Can simple Natural Language Generation improve Intelligent Tutoring Systems?

Barbara Di Eugenio and Michael J. Trolio
Electrical Engineering and Computer Science Department
University of Illinois at Chicago
1120 SEO (M/C 154), 851 S. Morgan St
Chicago, IL, 60607 USA
{bdieugen,mtrolio}@eecs.uic.edu

Abstract

One of our general goals is to answer the question of what is the “added value” of a Natural Language interaction for a learner that interacts with an ITS. To do so, we applied simple Natural Language Generation techniques to improve the feedback provided by intelligent tutoring systems built within the DIAG framework (Towne 1997a). We have evaluated the original version of the system and the enhanced one with a between subjects experiment. The results are mixed. Although differences rarely achieve statistical significance, there appears to be slight improvement on performance measures in the group interacting with the enhanced system; however, subjects using the original system have better recall of the actions they took.

Introduction

Current research on the next generation of Intelligent Tutoring Systems (ITSs) explores the usage of Natural Language (NL) as one of the keys to improving ITSs—see the work at the CIRCLE1 center (http://www.pitt.edu/~circle/), or the AutoTutor system (Graesser et al. 2000). Some projects aim at providing ITSs with a full-fledged dialogue interface, e.g. (Hume et al. 1996; Moore, Lemaire, & Rosenbloom 1996; Rosé, Di Eugenio, & Moore 1999; Freedman 1999; Graesser et al. 2000). We are following suit but with less ambitious goals. Rather than building a full-fledged dialogue interface, we set out to rapidly improve the language feedback provided by an existing ITS shell. In this project, we concentrate on sentence planning, and specifically, on sentence aggregation.

We took this approach for two reasons. First, we would like to understand what can be accomplished by interfacing an NL generator to an ITS taken as a blackbox. Efficiency and rapid prototyping are among the reasons we chose EXEMPLARS (White & Caldwell 1998) as our sentence planner. Second, this approach affords us the opportunity to evaluate the effect of sentence planning on the ITS effectiveness, because we can compare the ITS in its original form and in its enhanced form.

We will first discuss DIAG, the ITS authoring shell we are using, the kinds of feedback it produces, and the aggregation rules we implemented within EXEMPLARS. We will then discuss the results of the formal evaluation we conducted. We will conclude with a discussion of related approaches and of future work.

DIAG and sentence aggregation

DIAG (Towne 1997a; 1997b) is a shell to build ITSs that teach students to troubleshoot complex artifacts and systems, such home heating and circuitry. DIAG in turn builds on the VIVIDS authoring environment (Munro 1994). VIVIDS based tutors deliver instruction and practice in the context of graphical simulations. Authors build interactive graphical models of complex systems, and build lessons based on these graphical models.

A typical session with a DIAG application presents the student with a series of troubleshooting problems of increasing difficulty. DIAG’s tutoring strategy steers the student towards performing the tests that have the greatest potential for reducing uncertainty (Towne 1997a). Most of the times, a test consists of the visual observation of an indicator. DIAG keeps track of the tests the student performs, and the inferences that could be made from the symptoms shown. The student interacts with the application by testing indicators and trying to infer which faulty part (RU) may cause the detected abnormal states. RU stands for replaceable unit, because the only course of action open to the student to fix the problem is to replace faulty components in the graphical simulation. Figure 1 shows one of the graphical views in a DIAG application that teaches how to troubleshoot a home heating system. The subsystem being displayed is the furnace system. Some of its components are indicators (e.g., the gauges labeled Burner Motor RPM and Water Temperature). Others are either replaceable units, or other complex modules that contain indicators and replaceable units, e.g. the Oil Burner. Complex components are in turn zoomable, i.e., if the user clicks on them a new view that reveals their inner components is shown.

At any point, the student can consult the built-in tutor in one of several ways. For example, if the student suspects an RU to be faulty, s/he can ask the tutor to specify the likelihood that this part is the cause of the fault. The tutor will also indicate the state of any indicators that the student has
Figure 1: A screen from a DIAG application on home heating

explored and try to imply a correlation, positive or negative, between the states of the indicators to the RU in question. By utilizing the tutor’s feedback, the student can deduce relationships among the system parts and continually refine his/her solution.

