

Co-generation of Text and Graphics

David G. Novick and Brian Lowe

Department of Computer Science

The University of Texas at El Paso

El Paso, TX 79968-0518

+1 915-747-5725

novick@utep.edu

ABSTRACT

To reduce potential discrepancies between textual and graphical content in documentation, it is possible to produce both text and graphics from a single common source. One approach to co-generation of text and graphics uses a single logical specification; a second approach starts with CAD-based representation and produces a corresponding textual account. This paper explores these two different approaches, reports the results of using prototypes embodying the approaches to represent simple figures, and discusses issues that were identified through use of the prototypes. While it appears feasible to co-generate text and graphics automatically, the process raises deep issues of design of communications, including the intent of the producer of the documentation.

Categories and Subject Descriptors

I.7.2 [Document and Text Processing]: Document Preparation – Languages and system, multi/mixed media.

General Terms

Documentation Management

Keywords

Single source, aviation.

1. INTRODUCTION

Inconsistencies have long been recognized as a major problem for developers of documentation and have served as a driver of the movement toward single sourcing. In some cases, inconsistencies in documentation have been serious enough to pose a hazard to safety. For example, National Transportation Safety Board (NTSB) found that an-flight incident in a DC-9 commercial jet aircraft was caused by a discrepancy between text and graphics in a maintenance manual. The NTSB's Final Report stated that:

Shortly after an intermediate level-off at 13,000 feet, the crew heard a loud bang and felt the airplane shudder. They diverted to a nearby airport. ... An investigation revealed broken and bent thrust reverser linkages. A pivot bolt,

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.

SIGDOC'05, September 21–23, 2005, Coventry, United Kingdom.

Copyright 2005 ACM 1-59593-175-9/05/0009...\$5.00.

washers, castellated nut, and cotterpin associated with a driver linkage arm were missing. Two other company airplanes were found with a missing cotterpin from the pivot bolt. Examination of these assemblies revealed the bolt end did not protrude beyond the nut's outer edge. The result was that the cotterpin could not fit through the hole in the bolt. The manufacturer's parts manual and maintenance manual text indicated that one countersunk washer should have been below the bolt head and one flat washer under the nut. However, the corresponding illustration in the maintenance manual showed an additional washer (2 washers) under the nut. [8]

A section of relevant page from the manual [9] shows, at the upper left, the bolt with the two washers just before the nut and the cotter pin.

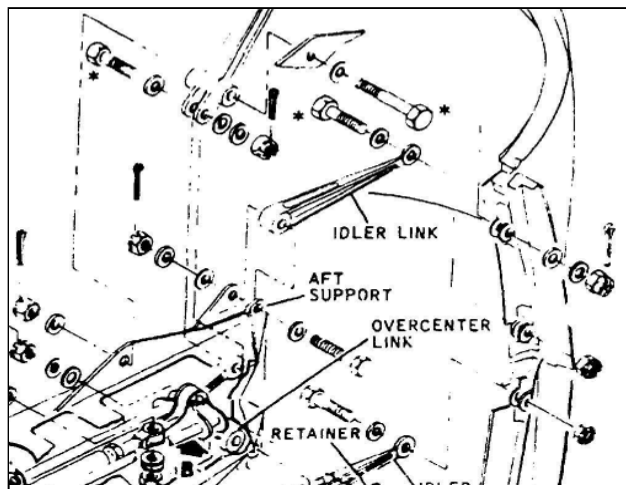


Figure 1. Detail of DC-9 Maintenance Manual

In contrast, the manual's textual procedure for "Removal/Installation Thrust Reverser Door Actuating Linkage" instructed "(4) Install bolt that connects driver link to overcenter link with bolt head inboard. One countersunk washer is installed under bolt head and one plain washer under nut." [9] As a result of this inconsistency, the NTSB's Cause Report cited the manufacturer's unclear information regarding the type and number of washers to be installed on the pivot bolt assembly as a factor relating to the accident.

Such inconsistencies cannot be addressed by relying solely on text or on graphics. Normally, both are needed to convey an adequate understanding of a system. Graphics provide context

needed to understand text [4, 1], and ambiguities in graphics can be explained in text [1].

