A Visualization of the Frame Representation Language

James Foster
GemStone Systems, Inc.
1260 N.W. Waterhouse Ave., #200
Beaverton, OR 97006 USA
1-503-533-3114
james@foster.net

Paul Juell, Ph.D.
Department of Computer Science and Operations Research
North Dakota State University, IACC #258
Fargo, ND 58105 USA
1-701-231-8196
paul.juell@ndsu.edu

Abstract

We wish to make frame-based computing easer to deal with by providing a visualization interface. This interface will both show the contents of the knowledge base and show the code execution of a frame-based program. We have implemented FRL, a frame-based language, and extended it with the visualization interface.

Categories and Subject Descriptors I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods--Frames and scripts (FRL); D.2.5 [**Software Engineering**]: Testing and Debugging--symbolic execution; D.2.6 [**Software Engineering**]: Programming Environments--graphical environments; H.5.2 [**Information Interface and Presentation**]: User Interfaces--Graphical user interfaces (GUI); I.2.5 [**Artificial Intelligence**]: Programming Languages and Software (FRL); K.3.2 [**Computers and Education**]: Computer and Information Science Education--Computer science education.

General Terms Human Factors, Languages.

Keywords Frame Representation Language; FRL; Visualization.

1. Introduction

Object-oriented programming was significantly influenced by knowledge representation research in the early artificial intelligence community. While some of these early efforts, particularly Lisp, have readily recognized descendants in today's technologies, other influences are less visible. One of the precursors to today's "object" was the "frame," a knowledge representation approach introduced in 1974, and the Frame Representation Language (FRL), a programming language used to explore the use of frames.

A few factors tend to limit the accessibility of FRL to today's students of objects. First, there does not appear to be a generally available running implementation of FRL's original syntax and core features. This means that the current study of FRL has been based on reading papers, not running code. Second, since the programming systems of the 1970s were primarily text-based systems, there was no graphical user interface available to explore the execution of FRL commands and their effects on the internal database of knowledge representation.

This demonstration will show a visualization of the FRL programming language that provides a bridge between the text-based representation that is a legacy of its 1970s technology and the more interactive and intuitive graphical user interface that is commonly used three decades later. We will show several views of the frames database and a view of the code being executed. It is a goal of this visualization to make the influential ideas of frames and FRL more accessible to today's students of object-oriented programming.

2. Review of Applicable Technologies

2.1 Knowledge Representation

When building a machine to solve a computational problem (such as ballistic firing tables), the creators found ways to represent and manipulate *numbers* so as to automate the steps a human might take to solve the problem. As researchers approached non-computational problems, they needed to represent and manipulate *ideas* and *concepts*.

2.2 Frames

Marvin Minsky's 1974 paper, *A Framework for Representing Knowledge* [1], introduced an influential model for representing knowledge. Instead of emphasizing the fine-grained details, he focused on how information fits into the larger context around it. Minsky described a system of frames, where each frame is a data-structure representing knowledge. A top-level frame contains a stereotyped situation and a lower-level frame represents a specific instance of that situation.

What was needed was a computer implementation of Minsky's frame system that could be used to apply these ideas. The most popular programming language for AI research at the time was (and arguably still is), Lisp. The ability to extend Lisp by adding functions and the interchangeability of code and data makes Lisp an attractive research environment for creating specialized programming languages. The Frame Representation Language (FRL) is one such language.

2.3 Frame Representation Language (FRL)

In 1977 researchers in MIT's AI Laboratory reported on FRL, an experimental language implemented in Lisp that was used to explore ways of using frames to represent knowledge. *The FRL Primer* [2] described this programming language by using a simple example that forms the basis of the visualization presented in this paper. A couple months later Roberts and Goldstein provided a more definitive description of the language in *The FRL Manual* [3].

FRL uses Lisp's list data structures to represent frames. The first element of the list is the name of the frame (a symbol) and subsequent elements of the list (if any) are "slots." Each slot is represented by a list in which the first element is the slot name (again, a symbol), and the subsequent elements of the list are "facets" of the slot. The frame definition continues in a recursive manner with "datum," "label," and "message" completing the structure. All frames are contained in a Lisp global, *FRAMES*.