After deciding which content to communicate, the original DIAG system uses very simple templates to assemble the text to present to the student. The result is that the feedback that DIAG provides is repetitive, both as a sequence of replies to requests for feedback, and within each verbal feedback. In many cases, the feedback presents a single long list of many parts. This problem is compounded by the fact that most DIAG applications involve complex systems with many parts. Although there are different levels of description in the system model, and hierarchies of objects, the verbal feedback is almost always in terms of individual indicators or units. The top part of Figure 2 shows the reply originally provided by DIAG to a request of information regarding the indicator named “Visual Combustion Check”.

We set out to improve on DIAG’s feedback mechanism by applying aggregation rules. For example, a long list of parts can be broken down by classifying each of these parts in to one of several smaller lists and then presenting the student with this set of lists.

The bottom part of Figure 2 shows our aggregation rules at work. The revised output groups the parts under discussion by the system modules that contain them (Oil Burner and Furnace System), and by the likelihood that a certain RU causes the observed symptoms. Notice how the Ignitor Assembly is singled out in the revised answer. Among all mentioned units, it is the only one that cannot cause the symptom. This fact is just lost in the original answer.

**DIAG and EXEMPLARS**

As our sentence planner, we chose EXEMPLARS (White & Caldwell 1998) over better known systems such as FUF (Elhadad 1993) and Penman (Bateman 1994) because of the complexity and learning curve of the latter two.

EXEMPLARS is an object-oriented, rule based generator. The rules (called exemplars) are similar to schema-like text planning rules because they are meant to capture an exemplary way of achieving a communicative goal in a given communicative context, as determined by the system designer. EXEMPLARS is a hybrid system that mixes template-style and more sophisticated types of text planning. The text planner selects rules by traversing the exemplar specialization hierarchy. The applicability conditions associated with each exemplar are successively evaluated in or-
The visual combustion check is igniting which is abnormal in this startup mode (normal is combusting).

- Oil Nozzle always produces this abnormality when it fails.
- Oil Supply Valve always produces this abnormality when it fails.
- Oil pump always produces this abnormality when it fails.
- Oil Filter always produces this abnormality when it fails.
- System Control Module sometimes produces this abnormality when it fails.
- Ignitor assembly never produces this abnormality when it fails.
- Burner Motor always produces this abnormality when it fails.

And maybe others affect this test.

The Visual combustion check indicator is igniting which is abnormal in startup mode. Normal in this mode is combusting.

Within the Oil Burner:
- These replaceable units always produce this abnormal indication when they fail:
  - Oil Nozzle
  - Oil Supply Valve
  - Oil Pump
  - Oil Filter
  - Burner Motor

The Ignitor assembly replaceable unit never produces this abnormal indication when it fails.

Within the Furnace System:
- The System Control Module replaceable unit sometimes produces this abnormal indication when it fails.

Also, other parts may affect this indicator.

Figure 2: Original (top) and revised (bottom) answers provided by DIAG to the same Consult Indicator query.
order to find the most specific exemplar for the current context. Because EXEMPLARS rules are schematic templates, they offer several practical advantages such as simplified development, runtime decision efficiency, and reduced demands on knowledge acquisition and representation (White & Caldwell 1998). Additionally, as EXEMPLARS is a fully object oriented framework, built on top of Java 1.1, it provides the developer with features inherent in object-model design such as extensibility and decoupling of the text planner from the domain application.

The communication between DIAG and EXEMPLARS is achieved using VNET communications, a TCP/IP-based client-server layer that is built into VIVIDS. The DIAG application is the server, and the client is a separate Java-based application that both communicates across a socket and that contains the EXEMPLARS rule base and feedback generator. This intercommunication is needed to correctly synchronize the flow of DIAG events.

In a VIVIDS based system, events are packets of procedural code that are activated by external triggering conditions. A student query triggers an event. Within this event, DIAG collects the information it needs to communicate to the student. In the original system, all the information is concatenated in a single string. In the DIAG-NLP system, this information is written to a text file that is then passed to EXEMPLARS. The text file contains one or more packets of information, structured as follows:

```xml
<DIAG event-name> <Indicator|ReplUnit> <attribute1-name> <attribute1-value> . . . <attributeN-name> <attributeN-value>
```

A portion of the text file that DIAG passes to EXEMPLARS for the example in Figure 2 is as follows:

```xml
ConsultIndicator Indicator name Visual combustion check state igniting modeName startup normalState combusting -- -- ConsultIndicator ReplUnit name Oil Nozzle fufer always -- -- ConsultIndicator ReplUnit name Ignitor assembly fufer no effect -- -- ConsultIndicator ReplUnit name System Control Module fufer sometimes
```

The attribute fufer² represents the strength of the causal connection between the failure of an RU and an observed symptom. The order of the information packets in the text file mirrors the order in which DIAG assembles the information, which is also directly mirrored in the feedback the original DIAG system provides (see top of Figure 2).