This paper begins to address inconsistencies between text and graphics in the preparation of documentation by exploring two approaches to producing both text and graphics from a single common source. One approach to co-generation of text and graphics uses a single logical specification; a second approach starts with a graphical representation created in a computer-aided design (CAD) system and produces a corresponding textual account. The paper reports the results of using prototypes embodying the approaches to represent simple figures and discusses issues that were identified through use of the prototypes. Both prototypes successfully produced consistent text and graphics in simple cases but the experience of developing and using the prototype raised deep issues of design of communications, including the intent of the producer of the documentation.

2. RELATED WORK

This work grows from three related lines of research. The first, as published in prior papers at SIGDOC, focused on consistency in documentation. The principal goal of this work was to assure that information in, for example, operating manuals, used terminology consistently, and was expressed via coherence maxims [6]. One aspect of coherence involved integrity of consistency and differentiation in referring to domain entities, actions and relations. The consistency aspect was expressed in Maxim 2, which held that references to the same thing should appear the same, promoted coherence of reference.

Based on the coherence maxims, tools were developed for creating consistent text, including COHERE [6] and DSTOP [7]. These tools were designed to automate much of the process of checking for consistency among technical terms, thus aiding authors in creating reliable documentation. However, both COHERE and DSTOP worked only for text. The problem of consistency between text and graphics was noted [7] as a problem that was beyond DSTOP's reach. Consequently, providing authoring tools that assure consistency between textual and graphical representations remained an open issue.

A second related line of research involves generation of text from a logical description: Other researchers investigated methods of automatically generating documentation content from a model of the system to be documented. In particular, the DiDoLog prototype [2] showed that a partial model of a system, such as that of an aircraft brake system, could be used to generate a textual description guaranteed to be consistent and complete with respect to the partial model. Another approach to the automated generation of text for procedures was developed by Paris et al. [10], who created a tool called DRAFTER for the generation of instructions in multiple languages. Using a knowledge editor, the author created an abstract, language-independent representation of the underlying system, including the steps to be taken by the user. From this specification, DRAFTER could produce instructions in both French and English.

The third related line of research involves the relationships between graphics and text, with generation of graphics from text or an abstract description. Illustrations adapted to the contents of small text segments were successfully generated in the domain of human anatomy [4]. In this approach, a natural-language

understanding module identified the referring expression in the text, and then selected for emphasis corresponding elements from a library of illustrations. This effort was in many ways more sophisticated than that pursued here, particularly with respect to analysis of text and in the generation of complex graphics. For example, the system would choose a camera angle best suited to make visible the diagram's elements of greatest interest. However, the graphics were essentially pre-drawn, with the program determining which graphics to select and emphasize. Thus the system had no inherent protections against inconsistency. Any particular graphic's actual relevance was dependent on the correctness of the developers' linking the concepts and illustrations.

A similar issue arose in a system [1] that generated referring expressions from an abstract description of the application domain. The system could produce appropriate co-references for mixed text and graphics but assumed that elements designated as co-referent were representations of the same world object. The system did generate natural-language referring expressions, but did so with prepared graphics. Again, this lacked a consistency check: there was no inherent assurance that the prepared graphic was, in fact, consistent with the corresponding logical representation used in generating the text. In concrete terms, this means that a diagram could have included two washers when the logical description from which the corresponding text was generated included only one.

The problem addressed in this paper, then, involves the intersection of the three related lines of research: generating both text and graphics in a way that guarantees consistency between them. This could be done either by having a single, common source from which both the text and graphics are generated, by interpreting a given text and generating the corresponding graphics, or by interpreting a given set of graphics and generating the corresponding text. The present study was predicated on the observation that going from text to graphics would be by far the most difficult approach, both because it involved unrestricted parsing and interpretation of written language and because the text would likely under-specify the graphics. The text-to-graphics approach has been tried (e.g., [3]), but required, even for simple problems from an introductory text book on physics, a system of great complexity and did not appear to scale. Moreover, a written description of graphical objects would not probably contain all of the information needed to generate a corresponding image, information such as dimensions, orientation, placement, colors, textures and so forth needed for realistic applications. Parsing a purely graphical source presents complementary problems.

Consequently, this paper explores the feasibility of the remaining approach: generating both text and graphics from a common source. This is done two ways: from a common logical specification and from a common model produced in a CAD program when drawing the graphics.

