Metaphor-Based Animation of OO Programs

Jorma Sajaniemi
Department of Computer Science
University of Joensuu, Finland

Pauli Byckling
Department of Computer Science
University of Joensuu, Finland

Petri Gerdt
Department of Computer Science
University of Joensuu, Finland

Abstract

Program visualization and animation have traditionally been done at the level of programming language implementation, or by using some formal graphical notation, e.g., UML; both of which are unfamiliar to novices. We present a metaphor approach to animating object-oriented programming with the goal to provide novices with visually rich visualizations that can develop and consolidate a valid mental model of object-orientation. For this purpose, we introduce new metaphors for such concepts as object, object instantiation, method invocation, parameter passing, and object reference. These metaphors are used in a system that provides the animation of a few, carefully selected OO programs for educational use.

CR Categories: K.3.2 [Computers and Education]: Computer and Information Science Education—computer science education; D.m [Software]: Miscellaneous

Keywords: program animation, metaphor, object-oriented programming

1 Introduction

Program visualization and animation have traditionally been done at the level of the programming language and its implementation in a computer. For example, variables have been visualized as boxes (representing memory locations), and nested function calls as a stack of frames containing parameters and local variables (representing the call stack implementation in many computer architectures). In object-oriented (OO) context, animation has also been based on UML diagrams that reveal connections between objects and classes and thus represent another level, i.e., relationships between components of an individual program. We know of only one program animation system, PlanAni [Sajaniemi and Kuittinen 2004], that builds its visualization on general programming knowledge (roles of variables) and uses metaphors to make this knowledge easier to assimilate by learners.

Novices have problems in learning the very basic OO concepts which result in misconceptions leading to either erroneous or suboptimal programming skill (see, e.g., [Eckerdal and Thuné 2005; Fleury 2000; Holland et al. 1997]). Program visualization is supposed to enhance learning and prevent misconceptions but the visualizations should be at the same level as the concepts to be learned. Thus visualizations that build upon programming language implementation or abstract UML notation may easily fail in helping novices to learn programming concepts.

Metaphor involves the presentation of a new idea in terms of a more familiar one [Carroll and Mack 1999]. In contrast to analogy, metaphor is not an exact counterpart but differs from the idea usually both in form and in content. The similarities and differences between the two ideas stimulate thought and can facilitate active learning [Carroll and Mack 1999; Hsu 2006]. This paper applies a metaphor approach to object-oriented programming. Our ultimate goal is to provide novices with metaphors that will help them in learning basic OO constructs. For this purpose, we present new metaphors for such concepts as object, object instantiation, method invocation, parameter passing, and object reference. The metaphors are designed to grasp the basic ideas of object-oriented programming; they do not rely on implementation issues or diagramming techniques designed for expert use. These metaphors are used in Flash animations that can be used in programming education.

2 Visualization of OO Concepts

An object encapsulates the existence, state and behavior of an entity. Its visualization should reflect these three aspects. The existence is limited by the instantiation of an object and its destruction in garbage collection. The state is manifested in the member variables, and the behavior is a result of method invocations that include the creation and destruction of local variables. The behavior of individual member and local variables can be described by roles that already have metaphors, e.g., a dog for the role follower, a box for gatherer etc [Sajaniemi and Kuittinen 2004]. For an object, we therefore use the metaphor of a watch panel with class-dependent fixed “monitors” for its member variables depicted in the form of role metaphors (see Figure 1).

The instantiation of an object is animated by making a copy of a class-specific blueprint template found in a blueprint book that lays normally outside the screen and emerges only when needed for object instantiations and method invocations. Each blueprint occupies its own page in the book, and class variables are located on the same page. Whereas the background color for blueprints is blue, the background color of the class variable area is white. This area becomes visible whenever an object of that class is active. Class variables are depicted with the same role metaphors as member variables.

For method invocations we use the metaphor of temporary workshops containing all parameters and local variables, and a workshop for the result of the invocation. Templates for workshops are found at the class-specific blueprint. If another method of the same object is invoked or a method is invoked recursively, a new workshop is created. Thus the number of co-existing workshops depends on the number of unfinished method invocations. To stress the fact that method invocations are associated with the object’s member variables, the workshops are attached to the watch panel depicting the object. Finally, a static method is visualized as a permanent workshop with a concrete foundation and a strong roof.
The traditional verbal metaphor for method call is “message passing”. We visualize this metaphor with an *envelope* that delivers actual parameters to the watch panel associated with the called object. The empty envelope stays on the workbench and is filled with the return value when the method invocation ends. Then the envelope flies back to the calling workshop along a path that was created during the method call.

For object references we use a *pennant* metaphor: an object reference is visualized as two pennants with the same unique color; one pennant is attached to the object reference variable and the other to the referenced object. A null reference is visualized with two pennants lying on the ground. If two variables refer to the same object, the object has two pennants. As a consequence, an object with no pennants cannot be referenced and is subject to garbage collection, which is animated by a *garbage vehicle* that moves around and stops next to each object, i.e., watch panel, with no pennants. The finalizer is then invoked and the watch panel is finally squashed into the vehicle.

### 3 Conclusion

The visualizations and animations presented above are not meant to be used in, e.g., debugging or comprehending large programs, but in elementary programming education. We suggest that roles of variables [Sajaniemi and Kuittinen 2004] are first introduced with their role metaphors using the PlanAni program animator. When the role metaphors are familiar, the OO metaphors can be introduced by animating a few, carefully selected OO programs. With appropriate student engagement this will give students a correct mental model of the relationships between OO concepts—a mental model that builds up on a spatial representation for program execution. The visually rich visualizations of the metaphors are expected to consolidate this mental model so that even though the visualizations do not scale up on computer screen, the metaphors do scale up in novices’ mental representations.

### 4 Acknowledgments

We would like to thank Ms. Taina Tikansalo for programming the animations. This work was supported by the Academy of Finland under grant number 206574.

### References


HSU, Y.-C. 2006. The effects of metaphors on novice and expert learners’ performance and mental-model development. Interacting with Computers 18, 4, 770–792.