As we had planned, we only minimally modified DIAG. The lines of code within the event that would generate strings in the text window are replaced with a call to the appropriate VNET function. Specifically, we modified four DIAG events, ConsultIndicator, ConsultReplUnit, ConsultDebrief, ConsultShowSuspRUs. The latter two are invoked after the subject has solved the problem, to get a “critique” of their troubleshooting strategy. The latter two are also fully functional, but as we did not collect metrics about them, we will not discuss them further.

Based on the message sent by DIAG, EXEMPLARS performs essentially three tasks:

1. it determines the specific exemplars needed;
2. it adds the chosen exemplars to the sentence planner as a goal;
3. it linearizes and lexicalizes the feedback in its final form, writing it to an external file.

In our current implementation, EXEMPLARS generates answers to all queries regarding indicators and RUs. DIAG still directly generates other textual feedback that did not appear as problematic, such as the initial statement of the problem.

EXEMPLARS needs to know the DIAG event that invokes it, but the text file EXEMPLARS receives from DIAG does not contain any directive regarding how to aggregate the information and how to format the text. After EXEMPLARS has produced its feedback, it writes it to another text file. Other code on the client side will trigger the appropriate DIAG event, that reads the new text file and appends it to the DIAG text window line by line (the text file produced by EXEMPLARS contains all formatting instructions) — see the bottom part of Figure 2.

### The rule base

The EXEMPLARS rule base being used with the home heating application was designed to be small and flexible. Recall that in EXEMPLARS rules, i.e., exemplars, are organized in an inheritance hierarchy. Exemplars inherit methods from their ancestors. The inheritance hierarchy is used as a decision tree, that the generator traverses until it finds the most specific rule that can be applied, and passes it to the text planner.

The rule base consists of description rules and aggregation rules.

Description rules are used when the full description of a part is required, such as whether the part is in a normal state, its current reading, and, if abnormal, what the normal state should be (see the first sentence in the bottom part of Figure 2). There are four description rules, the generic DescribePart, and its three daughters DescribeIndicator, DescribeReplUnit, DescribeEmptyList. The latter is used to generate the final clause for the ConsultDebrief event.

The aggregation rules are used to classify large lists of parts into a set of smaller lists. There are eight rules for aggregation. The topmost is AggByType, which controls nest-

²The name comes from DIAG and its origin is unknown to us.
ing of embedded lists. \textit{AggByType} has 6 daughters, among them:

- \textit{AggByContainer}: each part within this DIAG application is contained within a larger block, called a system module. The \textit{AggByContainer} rule accepts a list of parts, classifies each part by its containing module, and then creates a set of lists by module;
- \textit{AggByFufer}: it groups replaceable units according to the likelihood of being at fault for a specific symptom;
- \textit{AggByState}: it groups indicators by their normal / abnormal state.

The eighth aggregation related rule, \textit{Agg}, is not part of this subhierarchy, but is invoked by the other aggregation rules, because they do not directly create textual feedback. \textit{Agg} can accept a singleton list as well as a regular list. This rule also deals with formatting, namely, creating vertical lists, spacing, etc.

Aggregation rules are designed to allow composite aggregation. This has the effect of creating nested lists. As an example, a part can be aggregated by its type (indicator/RU), and then by its fault likelihood, and finally by its containing module. This type of composition would appear in the feedback as a three-level nested list. In practice, though, deep nesting of aggregation rules does not occur, because DIAG always discusses only one indicator or one replaceable unit at a time. As demonstrated in the bottom part of Figure 2, the most frequent application of the aggregation rules is to group parts according to the system module they belong to, and within each module, to group replaceable units by how likely it is they may cause the observed symptom.

To give the reader a rough idea of how often the rules are invoked, consider the means of event occurrences from Table 1. In our experiments, there were roughly 6 indicator consultations and 18 RU consultations over 3 problems, namely, 2 indicator consultations and 6 RU consultations per problem (i.e., dialogue). Thus, the \textit{DescribeIndicator} and \textit{DescribeReplUnit} rules were invoked 2 and 6 times, respectively; the \textit{AggByContainer} and \textit{Agg} exemplars were invoked 6+2=8 times per dialogue.