As the main goal was to assess feasibility, the text and graphics generated were simple. In contrast, the language produced by DiDoLog and especially DRAFTER were much more sophisticated. The conclusion section of this paper discusses prospects for increasing the complexity of the graphics and the sophistication of the text generated. For the present, the study worked with "blocks world" examples using an arrangement of contiguous cubes and simple declarative sentences in English.

Figures 2a and 2b depict configurations of boxes that were contemplated as kinds of diagrams that the prototypes would be able to handle.

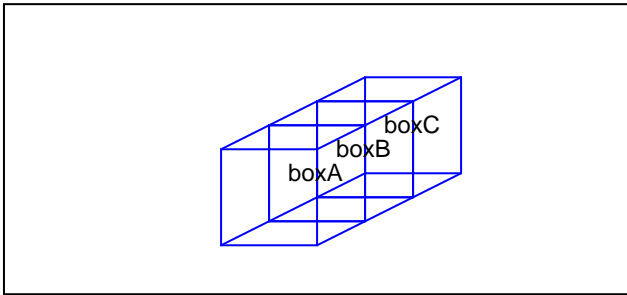


Figure 2a. Example of Target Graphics

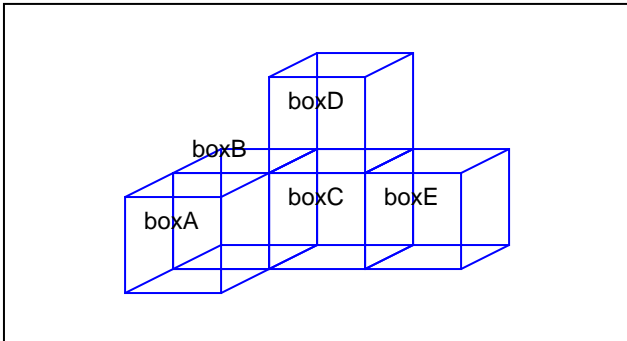


Figure 2b. Example of Target Graphics

Optimally, the prototype programs would produce text corresponding to the objects in Figure 2a along the lines of “Box A is front of Box B, and Box B is in front of Box C.” The more complex object in Figure 2b might be described as “Box B is behind Box A, Box C is right of Box B, Box D is on top of Box C, and Box E is right of Box C.” Other textual descriptions of these objects would be plausible; the conclusion section discusses these possibilities. In any case, the text and graphics for relatively simple descriptions form the set of test cases around which the prototypes were developed and on which the prototypes were tested. The next two sections describe the development and testing of the two prototype co-generation systems.

3. LOGICAL SOURCE

The first prototype system for co-generation of text and graphics implemented an approach where both the text and the graphics were generated from a single, common logical source. In this implementation, both the specification of the objects and the program itself were written in Prolog.

The objects in the prototype were boxes, whose dimensions were specified numerically but whose locations were specified in terms of relations to the other objects. So, for example, a box named “boxA” with depth 50, height 100 and width 100 would be represented as

```
object(box,boxA,loc(_X,_Y,_Z),size(50,100,100)).
```

The “loc” variables are uninstantiated, as the program will figure out this box’s 3D location from its relationships with other objects in the diagram to be represented. The relationships representable in the prototype were *on*, *under*, *left_of*, *right_of*, *behind* and *in_front_of*. With the defined objects and this set of relations, the

author can specify a set of states, relating the objects to each other, from which the text and graphics are generated. For example, the diagram in Figure 2a might be represented with three objects, “boxA”, “boxB” and “boxC” and these states:

```
state(in_front_of,boxA,boxB).
state(in_front_of,boxB,boxC).
```

The prototype takes these steps to go from the specification to the text and the graphics:

- Create a list of the unique objects included in the set of states. This was done for two reasons. First, any object might be included in more than one state. In the example above, object “boxB” is included in both states. Second, it is possible that the author declared an object, perhaps a box named “boxD”, but did not include it in any state; this means that the prototype could not simply rely on the set of declared objects.
- Place the objects in 3-dimensional space, instantiating each object’s location variables. This was done by sorting the objects, using the relation information in the states, into an ordered list of objects going from the object at the front left bottom of the diagram to the back right top of the diagram. All of the coordinates for the locations of the objects were expressed relative to the coordinates of the first object, for which an arbitrary location was fixed.
- Generate the text corresponding to the set of states, and display this in the Prolog interpreter’s interactions window. This was done simply, going from the set of states to directly transliterated text. In the example above, the states

```
state(in_front_of,boxA,boxB).
state(in_front_of,boxB,boxC).
```

would result in text such as “boxA is in front of boxB, and boxB is in front of boxC.”

