Notations for Software Engineering Class Structures

Pourang Irani

Faculty of Computer Science, University of Manitoba
Winnipeg, MB, R3T 2N2, Canada
irani@cs.umanitoba.ca

Abstract. This builds on previous work in which we have developed diagramming principles based on theories of structural object perception. We call these geon diagrams. We have previously shown that such diagrams are easy to remember and to analyze. To evaluate our hypothesis that geon diagrams should also be easy to understand we carried out an empirical study to evaluate the learnability of geon diagram semantics in comparison with the well-established UML convention. The results support our theory of learnability. Both "novices" and "experts" found the geon diagram syntax easier to apply in a diagram-to-textual description matching task than the equivalent UML syntax.

1 Introduction

Conceptualizing the design of a system is an important element of the entire software development process. This activity is supported by the use of sketches and diagrams to capture various aspects of the system being modeled. Many forms of diagrams have been developed for modeling software engineering problems such as those available through the Unified Modeling Language (UML) [6]. Although these diagrams are general and complete, the choice of graphical notations appear to be somewhat arbitrary so that only an expert in the field can easily learn them.

To some extent, learning and using software engineering semantics is analogous to learning semantics in a natural language. Chomsky's theory that language understanding is based on innate deep cognitive structures is now widely, if not universally, held [2]. It has also been argued that there is a similar deep structure in vision, although the purpose of this structure is not communication but perception of the environment. The perceptual theory of Marr contains visual primitives such as "blobs", "bars", and "terminations" [5]. These are interpreted according to a visual syntax thereby enabling us to understand 3D structured objects [1]. Jackendoff [4] argues that the rules of visual structure are similar to verbal language rules. He further proposes that there are cognitive "correspondence rules" between the visual meaning of a 3D structure and linguistic structure. This provides a natural link between visual structure and linguistic structures that may help explain why certain kinds of diagrams are easy to understand.

In our previous work [3], we derived a set of "naturally" occurring "geon correspondence rules" (or GCRs) to map diagram semantics. Here we describe the subset relevant to software class structures:

A. Blackwell et al. (Eds.): Diagrams 2004, LNAI 2980, pp. 441-445, 2004.

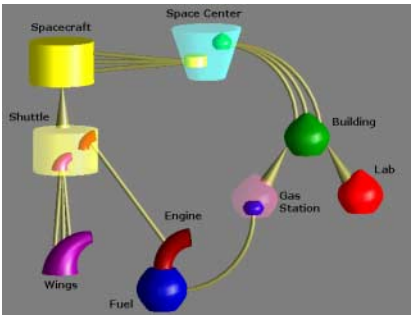

Fig. 1. Representing some related entities in a system describing a Space Center

GCR1: Inheritance - Geons with same shape can be used to denote *is-a* relationship.
GCR2: Dependency - If geon A is on-top-of geon B this suggests that geon A is supported by geon B.
GCR3: Aggregation - shows that Geon A is contained within Geon B, shown as an internal component attached to the same primitive geon on the outside.
GCR4: Multiplicity - to show multiple associations between two entities a series of attachments can best denote such a relationship.

Figure 1 illustrates an example of how these rules describe related entities of a Space Center. The Space Center has-many Buildings (containment with multiple connecting lines), and has-many Spacecraft. A Gas Station and a Lab are two different types of buildings (same shape primitive as Building). The Gas Station has Fuel (containment with connection). Shuttles are also a type of Spacecraft (same shape primitive). Shuttles have-many Wings, and has-one Engine. The Engine depends on Fuel (depicted on-top-of the Fuel entity).

While our previous results showed that the geon correspondence rules were easier to recall and more intuitive than UML rules [3], they did not say anything directly about their ability to help people match a diagram to a problem domain.

2 Experiment

We hypothesized that it should be possible to learn diagram semantics, more accurately in diagrams created with the perceptual notation presented above in comparison with an equivalent UML graphical notation. Error rate was measured for matching diagrams to informal written descriptions of various real-world problem domains.

2.1 Method

Diagrams. Five problem descriptions incorporating the semantics of generalization, dependency, one-to-many, and aggregation were constructed. The semantics in the problems were clearly presented using their common terminology. For example, to describe related entities of a neighborhood (Fig. 2.) the following text was provided:

*"The neighborhood **depends** on the city for clean water, sewage and garbage disposal. **Many** families live in this neighborhood. The neighborhood **has a** school and a pharmacy. It also has several **types** of stores: a grocery store, a convenience store, and a bakery."* Only one of the four diagrams accurately depicted the relationships in the corresponding problem description. The remaining three diagrams mis-represented several relationships.