Note that in this experiment, morphology, lexical realization and referring expression generation were all treated ad hoc, i.e., they were directly encoded in the appropriate exemplars. We will address some of these problems in future.

**Results**

Every student solved all the problems (DIAG gives them the option \textit{I give up}, but nobody used it). Thus, at the level
of problems solved, there is no difference between the two groups.

Tables 1, 2, 3 show the results for the measures we computed; the measures in boldface are those that are statistically significant, or at least marginally significant, and they will be discussed more in detail below. Note that these are cumulative measures across the three problems; we did compute the same measures for each problem, and the results for individual problems are similar to the cumulative ones we discuss here.

On the whole, Tables 1 and 3 show some small differences in favor of DIAG-NLP, although most of them are not statistically significant. We consider the lower number of indicator consultations in DIAG-NLP as evidence in favor of the effectiveness of the aggregated feedback: because the feedback is better and highlights what is important (such as that the Ignitor Assembly can never cause the Visual Combustion check to ignite, see Figure 2), the subjects can focus their troubleshooting without asking as many questions of the system, at least as regards indicators. However, Table 2 shows that subjects in DIAG-orig remember what they did better than those in DIAG-NLP (recall that the measures regarding indicators concern the indicators a subject consulted the system about, whereas the measures regarding RUs concern the RUs the subject replaced). All in all, this is a puzzling result, especially because subjects in DIAG-orig consult the system on indicators more often than subjects in DIAG-NLP, thus we would expect them to have more problems remembering what they did. Note that precision and recall are much higher for RU replacement than for indicator consultations, as would be expected because there are many more indicator consultations than RU replacements.

### Statistical significance of results.

Table 4 reports the measures that have or approach statistical significance in boldface. For each measure, we report the mean of the distribution, the standard deviation, and the results of several tests. Two tests (Shapiro-Wilk and Levene) are used to check whether the assumptions of the t-test are respected: whether the distribution is normal, and whether the two distributions have the same variance, respectively. The null hypothesis that the distribution is normal and that the variances are equal are respectively rejected when Shapiro-Wilks and Levene are statistically significant (p<.05). If the first assumption is not respected, the t-test is not applicable, and a non-parametric test such as Mann-Whitney can be applied (we report the results for Mann-Whitney for every measure anyway). If the variances cannot assumed to be equal, a variant of the t-test (Welch’s test) can be applied (in practice, the two tests have very similar results even when the variances are not equal) — in Table 4 the t-test column reports the appropriate kind of t-test, according to the results of the Levene’s test.

Table 4 shows that only two measures achieve real statistical significance, precision and F-measure, both regarding indicators. Note however that precision and recall for indicators are very poor for both groups (see Table 2). The measure indicator consultations exhibits a non-significant trend in the predicted direction. We also report statistics computed on the log of indicator consultations (taking the log often makes the distribution normal, as in this case). Both the t-test and the Mann-Whitney test remain at the same significance level, p=0.11.

Possibly, the most general conclusion to be drawn from Table 4 is that most of our distributions are not normal and that their variances are not equal. Regarding the first point, it is likely to occur because of the small number of subjects. The difference in variance is in our opinion more interesting. In the DIAG-NLP group, for almost all measures the standard deviation is lower than for DIAG-orig, in some cases significantly so, as shown in Table 4. In those few cases (RU precision, RU recall and RU F-measure) in which the standard deviation for DIAG-NLP is higher than for DIAG-orig, the difference is at most .02. Although these differences may be due just to sampling error, they can also be explained by the fact that a system that allows a more natural interaction engenders a more homogeneous behavior among subjects. As mentioned, the two groups are both composed of science or engineering majors. DIAG-orig includes 7 undergraduates, 9 graduates, and one research staff, DIAG-NLP 6 undergraduates, 9 graduates, and 2 research staff.\(^3\) One

---

\(^3\)We asked them for their SAT / GRE scores, but as few subjects provided them, we cannot use these scores to test for differences in level.
confounding variable could be the mastery level of English on the part of subjects. We set out to test only native or near-native speakers, but we did not find enough subjects, so we had to relax this constraint. Thus, we recomputed all the measures we discussed here on a reduced pool of subjects. 13 in the DIAG-orig group and 15 in the DIAG-NLP group. Six subjects were eliminated because their English was judged to be too poor by the experimenter. However, the results discussed hold across the reduced subject pool as well.