- Generate the graphics and display them in a separate graphics window. This was done by sorting the objects from back to front, bottom to top, and left to right, to prepare for dealing with hidden lines; projecting the 3-D coordinates into 2-D coordinates in the drawing space; and drawing each object by first hiding lines that it would obscure and then drawing the lines of the object itself.

The prototype was tested with specifications at levels of complexity spanning the examples in Figure 2. For example, with six boxes 100x100x50 the specification

```
state(on,boxB,boxC).
state(on,boxA,boxB).
state(left_of,boxD,boxB).
state(right_of,boxE,boxC).
state(right_of,boxF,boxA).
```

the prototype produced the text

```
boxB is on boxC, boxA is on boxB, boxD is left
of boxB, boxE is right of boxC, and boxF is
right of boxA.
```

and the diagram shown in Figure 3. In a second example, with six boxes 100x100x100, the specification

```
state(on,boxB,boxC).
state(left_of,boxA,boxB).
state(behind,boxD,boxB).
state(in_front_of,boxE,boxC).
state(right_of,boxF,boxD).
```

produced the text

boxB is on boxC, boxA is left of boxB, boxD is behind boxB, boxE is in front of boxC, and boxF is right of boxD.

and the graphics in Figure 4.

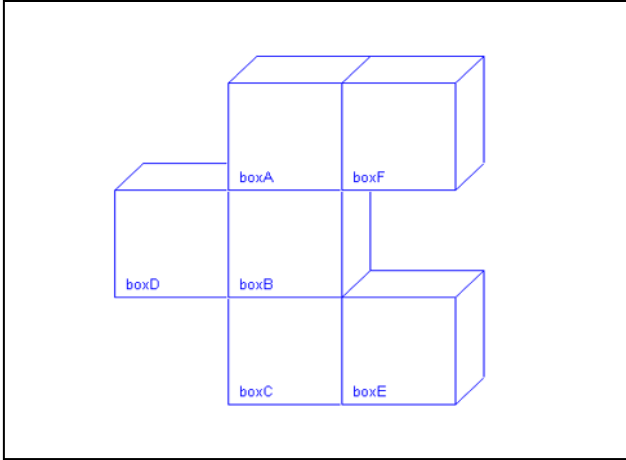


Figure 3. Graphic Produced by Prototype

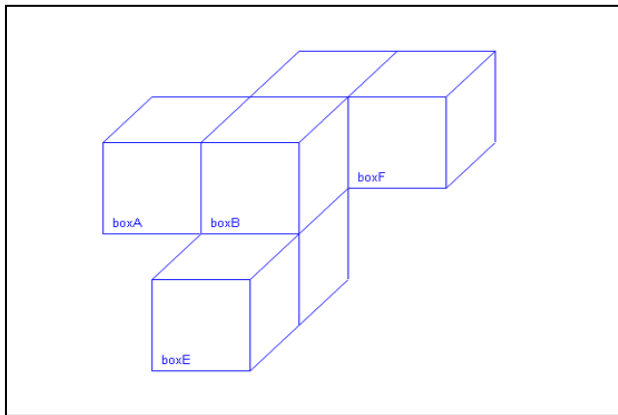


Figure 4. Graphic Produced by Prototype

The development and testing of the prototype suggest that it is feasible to generate both text and graphics of simple diagrams from a single, common, logical specification.

Small changes in the program could produce more sophisticated text. For example, it would be easy to preface an object's name with its type. This would replace text like "boxA is in front of boxB" with text like "Box A is in front of Box B."

A more difficult issue involves the order of the relations presented in the generated text. The prototype currently produces text for the relations in the order in which the relations appear in the specification. While this approach provides a crude measure of authorial control, it could produce disjointed text if the author did not take care with the order of the specifications and seems inconsistent with the philosophy of generating both text and graphics from an abstract specification. For example, a diagram as simple as Figure 2a could be described in multiple ways, including

- Box B is behind Box A, and Box C is behind Box B.
- Box A is in front of Box B, and Box C is behind Box B.

