# An Object-Oriented Represenation Language to Support Multiple Perspective Explanations

Christian Rathke
Institut für Informatik
Universität Stuttgart

David F. Redmiles
Department of Computer Science
University of Colorado, Boulder

## 1 Introduction

The EXPLAINER tool [Redmiles 93] was developed to help programmers solve tasks in computer graphics by allowing them to explore previously worked-out examples from several different perspectives. Perspectives are used by EXPLAINER to provide programmers access to information about an example through code listings, execution output, text statement of a problem domain, text statement of program features, and diagrams of program components and features.

To date, EXPLAINER has relied upon a simple semantic network for representing examples. Compared to object-oriented languages, this approach has the advantage that relationships between objects can be evaluated dynamically and, in particular, perspective relationships can be implemented outside of a strict inheritance hierarchy. However, this approach fails to make the salient concepts and relationships explicit and thereby hides the underlying principles on which explanations are based.

An object-oriented knowledge representation language called FrameTalk [Rathke 93] provides a more structured means of representing examples in EXPLAINER. FrameTalk extends the object-oriented basis of the Common LISP Object System (CLOS) to provide functionality associated with frame languages such as attached procedures, defaults, slot restrictions, and constraints. A perspectives mechanisme has been made an integral part of the FrameTalk language to support the representation of examples as required by the EXPLAINER system.

## 2 Multiple Perspectives in Explainer

The purpose of the EXPLAINER tool is to make apparent to programmers the programming plan underlying an example. This programming plan may have many interrelated perspectives. The example program shown in Figure 1 solves the problem of visualizing addition modulo 100.

Several perspectives contribute to this programming plan, including MODULO-ADDITION, CYCLIC-OPERATIONS, PLOT-FEATURES, PROGRAM-FEATURES, and LISP.

The EXPLAINER interface is similar to a hypermedia tool. Initially, minimal information about the example is presented. Programmers can then decide which specific features of the example they want to explore, presumedly choosing those most relevant to their current task. Information is accessed and expanded through a command menu. Almost all items presented on the screen are mouse sensitive and may be expanded.

Highlighting may be used to show the interrelationships between perspectives. For example, the highlighting between the perspectives enables the programmer to study the mapping between PLOT-FEATURES and LISP, i.e., between the labels around the circle and the LISP code to draw those labels.

Thus, the EXPLAINER interface allows programmers to study the programming plan from different perspectives and to study the interrelationship of those perspectives.

