Delete my old messages," which requires a multi-step plan to find each old message and erase it in turn. Thus, our NLIA implements a function from a single English sentence encoding a person's request to a command sequence that satisfies the request when executed on the appliance. Due to its reliability, we believe that our NLIA would be an attractive module for researchers investigating dialog systems.

While our model is readily extensible to multiple devices in a networked home, we have not yet addressed the issue of identifying which device the user is addressing based on context or content. However, the straightforward approach of explicitly naming a device when addressing it seems reasonable. For example, a person could say "VCR: record Seinfeld."

Finally, we assume that the NLIA has an accurate behavioral model of the appliances with which it integrates. If exogenous events can affect the device (e.g., an external caller leaving a message), we assume that the device will notify the NLIA of this fact. This assumption is reasonable because all existing devices we surveyed notify the user of exogenous events (e.g., the phone rings, the answering machine displays a count of new messages, and the thermostat display indicates whether the furnace is on or off). While our implemented system depends on this assumption, our overall approach does not. By using a more complex, information-gathering planner such as PUCCINI or XII [9, 10, 11], our NLIA would operate correctly even without notification of these events.

Interpreting a user's commands is more complex if there are multiple plans being executed at the same time. In this case a user's command can affect not just the appliance, but also the agent's execution stack of previously planned actions. Thus, we assume that the NLIA will process new requests only after previously planned actions have been fully executed.

message_number	message
1	Old
2	Old
3	New
…	…
64	Blank

Table 1: The "message_list" table is one of five relations comprising the database that EXACT uses to represent the state of the Panasonic KXTC1040W.

4. NLIA BY REDUCTION

In this section we show how to build an NLIA using an NLIDB and a planner. The reduction is based on the observation that user commands to an appliance are made relative to the device's state, either by querying the state (e.g., "When is the sprinkler system set to water?") or modifying it (e.g., "Set the thermostat to 68°."). Since a relational database is a convenient and natural way to conceptualize a device's state, a user's command can be modeled with SQL statements, and can be computed using an NLIDB. In order to create a full NLIA, however, we need to show a method for satisfying the SQL query and update statements, using the device's primitive command set – for this task, we use a planning algorithm. As we argue below, one of the advantages of this approach is the construction of a reliable (i.e., sound and complete) NLIA by exploiting the forma

```
(:action delete-message
   :parameters (?x - integer)
   :precondition (and (leq 1 x)  (leq x 64) (playing ?x))
   :effect (and (not (playing ?x))
                (when (leq ?x 63)
                   (playing (+1 ?x)))
                (forall (?y)
                    (and (not (message-list ?x ?y ?z))
                         (message-list ?x Blank Old)))))

(:action play
   :parameters ()
   :precondition (not (playmode))
   :effect (and (playmode)
                (playing 1)
                (not (message-list 1 New))
                (message-list 1 Old)))

(:action play-next
   :parameters ()
   :precondition (playmode)
   :effect (forall (?x)
              (when (and (playing ?x) (leq ?x 63))
                 (and (not (playing ?x)) (playing (+1 ?x))
                      (not (message-list ?x New))
                      (message-list ?x Old)))))
```

l properties of existing NLIDB and planning systems.

Formally, we model an appliance as a pair, <A, DB>, where A is a set of action descriptions in the PDDL planning language [15] and DB is a database representation of the appliance's initial state. For example, table 1 shows a fragment of a sample relation from the database model of its internal state, and figure 1 shows some sample actions for the Panasonic phone.

Our NLIA is composed of four parts: the appliance model <A, DB>, an NLIDB, a translation module, and a planner. At run-time, the system takes as input a natural language sentence and

feeds it to the NLIDB (figure 2). The NLIDB converts the input to an SQL statement consistent with the schema of database DB.

Figure 1: PDDL encoding of some Panasonic KXTC1040W actions.