The first version seems to emphasize the linear order of the boxes, and the second version seems to emphasize Box B as a pivotal object.

This issue might be addressed in a number of ways. First, the text could be generated from the set of relations, but inferring an order for them. Second, the text could be generated from the ordered list of objects, similar to the way in which the text is generated in the graphics-as-source approach presented next, except that the task would be easier because the specification already provides information on which objects touch which.

4. CAD MODEL

The second prototype system for co-generation of text and graphics implemented an approach where the graphics were generated by modeling the objects with a computer-aided design (CAD) system and the text generated from a descriptive file produced by the CAD system. In this implementation, the specification of the objects was implicit in the CAD design; the program was again written in Prolog.

Using this approach, the author would use the CAD program to produce the diagram he or she intended, export this graphical representation to a textual specification file, and then run the prototype on the specification file to produce the textual description of the diagram. Figure 5 shows a blocks-world diagram produced using the CAD program.

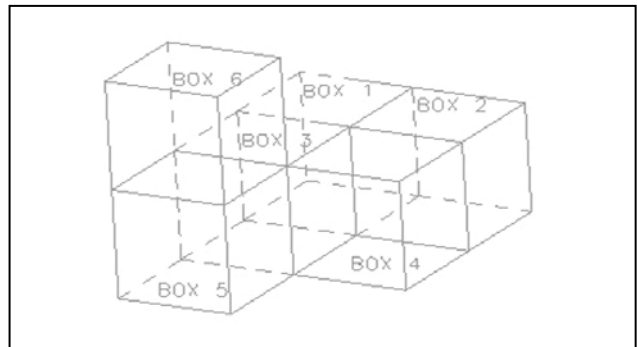


Figure 5. Diagram Produced in CAD Program

Based on this diagram, the author could use the CAD program to export a formal specification of the diagram. This specification is in the form of a text file with many information items, only a few of which (name, location, size) are actually relevant to the display task. For example, the first part of the specification file for the diagram depicted in Figure 5 reads as follows (with some "=" characters elided for better formatting in this paper):

```

=====
Information listing created by : student
Date                          : 5/19/2005
3:16:06 PM
Current work part              :
H:\assdfa.prt
Node name                      : cim21
=====
Information on object # 1

Name                           BOX 1
Owning part                     H:\assdfa.prt
Layer                          1
Type                           solid body
Color                          11 (Orange)
Font                            SOLID

```

The entire specification file produced by the CAD program for the diagram in Figure 5 was 498 lines long. The level of detail provided in such specifications meant that, although it might be surmised that having the graphics already drawn would lead to a simpler program, the CAD-model approach led to a program that was actually longer than the logical-source approach. The first prototype, using the logical-source approach, defined 49 Prolog predicates and was expressed in fewer than 220 lines of code, not including comments. The second prototype, using the CAD-model approach, had about 265 lines of code. The principal factors in this difference are that the second prototype had to parse the specification file and determine the relations among the objects, including determining which objects were adjacent to each other. In the second prototype, there were 49 Prolog grammar productions that parsed the CAD file, and 35 predicate clauses for the rest of the program.

For the diagram in Figure 5, which was generated via the CAD program and produced the 498-line specification file excerpted above, the CAD-model prototype produced the following text:

```
BOX 1 is left of BOX 2, BOX 1 is behind BOX 3,
BOX 2 is behind BOX 4, BOX 3 is left of BOX 4,
BOX 3 is behind BOX 5, and BOX 5 is under BOX 6.
```

Comparison of the figure and the text shows that the text does correspond to the arrangement of the boxes in the diagram. For a simpler example, the CAD program generated the diagram shown in Figure 6. From the 249-line specification file for the diagram, the prototype produced the following, expected text:

```
BOX 1 is in front of BOX 2, and BOX 2 is in
front of BOX 3.
```

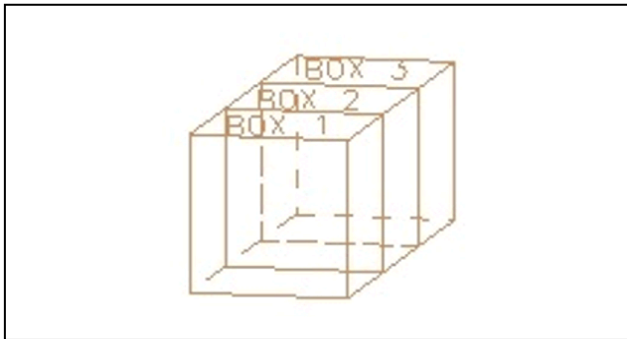


Figure 6. Diagram Produced in CAD Program

As with the logical-source approach, the CAD-model approach also presents issues of the order of the relations presented in the generated text. For example, the diagram in Figure 6 might equally produce text such as “Box 1 is in front of Box 2, and Box 3 is behind Box 2.”