### Related work

There are two areas of research related to the work presented in this paper: NLG and evaluation of NLG systems, and NL interfaces to ITSs, and their evaluation.

(Reiter & Dale 1997) define sentence aggregation as the process of grouping messages together into sentences. They note that types of sentence aggregation include conjunction, ellipsis, set formation, and embedding. The work we have described in this paper mainly falls within set formation, like e.g. (Dalianis 1996; Huang & Fiedler 1996; Shaw 1998). We are not including embedding (Scott & Sieckenius de Souza 1990; Cheng & Mellish 1997) in our discussion. For a recent and comprehensive review of work in aggregation, see (Reape & Mellish 1998).

As (Reiter & Dale 1997) points out, the most difficult aspect of aggregation is to decide which of the numerous potential aggregations should be performed. Our choice was constrained by relating our aggregation rules to the structure of the domain and the tutoring strategy embodied in the system in question (DIAG).

The most common problem in all the work on aggregation, including ours so far (but see below), is that aggregation rules and heuristics are plausible, but are not based on any hard evidence. For example, in his PhD thesis (1996) Dalianis bases his aggregation rules and heuristics on a corpus study, however his corpus study is not convincing. Aggregation rules could be posited from a corpus only if we knew the underlying representation the text had been aggregated from. Unfortunately, we do not have access to the mental representations of those who produced the text. Although assuming a representation such as a first-order logic may be a sound move from the point of view of system implementation, it does not really answer the question. (Shaw 1998) concentrates on generating coordinated constructions. He shows how his algorithm can produce various linguistic constructions, but as for Dalianis, the grounds on which the specific rules are based is unclear, and no evaluation is included.

A system that uses templates like EXEMPLARS and that is intended to be used for tutoring dialogues is YAG (McRoy, Channarukul, & Ali 1999). According to McRoy et al., using templates, as opposed to e.g. unification with a grammar, provides a fast mechanism for surface generation. However, a problem with templates is that they potentially proliferate. The inheritance mechanism in EXEMPLARS partly prevents this problem; however, we suspect such proliferation occurs in YAG, because it uses no inheritance mechanism.

Evaluation is still a major unresolved issue for text generation systems (Dale & Mellish 1998; Dale, Di Eugenio, & Scott 1998) and more in general for dialogue systems (Walker & Moore 1997). Text generation systems have been evaluated by using human judges to assess the quality of the texts produced (Lester & Porter 1997; Chu-Carroll & Carberry 1998); by comparing the system’s performance to that of humans (Yeh & Mellish 1997); and indirectly through task efficacy measures (Young 1997). The major stumbling block for progress is determining what metrics and methods should be used: for example, how can the quality of an output text be measured?

As regards NL interfaces to ITSs, we discussed relevant projects in the introduction. The evaluation of NL interfaces to ITSs is another area that needs investigation. ITSs are often evaluated in terms of pre/post-test score, however other measures such as some of the metrics we have discussed earlier (time on task, recall of performed actions) may be appropriate as well. To our knowledge, the only ITSs with an NL interface which has been formally evaluated is CIRC-SIM (Evens et al. 1993; Kim, Glass, & Evens 2000), but the results of the evaluation are not available yet. We are not aware of anybody who has tried to do what we have pre-

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Values for DIAG-orig</th>
<th>Values for DIAG-NLP</th>
<th>Shapiro-Wilks</th>
<th>Levene</th>
<th>t-test</th>
<th>Mann-Whitney</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consultations</td>
<td>11.4, 11.2</td>
<td>5.9, 4.0</td>
<td>.75, p=.01</td>
<td>5.47, p=.03</td>
<td>—</td>
<td>98, p=.11</td>
</tr>
<tr>
<td>Consultations (log)</td>
<td>.32, 1.10</td>
<td>.52, 9.35</td>
<td>.91, p=.09</td>
<td>1.62, p=.11</td>
<td>98, p=.11</td>
<td></td>
</tr>
<tr>
<td>Parts Replaced</td>
<td>11.4, .92</td>
<td>.91, p=.09</td>
<td>1.83, p=.09</td>
<td>2.51, p=.02</td>
<td>—</td>
<td>133, p=.76</td>
</tr>
<tr>
<td>Indicator Precision</td>
<td>.48, p=.01</td>
<td>.58, p=.01</td>
<td>4.97, p=.005</td>
<td>—</td>
<td>—</td>
<td>135, p=.76</td>
</tr>
<tr>
<td>Indicator Recall</td>
<td>.92, p=.23</td>
<td>.95, p=.48</td>
<td>3.35, p=.004</td>
<td>2.19, p=.04</td>
<td>95, p=.09</td>
<td></td>
</tr>
<tr>
<td>Indicator F-measure</td>
<td>.86, p=.02</td>
<td>.93, p=.25</td>
<td>0.73, p=.40</td>
<td>—</td>
<td>93.5, p=.08</td>
<td></td>
</tr>
<tr>
<td>Indicator Consultations</td>
<td>.58, p=.01</td>
<td>.95, p=.48</td>
<td>4.33, p=.047</td>
<td>2.51, p=.02</td>
<td>58, p=.08</td>
<td></td>
</tr>
</tbody>
</table>