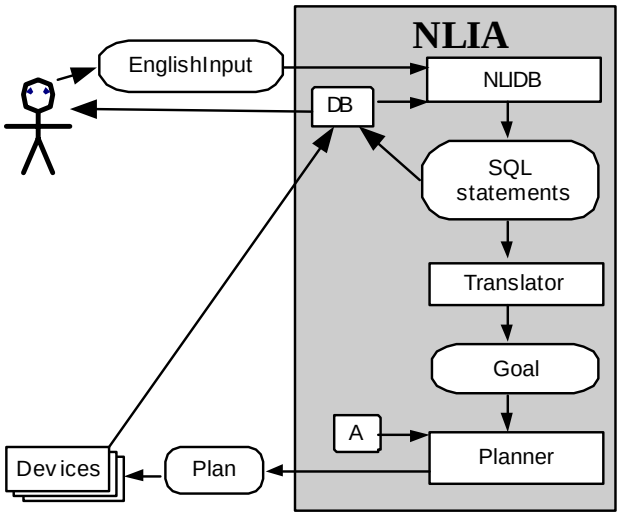

Figure 2: Building an NLIA out of an NLIDB and a Planner. The gray box depicts the NLIA.

If the SQL denotes a query, it is executed on DB and the result is returned to the user. For example, the English question, "what is the answering machine volume?" is mapped to the following SQL query:

```
SELECT volume
FROM answer_machine
```

If the SQL is an update, the translator converts it into a goal in the planning language, which is then sent to the planner to generate a sequence of actions from A. When these actions are executed on the appropriate devices (and also when exogenous events occur), DB is updated to ensure correspondence with the device's actual state.

We are able to skip the translation and planning stages in the case of a SELECT query because of our assumption that the database contains the complete state of the appliance. If this assumption were not met, it might be the case that several actions would have to be taken before the action that answers the query, and such a case would require planning.

The translation step, converting from an SQL update statement to a goal, requires some explanation. As Reiter has shown [22], database updates can be modeled using the situation calculus; for our purposes PDDL suffices also.

Without loss of generality, suppose that DB has relational schema $\mathbf{X} = \{X_1, \ldots, X_n\}$, where each relation X_j has attributes $a_{j1}, \ldots, a_{jk}$, where k varies from table to table. DB contains a set of tuples satisfying each X_j and because of our earlier assumption regarding notification of exogenous events, we can make the closed world assumption [23].

An SQL update statement has the form:

```
UPDATE Xj
SET a1=c1, …, au=cu
WHERE am=cm, …, an=cn
```

where the a_i are attributes of X_j, and the c_i are constants. We can convert this SQL statement into a PDDL goal with a process similar to that used for generating a propositional form for a universally quantified goal in classical planning [27]. The first step is running the following SQL query on DB, the agent's model of the device.

```
SELECT DISTINCT a1, …, ak
FROM Xj
WHERE am=cm, …, an=cn
```

Execution of this query will retrieve a set of tuples, $\{t_i\}$, whose values need to be modified. Let $\{t_i'\}$ denote the corresponding set of tuples obtained by changing the value of the j^{th} attribute of each tuple to c_j, for all $m \le j \le n$. Let Ψ denote the set of all tuples in DB, and let CWA denote the function which computes the closure of a set of relational atoms.[3] The planner is given the following ground problem:

$$\text{Init} = \Psi; \quad \text{Goal} = \text{CWA}(\Psi - \{t_i\} \cup \{t_i'\})$$

The reason for grounding the goal during the translation step is somewhat subtle. The SQL UPDATE command's SET clause refers to the goal state, i.e. the desired state of the device after the plan has been executed. In contrast, the WHERE clause refers to the state of the device before any changes are made; that is, it refers to the device state at the commencement of planning. Unfortunately, PDDL has no notation for making this distinction[4], so if the translation step left any universally quantified variables in the resulting goal, they would all refer to the goal state, or the state after the plan.

4.1 Safety Revisited

We include Ψ as part of the goal to ensure that the planner doesn't generate a plan that has the nasty side effect of falsifying something that is currently true. For example, the simplest plan to delete all *old* messages is to delete *all* messages, new and old (the phone has a single command to do this). By explicitly stating in the goal that new messages should not be deleted, we force the planner to come up with a safe plan. Similarly, we would not want the planner to respond to "call Sue" by first randomly recording a new answering machine greeting, and only then placing the call. Thus, we prevent the planner from unwanted positive side effects by computing the CWA. In domains where resource usage or other side effects are necessary, one can exclude predicates describing these resources from the goal.