5. CONCLUSION

This paper explored the co-generation of text and graphics from a single source in order to automate consistency in documentation, using two different approaches. One approach used a common logical specification from which both text and graphics were generated. The second approach created the graphical presentation in a CAD system and, from the CAD model, produced a corresponding textual description. The paper reported our experiences in building and testing prototype Prolog programs

for the two approaches to represent simple “blocks world” figures. Both prototypes successfully generated consistent text and graphics for diagrams in the blocks-world domain. While these experiences suggest that it appears feasible to co-generate text and graphics automatically, the process raises deep issues of design of communications, including the intent of the author. In this section, we discuss these issues with a view toward outlining paths for improving the co-generation process.

5.1 Text Issues

Although there are a number of ways to make the prototypes’ production of text more sophisticated, there remain much deeper issues of authorial intent and the role of context in producing the text. As indicated for both prototypes, even in the simplest cases there are multiple ways of representing a diagram in text. For example, from the specification

```
state(in_front_of,boxA,boxB).
state(in_front_of,boxB,boxC).
```

possible texts include:

1. “Box A is in front of Box B, and Box B is in front of Box C.”
2. “Box A is in front of Box B, and Box C is behind Box B.”
3. “Box C is behind Box B, and Box A is behind Box A.”

These appear to be the most intelligible possibilities. Less coherent variations could also be produced along the same lines. But even from the versions with greater coherence, which alternative should one choose to produce? The answer appears to arise from the intent or context view of the author. One way of handling the issue would involve having the author designate one of the component objects as a key object from which the reader’s understanding of the diagram should flow. Choosing “Box A” as the key object might lead to text (1), choosing “Box B” as the key object might lead to text (2), and choosing “Box C” as the key object might lead to text (3). The intuition behind these examples is that the key object serves as a kind of anchor or reference point from which the relations to the other objects are extended. But even in this simple example, going from key object to text remains an intuitive process rather than a well-understood, algorithmic process. For example, if “Box A” is designated the key object, then an equally plausible and perhaps better text would be “Box B is behind box A, and Box C is behind Box B.”

For more complex diagrams the problems grow more difficult. For a diagram such as that in Figure 2b designating a single key object may not be sufficient, as this might lead to a hard-to-understand, long string of conjunctive clauses. And for realistic diagrams, much less the actual DC-9 manual diagram of Figure 1, the problem is far more serious. The diagram may include clusters of related objects that do not touch each other. Or the diagram may be intended to include more than one focal or otherwise important object. In Figure 1, for example, would the key object be the bolt on which the washer is placed? The washer? The cotter pin? All of these? How could the author of original documentation have anticipated that the subsequent focus of an incident investigation would be the placement of the washer? The relevance of all of these questions suggests that progress in automating co-generation of text and graphics depends less on the mechanics of the generation of the text or of the graphics, or even on correctness-assuring means of tying text and graphics, than on ways of understanding and specifying the author’s intent.

5.2 Graphics Issues

The implementation of the prototypes raises a number of issues with respect to graphics part of the co-generation process. Some of these issues are relatively mechanical, and others are more complex. The simpler issues include:

Handling objects of varying dimensions. Both prototypes now function only with boxes of uniform size. While these were sufficient for purposes of demonstrating feasibility, diagrams produced from arrangements of uniform boxes eventually get old. Oddly enough, this limitation arises from different reasons in the two prototypes. In the logical-source prototype, there were issues with the hidden-line function and, more serious, it was not clear what physical layouts were appropriate for objects of different sizes using the available set of relations. For the CAD-model approach, the limitation arose because the part of the program that detected the semantic relations among the objects relied on vertex-matching rather than surface-matching. This could be fixed with some effort, and the results might give insight into the layout problem for the logical-source approach.