FRL extends Lisp by adding a number of functions to manipulate frames. Functions are provided to create a new frame, to add information to a frame, to query a frame for some information, and to remove information from a frame. In addition, support is provided for Minsky's proposal that frames be related to one another in a hierarchy with default values provided by a higher level if not explicitly provided by a lower level. Also, following the Lisp approach that data and code are interchangeable, a frame can have code attached to it so that if a value is added or removed from a slot, appropriate functions are executed.

3. Problem and Solution from User's Point of View

3.1 Problem

While the modern graphical user interface (GUI) can be traced to research done in the 1970s, it was not in general use until years later and the typical Lisp environment (in which FRL was embedded) had a text-based user interface. Much like the now-ubiquitous shell (whether DOS or UNIX), a Lisp environment prompts the user for entry of an expression, parses and executes the expression, and then prints the result to the terminal. An expression can have side-effects that change the state of the internal system, but that change would not be visible unless it happened to be displayed then or later. The typical output of an expression involves a lot of symbols and parenthesis, which can be rather confusing to for a user trying to determine what is happening.

An additional difficulty with learning about the FRL system is that while it was influential, its legacy is mostly in what grew out of its ideas. As a language or system in itself, it did not continue to be maintained and refined (as compared to Lisp, for example, which has an enthusiastic academic and commercial following almost fifty years after it was introduced). Most students of AI are exposed to FRL through scanned images of the original papers.

There exists at least one implementation of FRL that can be installed in a modern Lisp environment. Unfortunately, because of changes in Lisp since the introduction of FRL in 1977, this implementation does not support the FRL implementation described in the original papers. For example, the FRL convention is to end a *label* with a colon character. According to the current Lisp standard, the colon character is reserved to describe packages. This means that the original FRL examples cannot be exactly demonstrated in the available implementations. In addition, because its host Lisp environment is still following the command-line model of user interaction, the user interface is limited to seeing a text representation of the results of each command, so the user cannot explore the current data without executing other FRL commands.

What is missing is an interactive implementation of FRL that provides the visual feedback typically found in today's tools. A tool to fill this gap will be demonstrated.

3.2 Solution

Since FRL is essentially a programming language, we take as our visualization model the modern integrated development environment (IDE). The typical IDE has separate windows or panes for the code and for the data so that the user can examine the state of the system from a variety of perspectives. In the code pane, the expression to be evaluated next is highlighted. In the data pane, the current state of the local variables is displayed.

Like the modern development tools, our FRL tool separates the code from the data and allows exploration of the date without executing new code. In order to examine alternatives for viewing the frames database, we have several different views of the data, each shown in a separate tab.

4. Implementation

The demonstration presented here is a working FRL system with a visualization that shows, in a step-by-step manner, the execution of the FRL code and the impact of each command on the internal database. The environment chosen for this implementation is based on another language from the 1970s, Smalltalk. [4]

Source code for the visualization, along with an executable for Microsoft Windows platforms, is available for download from the web at http://james.foster.net/frl.

References

[1] Minsky, M. 1974 *A Framework for Representing Knowledge*. Technical Report. UMI Order Number: AIM-306., Massachusetts Institute of Technology.

[2] Roberts, R. B. and Goldstein, I. P. 1977 *The FRL Primer*. Technical Report. UMI Order Number: AIM-408., Massachusetts Institute of Technology.

[3] Roberts, R. B. and Goldstein, I. P. 1977 *The FRL Manual*. Technical Report. UMI Order Number: AIM-409., Massachusetts Institute of Technology.

[4] Kay, A. C. 1993. The early history of Smalltalk. In *the Second ACM SIGPLAN Conference on History of Programming Languages* (Cambridge, Massachusetts, United States, April 20 - 23, 1993). HOPL-II. ACM Press, New York, NY, 69-95. DOI= http://doi.acm.org/10.1145/154766.155364