As an example of this reduction in action, consider the operation of our Panasonic phone NLIA on the sentence, "Delete all my old messages." Assume that this NLIA is given the actions from Figure 1 and the database fragment from Table 1 as part of its inputs. The NLIDB translates the input sentence into the SQL statement:

```
UPDATE message_list
SET message = Blank
WHERE message = Old.
```

[3] Closing a set of positive literals means explicitly adding the negation of any atom, which is absent from the set. If the set of relations and constants is finite and there are no function symbols (which is our case), this operation takes polynomial time.

[4] See, for instance, SADL [8] for an action description language that does provide notation to distinguish between the initial and goal states.

Our translator takes this SQL statement and performs the syntactic manipulation required to rewrite it as a grounded goal in PDDL. Let <old-msg-num-1> through <old-msg-num-n> represent the numbers of all the old messages in the message_number column of table message_list, and let <new-msg-num-1> through <new-msg-num-m> represent the numbers of all the new messages. In PDDL the grounded goal looks like:

```
(and (not (message-list <old-msg-num-1> Old))
     (message-list <old-msg-num-1> Blank)
     …
     (not (message-list <old-msg-num-n> Old))
     (message-list <old-msg-num-n> Blank)
     (message-list <new-msg-num-1> Old)
     …
     (message-list <new-msg-num-m> Old))
```

For brevity's sake, we omit the other relations in the database from the above goal. The planner takes this goal, together with the initial state in the database DB, and returns a plan starting with the play action. Next, the plan will contain a delete-message action if message number one is one of <old-msg-num-1> through <old-msg-num-n>, and a play-next action otherwise. This repeats until all 64 messages have been checked and deleted if they are old.

4.2 Formal Properties

We are now in a position to state the benefits of our NLIA reduction precisely. Abstractly, one can consider an NLIDB, N, as a function from English sentences to SQL statements. Similarly our translator, T (described above), is a function from SQL to planning problem specifications.[5] Finally, a planner is a function from these problem specifications to action sequences. Since an NLIA takes an English sentence, ε, and generates action sequences for the appliance, one can summarize our reduction as follows:

$$NLIA(\varepsilon) \equiv P \circ T \circ N (\varepsilon)$$

Popescu *et al.* [20] define the conditions under which an SQL statement is a *valid interpretation* of an English sentence ε, but the definition is too complex to include in this paper. We borrow from [20] the far simpler definitions of soundness and completeness below.

Definition. An NLIDB is *sound* if any SQL it outputs is a valid interpretation of its input sentence ε. An NLIDB is *complete* if it returns all valid interpretations of ε.

Note that if an NLIDB is both sound and complete and it returns a single SQL statement in response to a user's utterance, then it has unambiguously determined the user's intent – *subject to our assumptions, of course*.

Definition. Let S be an SQL statement over a relational database DB. An appliance reaction R is *consistent* with S if S is a query and R answers the query, or if S is an update and R is a sequence of legal device commands that changes DB accordingly. An NLIA is *sound* if in response to input ε, its reaction is consistent with some valid interpretation of ε. An NLIA is *complete* if it makes a consistent reaction to a valid interpretation of ε, when one exists.

[5] Given a fixed set of actions, a planning problem is an initial state / goal pair, thus T maps from SQL to the cross product of tuple specifications with itself.

There are a variety of formulations of automated planning [19], but we briefly summarize with the following.

Definition. A planner is *sound* if any plan it outputs will transform the initial situation into a world state where the goal holds. A planner is *complete* if it returns a plan when one exists.

We can now formally state the two central benefits of our reduction:

Proposition 1 [Soundness]. Let N be a sound NLIDB, let P be a sound planner, and let T be the translation scheme described above. Then P ◦ T ◦ N is a sound NLIA.

Proposition 2 [Completeness]. Let N be a complete NLIDB, let P be a complete planner, and let T be the translation scheme described above. Then P ◦ T ◦ N is a complete NLIA.

The proofs are omitted due to lack of space.

4.3 Significance of the Theory

Of course, theoretical guarantees only apply in practice if their assumptions are satisfied (e.g., all the words in the sentence are known – see [20] for the complete enumeration). Our experimental results (section 6) provide some evidence that these assumptions are realistic.