Improving the placement of labels. This is primarily an issue for the logical-source approach, where labels are sometimes obscured by objects closer to the foreground. In Figure 4, for example, the labels for Box B and Box D are not visible. The CAD program did a reasonably good job of placing labels in the diagram. This was achieved, however, through the use of wire-frame objects rather than opaque objects, as can be seen by comparing Figures 4 and 5. The solution for the opaque case would involve reasoning about where to place labels that might be obscured. Or, as illustrated in Figure 1, labels might be placed adjacent to the objects. Moreover, as can also be seen in Figure 1, not all objects may need to be labeled.

More difficult challenges include the under-specification of graphics in the single-source approach, and adding additional, more realistic objects.

Remedying the under-specification of the graphics in the single-source approach. Some early progress on this issue was made by Mackinlay [5], who showed how graphs could automatically represent relational information. Data associated with relations, though, are necessarily characterized with certain kinds of information. In more general cases, having sufficient information about the semantics of the data will likely depend on the author. And for presentational aspects of creating a diagram—elements such as point of view, textures and colors that are not specified in the logical source, the author can be free to specify these to the extent that they do not affect the text. Nevertheless, it may be burdensome to create such specifications textually rather than in a GUI, and letting the author change the diagram, post-generation, via a GUI reintroduces the risk of differences between the diagram and the text.

Adding additional, more realistic objects. The prototypes are limited to generating text and graphics for cubes and oblong boxes. The prototypes cannot handle objects of different sizes, much less the variety of graphics required for realistic documentation. For the logical-source prototype, adding new kinds of objects would require more sophisticated rendering algorithms, but these techniques are well understood and would not pose a serious problem. The CAD-model prototype faces the complementary problem of determining whether and how objects are touching. In the current prototype, objects are determined to

be touching if they have vertices in common. For objects that partially overlap, more sophisticated but still relatively simple algorithms will be needed. For other, even more complex objects such as bolts and washers, algorithms will be needed for finding relations such as “on” and “through.”

While these issues require resolution before practical systems for co-generation can be deployed, the demonstrated feasibility of both the logical-source prototype and the CAD-model prototype suggest that co-generation approach has substantial promise for assuring consistency of text and graphic in documentation.

6. ACKNOWLEDGMENTS

This work was supported by National Science Foundation Award No. 0080940. The Unigraphics CAD software was provided through General Motors's PACE program. The members of UTEP's Interactive Systems group contributed helpful comments.

7. REFERENCES

- [1] André, E., and Rist, Thomas (1994). Referring to world objects with text and pictures, *Proceedings of the 15th Conference on Computational Linguistics*, Kyoto, Japan, 530-534.
- [2] Barth, H. (1997). *DiDoLog--A system for developing and automatically generating guaranteed consistent and complete user manuals for interactive systems*, RVS-Demo-01, <http://www.rvs.Uni-bielefeld.De/~ihbarth/DiDoLog.html>, December 8, 1997.
- [3] Bulko, W. C. (1988). Understanding text with an accompanying diagram, *Proceedings of the First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Tullahoma, TN, 2, 894-898.
- [4] Hartmann, K., and Strothotte, T. (2002). Spreading activation approach to text illustration, *Proceedings of the 2nd International Symposium on Smart Graphics*, Hawthorne, NY, 39-46.
- [5] Mackinlay, J. (1986). Automating the design of graphical presentations of relational information, *ACM Transactions on Graphics*, 5(2), 110-141.
- [6] Novick, D., and Juillet, J. (1998). Documentation integrity for safety-critical applications: The COHERE project, *Proceedings of SIGDOC 98*, Quebec, September, 1998, 51-57.
- [7] Novick, D. (2000). A why-what-how tool for development and documentation of operating procedures, *Proceedings of IPCC/SIGDOC 2000*, Cambridge, MA, September, 2000.
- [8] NTSB (1996). Docket CHI96LA011.
- [9] NTSB (2004), personal communication.
- [10] Paris, C., Vander Linden, K., Fischer, M., Hartley, A., Pemberton, L., Power, R., and Scott, D. (1995). *A support tool for writing multilingual instructions*. Information Technology Research Institute Technical Report No. ITRI-95-11.