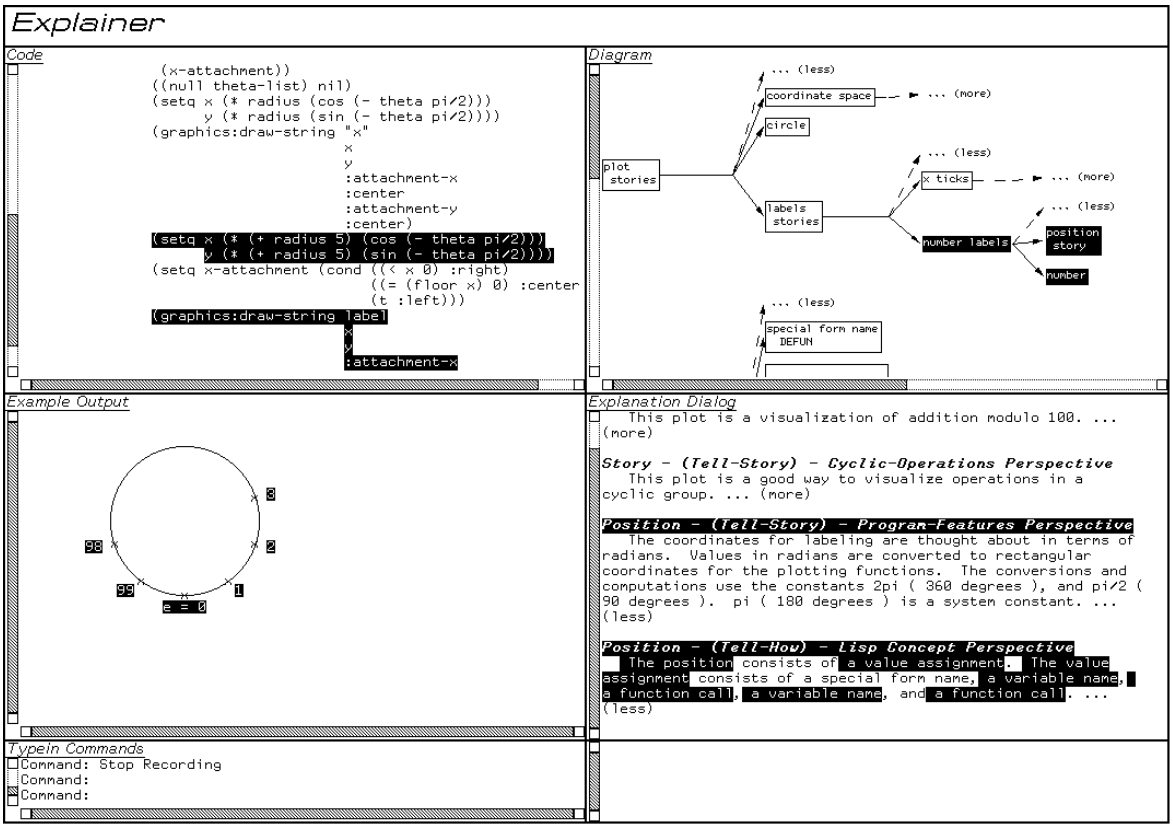



Figure 1: Exploring the "Modulo Addition" Example in EXPLAINER

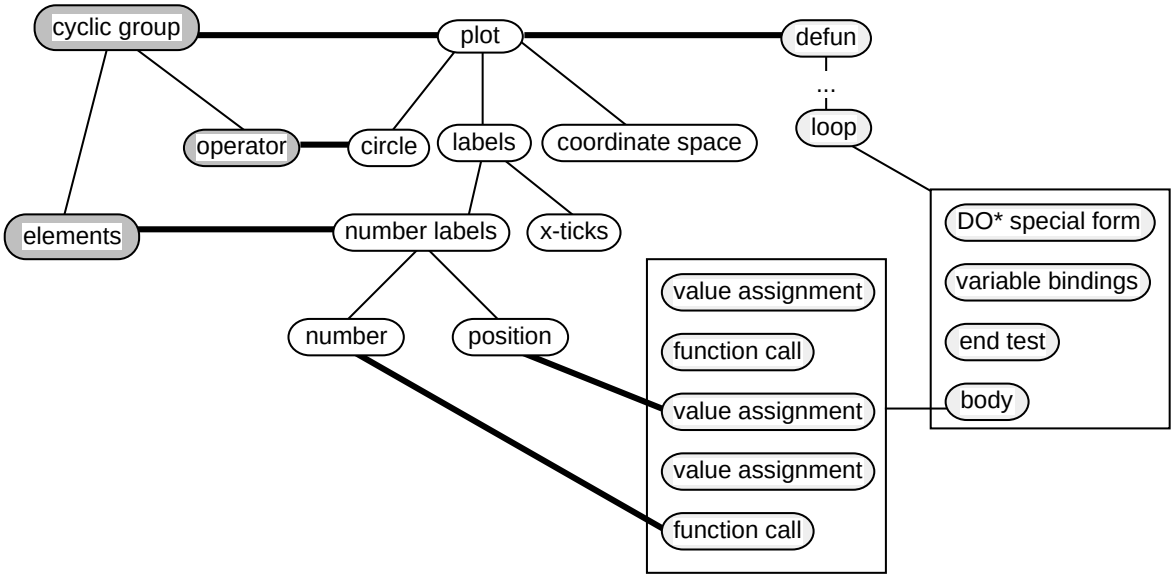
## 3 A Semantic Network Approach to Representation

Originally, Explainer's knowledge was encoded in a semantic network. The nodes of the semantic network represented concepts in an example program code. For the Modulo Addition Example, the nodes instantiated concepts including the modulo operator, labels and a circle of the plot features, and function calls, arguments and value assignments of a LISP program (see Figure 2).

Instances of nodes are connected by three kinds of links: *components, roles* and *perspectives.* The component links connect one instantiation to zero or more instantiations which comprise it *in the same perspective*. In the above example, a PLOT consists of a COORDINATE-SPACE, a CIRCLE, and LABELS.

The component link captures the "how to" or implementation knowledge in an example. The role link is the inverse of the components link and supplies one kind of "why" or goal knowledge. Instantiations exist in one specific perspective. However, they can have equivalent or analogous counterparts in other perspectives. For example the OPERATOR and ELEMENTS components of the CYCLIC-OPERATIONS perspective are equivalent to the CIRCLE and NUMBER-LABELS components of the PLOT-FEATURES perspective.