| Table 4: Measures with or approaching statistical significance |

```
sent in this paper, to compare two different versions of the same ITS that differ only with respect to the language they produce.

**Discussion and Future Work**

We had two goals in this work: to ascertain whether it is possible to quickly improve the language feedback an ITS provides, and if yes, whether producing better language improves the learner’s experience. We think we achieved the first goal; as regards the second goal, we collected some positive evidence, but not as strong as strong as we would have liked.

Regarding quickly augmenting an ITS with simple sentence planning, we did indeed do so. Although our rule base is tailored to our specific domain, it is portable to any DIAG application, because all DIAG domains are strongly hierarchical, and all DIAG applications are built in terms of indicators and RUs.

In fact, we argue that our rule base would be easily portable to other ITs, as long as their domains are strongly hierarchical. The fact that EXEMPLARS represents rules as a class hierarchy lends itself well to the idea of using natural language rules for describing the objects of modeled systems. The designer can begin with a rule to describe a generic object and extend it to the most specific level necessary. The larger the rule class hierarchy and more specific the rule class definitions, the smaller and more specific are the associated text templates.

The disadvantage of the rule class hierarchy approach used by EXEMPLARS is that it is likely to break down as the complexity of the text increases. The designer may find that either the rule class hierarchy proliferates unnecessarily or that the most specific rules become complex with code. In general, the developers of EXEMPLARS claim the system to be very flexible, because its rules can be used at every level of a natural language generator, including intentional/rhetorical. At this point, it is unclear to us whether EXEMPLARS can be used indeed employed for more sophisticated types of text planning, that include a more complex decision making process, co-constraining goals, etc. We have started experiments in this regard. We are trying to push EXEMPLARS on simple kinds of discourse planning, such as that necessary to generate from shallow RST trees (Mann & Thompson 1988), and to take global focus into account.

Our plans for the medium and long term also include:

- To conduct a constrained data collection that will help us find some empirical foundations for the proposed aggregation rules. We intend to collect tutoring dialogues between a student interacting with a DIAG application and a human tutor that provides the answers to the queries the student asks of DIAG. As much as possible, we would like to constrain the tutor to provide feedback that includes only the facts that DIAG would have communicated at that specific moment.

- To improve realization, which was achieved in an ad hoc manner in the current system. We will be using a different version of EXEMPLARS which is integrated with RealPro (Lavoie & Rambow 1997), a surface realizer developed at CogenTex like EXEMPLARS.

- To better understand and incorporate references to the graphical model and layout in the language. In our current implementation, formatting is generated by the Agg rule according to our intuitions. However, layout deserves more attention. Layout performs functions similar to intonation in spoken language, namely, it can convey meanings beyond those directly related to the text (Wright 1997).

- To model global focus within DIAG. At the moment, each reply from DIAG is built independently from the previous ones, and no discourse focus is maintained. It is thus possible (and in fact happens in applications which have more components than the home heating system) that one reply talks about some RUs, and the next reply does not make any reference to those RUs just mentioned. This will also help us understand what the real limits of EXEMPLARS are.

Like many of the projects mentioned in the introduction, our ultimate goal is to devise a dialogue architecture that can be easily interfaced to ITs. However, we will pursue this goal by first addressing the specific communication needs of an ITS built within the VIVIDS/DIAG ITS authoring frameworks.

**Acknowledgements.** This work is supported by grant N00014-99-1-0930 from the Office of Naval Research, Neural and Cognitive Science Division. We are grateful to CogenTex Inc., in particular to Mike White, for making EXEMPLARS available to us.

**References**