Another potential objection to our theory is that it does not guarantee the reliability of a full-blown conversational speech interface; speech recognition, in particular, is likely to result in errors. While this is clearly true, we see great value in having an NLIA that is guaranteed to reliable – this enables the interface designer to localize errors to other modules and to institute the appropriate recovery strategy.

Consider, by way of analogy, a sophisticated chess-playing program that combines mini-max search with alpha-beta pruning, a complex and tunable evaluation function, specialized hardware, etc. Suppose we prove that alpha-beta pruning is "reliable" in that it only prunes moves that the search procedure would eventually discard. Well, the reliability of alpha-beta pruning does not guarantee that the chess program always makes the best move. However, when the program makes a mistake, we know that it is definitely *not* due to alpha-beta pruning. Again, the guaranteed reliability of one module, enables the program's designer to focus his attention on other modules.

5. THE "EXACT" IMPLEMENTATION

In order to test the theory developed in section 4, we built the EXACT natural language interface to a telephone using the Precise NLIDB [20] and Blackbox planner [13] as foundations. We handcrafted a database model for the Panasonic KXTC1040W from its user manual [1]. This model is used both as an input to the Precise NLIDB and as the source of state information for the planner. Finally, we created a set of actions that model the phone's commands, as described in the user manual. This action set is also input to the planner.

The system takes an input sentence, converts it into a set of possible SQL statements using Precise, translates those into a set of goals, and looks for a plan to satisfy each goal. If there is more than one goal and at least one goal has a plan, then we have an ambiguous sentence, and EXACT needs to ask the user for help in disambiguating. If no goal has a plan, then the phone cannot support the function being asked for, so EXACT can tell the user as much. If there is exactly one goal and it has a plan, EXACT

can simply carry out that plan. In our experiments, the last case was by far the most common.

The dataset on which we evaluated our system includes examples of impossible requests, but EXACT is well equipped to handle this problem: if a sentence does not map to an appropriate SQL statement, either because of unknown words or because there is no attribute-value pairing for the sentence, then we can say that the NLIDB cannot understand the sentence. On the other hand, if the sentence maps to an SQL statement, but the planner fails to find a plan for that goal, then since our planner is complete we can say that the appliance does not support this function.

Our interface inherits desirable qualities, like reliability and portability across many appliances, from the planner and the NLIDB, but we had to make extensions to both components as explained below.

5.1 The NLIDB Component

Precise is a highly portable NLIDB that guarantees soundness of its SQL interpretations for certain kinds of English sentences, called *semantically tractable questions* [20]. Precise automatically generates a lexicon based on the names of relations, attributes, and values in its input database. At its core, it reduces the problem of mapping a sentence to an SQL query to a graph-matching problem, which can be solved using the maxflow algorithm. Precise relies on the Charniak parser to extract attachment information from the sentence to help constrain the matching problem. Finally, Precise generates the complete set of possible SQL interpretations of the sentence that are consistent with its lexicon and its parser.

Precise was originally built to support only questions, which translate to SELECT queries in SQL. Since an NLIA has to respond to requests for changing the state of the appliance, which translate naturally to SQL UPDATE statements, we had to extend Precise appropriately. There is no room to explain the technical details of Precise, but the essence is an extension of Precise's graph matching algorithm to keep track of two attributes, a "pre-attribute" and a "post-attribute." The pre-attributes contain values in the database state before an UPDATE, and the post-attributes contain the new, updated values. Any values that are set by an UPDATE statement are matched with post-attributes, and all other values are matched with pre-attributes. In order to understand an input sentence, the system needs to classify it as an UPDATE or a SELECT, and proceed appropriately. Precise's other modules, including its tokenizer, lexicon, and parser, remain unchanged.

5.2 The Planning Component

We have already explained how we prevent unwarranted side effects and how we finesse the problem of information gathering. But the domain of household appliances also has a surprising amount of temporal complexity.

First, user commands may involve events occurring at specific future times (e.g., "record Star Trek"). While one could use a temporal planner to handle these goals, we instead rely on the temporal capabilities of the device itself. Since it is possible *now* to set a VCR to program *later*, EXACT can handle this goal with a classical planner. If the VCR did not support this type of operation, EXACT would explain that the goal was unachievable.