The network is searched for neighboring nodes according to a parameterized search procedure. For instance, in the situation shown in Figure 1, the programmer applied the "Highlight" command to the LABELS concept presented in the diagram (upper right pane) or in the execution output (lower left). This action initiated a search through the network for instantiations which are related by perspective links to LABELS. Several related concepts, which may be traced in Figure 2, were identified and highlighted. At




Three perspectives are shown in this figure (from left): CYCLIC-OPERATIONS (darkest ovals), PLOT-FEATURES (unshaded ovals), and LISP (shaded ovals). The thin lines represent components/roles links. The thick lines correspond to perspectives links.

Figure 2: Partial Representation of the Modulo Addition Example

a minimun, any instantiation is related through the *root*, which is equivalenced to the main function of the LISP code for that example. Thus, any instantiation can be reached from any other, though some are distant with respect to the number of intervening links and nodes.

## 4 An Object-oriented Approach to Representing Perspectives

The notion of perspectives has been made an integral part of an extended object-oriented representation formalism, called FrameTalk. FrameTalk extends the object-oriented basis of the Common LISP Object System (CLOS) to provide functionality associated with frame languages. FrameTalk is being used to replace the ad hoc semantic network representation currently used in the EXPLAINER system.

Figure 3 shows how the three perspectives for the Modulo Addition Example are specified. The Modulo Addition Example is represented by a frame consisting of three perspectives: CYCLIC-OPERATIONS, LISP and PLOT-FEATURES. Each of the perspectives consists of slots with slot descriptions, which restrict possible slot fillers. For instance, the OPERATOR slot of the CYCLIC-OPERATIONS perspective must be filled with exactly one function and all of the values of the ELEMENTS slot must be numbers. In this example, all slots mentioned *differentiate* a COMPONENTS slot, which is defined at a more general level (not shown) in each of the three perspectives. Fillers of a component slot belong to the same perspective as the component slot.

The RELATIONS part of the frame specifies the *same-object* relation for components in different perspectives. This relation ensures that slot fillers point to the same object (though potentially in different perspectives).

The definition is used to enforce perspective relations on instantiations. Asserting the fact that a given LISP program is described by the LISP perspective of the modulo addition frame generates instantiations

```
(defframe modulo-addition ()
   (:perspectives
        (cyclic-operations (operator (:one function)(:diff components))
                           (elements (:all number)(:diff components)))
        (lisp (function (:one function)
                        (:diff components))
              (arguments (:all lisp-variable) (:diff components))
              (body (:all lisp-form) (:diff components)))
        (plot-features (coordinate-space (:diff components))
                       (circle (:diff components))
                       (labels (:all graph-label) (:diff components))))
   (:relations
        (same-object (cyclic-operations operator) (plot-features circle))
        (same-object (cyclic-operations elements)
                     (plot-features labels number-labels))
```

Figure 3: FrameTalk Definition of the Modulo Addition Example

for the other perspectives and establishes the necessary relationships. The remaining details, which are specific to the example, would be filled in by the programmer. The declarative form of the definition also more readily enables reuse of the knowledge from one example frame to the next.

From a formal point of view, frames are composed of partial descriptions each of which belongs to a distinct perspective. Due to the perspectives structure of frames, an object is represented as a composition of instantiated partial descriptions. An object may be accessed by any of its instantiations, i.e., through any of its perspectives. Reasoning components of FrameTalk make use of this partitioning. For instance, the *realizer,* a component which associates partial descriptions to instantiations, does not cross a perspective's boundary, thereby significantly reducing its computational complexity.

# 5 Conclusions

The proposed perspectives mechanism extends object-oriented representation languages by providing additional structure for formulating and using knowledge:

- Perspectives allow the formulation of *partial definitions*. Composition of properties can be structurally separated from specialization of properties.
- Different views can be expressed by combining properties in different ways. Objects may be described from different viewpoints by multiple perspectives.
- The perspectives structure can be used for comparing objects and answering questions for explanation purposes.

Furthermore, the multiple perspectives representation serves the dual role of providing a framework for organizing knowledge according to the needs of the mental models people rely upon during problem solving. This supports better provision and structuring of the knowledge represented for explanation.

## References

[Rathke 93] C. Rathke. Object-Oriented Programming and Frame-Based Knowledge Representation. In *Proceedings of the 5th IEEE International Conference on Tools with Artificial Intelligence*, pp. 95–98, Boston, MA, November 1993. IEEE.

[Redmiles 93] D. Redmiles. Reducing the Variability of Programmers' Performance Through Explained Examples. In *Human Factors in Computing Systems, INTERCHI'93 Conference Proceedings*, pp. 67–73. ACM, 1993.