Training. Twenty-six paid volunteers were collectively given an hour-long instruction on the various semantics and their respective notations. The training included an introduction to object oriented modeling, a description of each semantic with its UML and geon diagram notation, and sample UML and geon diagrams of complete systems with objects and their relationships. The emphasis during the training was placed on the concept underlying each semantic. It was only during this training phase that subjects had access to viewing the notations. The subjects were asked to return a week later for the experiment. At the testing stage they were tested individually.

Task. For this experiment we used a diagram-to-problem matching paradigm. After reading each problem description, the subject was asked to match one of the four diagrams created for that problem. The subject marked on the hand-out sheet the number of the matching diagram. The problem descriptions were available to the subjects while reading the diagrams, and so they could occasionally consult the description. Therefore we were not testing subject memory of a given problem text.

Subjects were restricted to two minutes for matching a diagram to a problem description. A within-subject design was used where half the subjects matched the UML diagrams first and the other half matched the geon diagrams first.

Twelve of the 26 subjects had previous exposure to UML (experts), the others had never been exposed to UML (novices).

2.2 Results

The results are obtained by averaging each subject's scores. Overall subjects matched the informal problem descriptions to Geon diagrams with an error of 14.6% vs. 36.2% with the UML diagrams. A One-Sample T-Test (or Sign Test) statistically shows that subjects performed better with the geon diagrams ($p < 0.0001$). Combining the results we can say that there were more than twice as many errors in analyzing and matching the UML diagrams than the Geon diagrams.

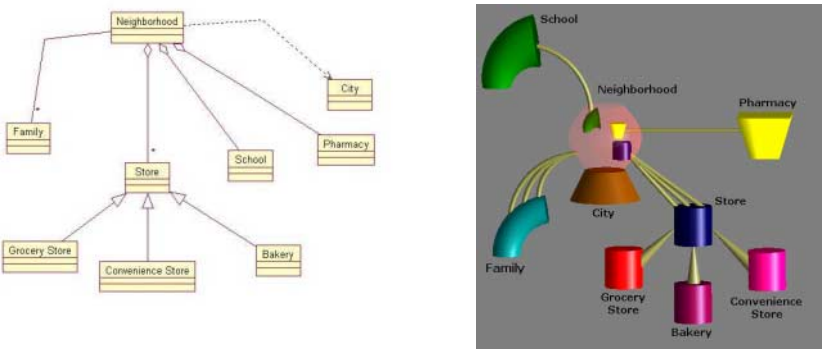

Fig. 2. Sample UML and equivalent Geon diagram for representing entities of a neighborhood

2.3 Discussion

The results obtained from the experiment described here, show that the mapping of specific software engineering semantics (inheritance, dependency, etc.) onto "geon correspondence rules" can be used as guidelines for making effective diagrams. In particular we see that the geon diagrams are well suited for learning a subset of object-oriented concepts such as those necessary for modeling software class structures. In comparing the learnability of matching problems to diagrams, we found that subjects, regardless of their experience in software modeling, were capable of learning and interpreting the perceptual syntax with fewer errors. The results were particularly significant in showing that with very little training, experts (subjects experienced only with UML diagrams and semantics through a software engineering course at the university) performed better with the geon diagrams. The use of a diagrammatic notation that requires minimal training may be particularly useful in instances where end users are involved in the development process and therefore need to quickly learn the diagrammatic notations. The experiment described in this paper focused on only one aspect of learning, i.e. matching problem descriptions to diagrams. While this constitutes a justifiable starting point for this line of research, further experimentation needs to be conducted in order to determine whether the geon notation can facilitate the process of software modeling by allowing the user to create proper abstractions of a problem.

References

[1] Biederman, I., Recognition-by-Components: A Theory of Human Image Understanding, Psychological Review, 94:2, 115-147, 1987.
[2] Chomsky, N., Aspects of the theory of syntax, Cambridge, Mass: MIT Press, 1965.
[3] Irani, P., Ware, C. and Tingley, M., Using Perceptual Syntax to Enhance Semantic Content in Diagrams, IEEE Computer Graphics & Applications, 21:5, 76-85, 2001.
[4] Jackendoff, R., On Beyond Zebra: The relation of linguistic and visual information, Cognition, 26, 89-114, 1987.

[5] Marr, D., Vision: A computational investigation into the human representation and processing of visual information, San Fransisco, CA: Freeman, 1982.
[6] Object Management Group, Unified Modeling Language (UML™), version 1.4, September 2001.