Second, numerous actions are *durative*; execution occurs over an interval of time (e.g., "play all messages"). Although it might seem natural to model the temporal aspects of these actions explicitly (e.g., in PDDL 2.1 level 3[6]), we tried this and discovered problems (see below).

Finally, different commands naturally translate into goals with different temporal annotations, but the nature of the temporal mapping is complex and subtle. For example, consider the command "Play all messages." Note that the user (presumably) doesn't want the answering machine to play the messages forever; once is enough. Thus if one models `play' as a durative action (which transiently plays a message), one cannot model the goal as one of achievement. In a durative model, the goal has an implicit temporal annotation that it must be true *at some point* during execution, even if it is not true at the end of all execution. There are two problems with such a model. First, few planners support such expressiveness; indeed PDDL 2.1 level 3 does not even allow one to express the goal. But the deeper problem is related to disambiguating natural language. Consider the commands "Play all messages" (which does *not* require playing to be true at the end of the plan) and "Turn on answering machine" (which does). We could think of no principled way to distinguish between these goals; clearly the user would be very upset if EXACT responded to the last command by turning answering on and then off again!

Our solution is to use a classical atomic model of time, implicitly recognizing that exogenous events may subsequently change the device state. Philosophically, this is consistent with PDDL 2.1 level 5 in which all actions are instantaneous, but some may initiate physical processes (in this case the process of a message being played) that evolve over time. We do not need to model the processes explicitly, since (by assumption) the networked device notifies the agent of events such as incoming messages. Thus execution of the play command does *not* terminate with the device playing; it simply stops later of its own accord. By modeling the device in this fashion, we are able to use the Blackbox planner [13], which operates by compiling PDDL planning problems into a set of propositional clauses that is satisfiable exactly when an n-step plan exists. A fast satisfiability algorithm is used; if an assignment is found, a reverse compilation phase generates the plan; otherwise, the plan length is incremented.

6. EXPERIMENTAL EVALUATION

To test our system, we gathered a total of 72 sentences from seven graduate students and faculty at the University of Washington. The people who provided us with data were given a concise list of the features available on the Panasonic phone and were asked to write down sentences in English that they might use to invoke those features.

Of the 72 sentences gathered, we used 61 in our experiments. The eleven sentences we excluded did not describe a goal or an action, but rather they implied it. For example, we excluded the sentence, "It's too loud." This sentence does not describe a goal state or an action to achieve a goal. Instead, it describes the current state and implies the goal. Such sentences clearly demonstrate the importance of processing speech acts, but are beyond the current capabilities of EXACT.

[6] http://www.dur.ac.uk/d.p.long/competition.html

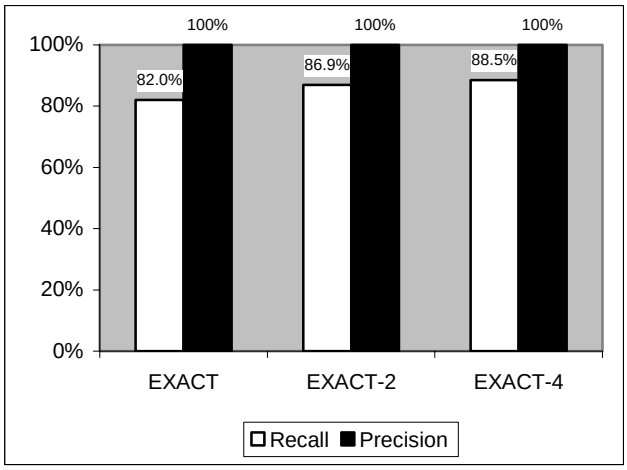

Figure 3: EXACT's performance in our experiment.

Figure 3 shows EXACT's performance on our dataset. Recall is the fraction of the sentences in the dataset where EXACT put forth an interpretation.[7] We refer to these sentences as *tractable* sentences. On intractable sentences, EXACT indicates it cannot understand the request and asks for a paraphrase.

Precision is the fraction of the tractable sentences in the dataset that EXACT interpreted correctly (including the interpretation that the sentence was an impossible request, where appropriate). To measure precision and recall on our data, we labeled each sentence with one or more valid interpretations.[8] As the black bars in the figure show, we achieve 100% precision on our dataset, reflecting our commitment to reliability and the utility of our underlying theory.

EXACT's recall is 82.0% because on eleven sentences it is unable to choose a definitive interpretation. If we allow EXACT to ask the user to choose between two possible interpretations (EXACT-2) the recall goes up to 86.9%, and if the user may choose between up to four interpretations the recall goes up slightly to 88.5% (EXACT-4). In the remaining cases, EXACT is unable to interpret the sentence because it contains words outside of EXACT's lexicon. Unlike some natural language interfaces, EXACT does not attempt to ignore unfamiliar words. To ensure reliability, EXACT declines to interpret a sentence that contains any unknown words.

The increasing recall from EXACT to EXACT-2 and then to EXACT-4 quantifies the amount of ambiguity in our data. Two sentences have exactly two interpretations; both cases reflect the fact that the phone has two different volume settings (one for the answering machine and one for the ringer). For a sentence like, "Turn up the volume," there are two interpretations, and both are potentially correct. Only one other input sentence has multiple interpretations in our data, and that is the somewhat idiosyncratic single-word command "Aloud." The four interpretations of this sentence are actions to turn on the speakerphone (the intended goal), turn on the handset ringer, turn on the intercom, and playback messages, all of which play sounds aloud. With further tuning of the system, such ambiguities could be resolved.

[7] Remember that to maximize reliability, EXACT does not try to "guess" the correct interpretation when it is not sure.

[8] EXACT's interpretation is correct if it is a member of the set of valid interpretations.

7. FUTURE WORK

There are a number of important directions for future work. First, we need to test EXACT on a much wider range of users and appliances. Second, we need to link EXACT to actual device hardware, and to a speech recognizer. Third, we need to address the user interface issues that arise due to speech recognition errors and hardware problems. For example, it is appropriate for EXACT to confirm its plans with the user before taking some actions, but excessive confirmation can be a nuisance to the user. Finally, it will be essential to enable EXACT to participate in full-blown dialogs with users. Such dialogs would enable EXACT to choose between multiple competing interpretations and to learn new words, phrases, or idioms, thereby improving its recall.

8. RELATED WORK

We build on the large body of work in natural language interfaces to databases. See Androutsopoulos *et al.* [4] for a survey. Only a small number of NLIDBs handle updates, and none have considered the NLIA problem.

Our emphasis on provable soundness as a foundation for a reliable natural language interface is shared with Popescu *et al.* [20], but our work on EXACT goes beyond Precise in several important ways. First, we introduce and analyze the idea of reducing the NLIA problem to the NLIDB problem while maintaining the soundness and completeness of the interface. Second, EXACT composes Precise with a planner to automatically generate an NLIA. Third, we extended Precise to handle updates. Finally, we show experimentally that linking EXACT to a 'typical' household appliance, the Panasonic KXTC1040W phone, yields a highly reliable interface that can handle goals, impossible requests, and safety concerns.

Young and Moore [30] have described DPOCL, a sound discourse planner that satisfies a limited form of completeness. They argue that the formal properties of previous discourse planners have been largely ignored, and that this lack of understanding leads to inconsistencies in the representation of discourse. Their focus, however, is on representing speaker intentions in texts. EXACT focuses on simpler natural language utterances, and seeks to use them to control household appliances.

Quesada et al. [21] describe a spoken dialogue agent in the D'Homme project that is specifically designed for interacting with household appliances. The agent uses a semantic grammar for a restricted and tractable subset of natural language that they call a Natural Command Language. The D'Homme agent is capable of understanding complex natural language dialogs, but it does not guarantee reliability. The agent also has no built-in planning capability, so it cannot handle all goals that require multi-step plans.

A number of commercial systems are being built to handle dialog with household appliances. Some examples include the Linguamatics Automated House[9], the SmartKom Home/Office[10], the Fluency House[11], and Voxi Smart Homes[12]. As these are commercial systems, they do not report vital information about their mechanisms.

[9] http://www.linguamatics.com/technology/dialogue/home.html

[10] http://www.smartkom.org

[11] http://visualhouse.fluencyvoice.com/housecgi/fluencyHouse.html

[12] http://www.voxi.com

TRIPS (Allen, *et al.* [3]) is an agent architecture for handling natural conversation in the travel-planning domain. In contrast to this kind of system, EXACT is designed to be easily portable to a number of different domains, and the domains of interest generally have a much simpler structure to the natural language interaction.

The Universal Speech Interface (USI) [25] project at CMU has design goals very similar to ours. USI/Gadget is a template system that is portable to many different appliances. The natural language capabilities of the system, however, are constrained by the speech recognition technology, so the interaction is keyword-based.

Various ubiquitous computing projects (e.g., MIT's intelligent room [6]) have considered multi-modal interfaces that include language. However, they have not considered the reliability of the language module, nor have they considered embedding a planner into the system to satisfy high-level goals, decline impossible requests, and abide by safety constraints.

A number of other researchers have modeled appliances and developed specification languages (e.g., in XML) for appliance interfaces [2, 8, 12, 18]. Systems designed around these specifications have tried to create a single interface to all appliances on another, remote appliance like a PDA. The Personal Universal Controller (PUC) [16] generates an interface to appliances, using XML specifications. Unlike EXACT, the PUC executes simple commands, not plans, and does not use natural language.

The menu2dialog system [14] creates a dialog planning system from a menu-based natural language system. It is not fully automated, however, and it does not consider the problems of reliability and safety.

Like EXACT, Softbots [7] use planners to develop complex plans on behalf of users. However, one drawback of softbot-based interfaces has been their lack of natural language capabilities, which can make them difficult to use, especially for novice users.

The Unix Consultant (UC) [29] is a natural language tutoring system for Unix. Although UC was designed as an interface to a device, UC has a very different focus from EXACT. UC is not portable across many devices; instead, its focus is on a complete model of a single, highly complicated "device" (the Unix shell). Furthermore, its response to user input is to *advise* the user on how to accomplish his or her goal, rather than performing actions itself. Finally, due to its complexity, UC makes no reliability guarantees.

9. CONCLUSION

This paper sketches a novel answer to the fundamental question: how do we build a reliable natural language interface to household appliances? Our answer, encapsulated in Figure 2, is to leverage more than thirty years of research on natural language interfaces in databases and reduce the appliance problem to the database problem. We show how, when coupled with a planner, this approach has a number of advantages, including formal guarantees of soundness, the ability to enforce safety, and the ability to appropriately handle high-level goals and impossible requests. Our preliminary experiment complements our theoretical arguments by showing that our interface in fact displays these advantages in practice.

10. ACKNOWLEDGEMENTS

We thank Ana-Marie Popescu for her help with using and extending the Precise system. We thank Krzysztof Gajos, Keith Golden, Tessa Lau, Mike Perkowitz, and the anonymous reviews for their insightful comments on previous drafts. This research was supported in part by NASA grant NAG 2-1538, NSF grants IIS-9872128 and IIS-9874759, and ONR grants N00014-02-1-0932 and N00014-02-1-0324.

11. REFERENCES

[1] Panasonic Cordless Answering Sys. Op. Instructions. http://www.pasc.panasonic.com/OperatingManuals/KXTC1040W.PDF. Panasonic Consumer Electronics Company.

[2] Abrams, M.; Phanouriou, C.; Batongbacal, A.L.; Williams, S.M.; and Shuster, J.E. UIML: An Appliance-Independent XML User Interface Language. *The Eighth International World Wide Web Conference.* 1999. Toronto, Canada.

[3] Allen, J.; Ferguson, G.; Stent, A. An architecture for more realistic conversational systems. *Intelligent User Interface*, 2001.

[4] Androutsopoulos, I.; Ritchie, G.D.; and Thanish, P. Natural Language Interfaces to Databases – An Introduction. *Natural Language Engineering, vol 1, part 1*, 29-81, 1995.

[5] Barry, D. "Remote Control." *The Washington Post*. Sunday, March 5, 2000. pp. W32. Tribune Media Services.

[6] Coen, M.; Weisman, L; Thomas, K; Groh, M. A Context Sensitive Natural Language Modality for the Intelligent Room. *Proceedings of MANSE'99.* Dublin, Ireland. 1999.

[7] Etzioni, O. and Weld, D. A Softbot-based Interface to the Internet. *Communications of the ACM*, July, 1994.

[8] Eustice, K.F.; Lehman, T.J., Morales, A.; Munson, M.C.; Edlund, S.; and Guillen, M. A Universal Information Appliance. *IBM Systems Journal.* 1999. 38(4): pp. 575-601.

[9] Golden, K. Leap Before You Look: Information Gathering in the PUCCINI Planner. *AIPS*, 70-77, 1998.

[10] Golden, K.; Etzioni, O.; Weld, D. Omnipotence without Omniscience: Sensor Management in Planning. *AAAI*, 1048-1054, 1994.

[11] Golden, K.; Weld, D. Representing Sensing Actions: The Middle Ground Revisited. *KR*, 1996.

[12] Haartsen, j.; Naghshineh, M.; Inouye, J.; Joeressen, O.J.; and Allen, W. Bluetooth: Vision, Goals, and Architecture. *ACM Mobile Computing and Communications Review.* 1998. **2**(4): pp. 38-45.

[13] Kautz, H. and Selman, B. BLACKBOX: A New Approach to the Application of Theorem Proving to Problem Solving. *AIPS98 Workshop on Planning as Combinatorial Search*. pp. 58-60. 1998.

[14] Larsson, S.; Cooper, R.; Ericsson, S. menu2dialog. *IJCAI Workshop on Knowledge And Reasoning In Practical Dialogue Systems*, 2001.

[15] McDermott, D. PDDL--The Planning Domain Definition Language. *AIPS*, 1998.

[16] Nichols, J.; Myers, B.A; Higgins, M.; Hughes, J.; Harris, T.K.; Rosenfeld, R.; Pignol, M. Generating Remote Control Interfaces for Complex Appliances. *CHI Letters: ACM*

Symposium on User Interface Software and Technology, 2002.

[17] Norman, D. How Might People Interact with Agents? in J. Bradshaw (Ed.), (1997). *Software Agents*. AAAI Press/The MIT Press, Menlo Park, CA.

[18] Olsen Jr., D.R.; Jefferies, S.; Nielsen, T.; Moyes, W.; and Fredrickson, P. Cross-modal Interaction using Xweb. *ACM SIGGRAPH Symposium on User Interface Software and Technology.* 2000. pp. 191-200.

[19] Penberthy, J.S. and Weld, D. UCPOP: A Sound, Complete, Partial Order Planner for ADL, *KR*, 103-114, 1992.

[20] Popescu, A.; Etzioni, O.; Kautz, H. Towards a Theory of Natural Language Interfaces. *Intelligent User Interfaces*, 2003.

[21] Quesada, J.F.; Garcia, F.; Sena, E.; Bernal, J.A.; Amores, G. Dialogue Managements in a Home Machine Environment: Linguistic Components over an Agent Architecture. *SEPLN*, 89-98. 2001.

[22] Reiter, R. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* MIT Press: Cambridge, MA, 2001.

[23] Reiter, R. On closed world databases. *Logic and Data Bases*, 55-76. ed. Gallaire, H. and Minker, J. Plenum Press, 1978.

[24] Schneiderman, B. and Maes, P., Direct Manipulation *vs.* Interface Agents, *Interactions*, 4(6), 1997, pp 42-61.

[25] Shriver, S.; Toth, A.; Zhu, X.; Rudnicky, A.; Rosenfeld, R. A Unified Design for Human-Machine Voice Interaction. *CHI*. 2001.

[26] Walker, M.; Fromer, J.; and Narayanan, S. Learning Optimal Dialogue Strategies: A Case Study of a Spoken Dialogue Agent for Email. *ACL/COLING 98*. 1998.

[27] Weld, D. An Introduction to Least-Commitment Planning. *Journal of AI*, 27-61, 1994.

[28] Weld, D. and Etzioni, O. The First Law of Robotics (a call to arms). *AAAI*. 1042-1047. 1994.

[29] Wilensky, R.; Chin, D.N.; Luria, M.; Martin, J.; Mayfield, J.; Wu, D. The Berkeley Unix Consultant Project. *Computational Linguistics*, 1988.

[30] Young, R.M.; Moore, J.D. DPOCL: A Principled Approach to Discourse Planning. *Seventh International Workshop on Text Generation.* pp. 13-20. 1994.