Frames or Objects, or Both?

Ora Lassila

Laboratory of Information Processing Science
Helsinki University of Technology
Otakaari 1A, SF-02150 ESPOO, FINLAND

Workshop Notes from the Eighth National Conference on Artificial Intelligence (AAAI-90): Object-Oriented Programming in AI. Boston (Massachusetts, U.S.A.), July 1990.

Abstract:

This paper discusses the differences and similarities of object-oriented programming systems and frame systems, and advocates the amalgamation of these two kinds of systems. A frame system called BEEF is introduced. It provides, in addition to frame system characteristics, the basic functionality and ease of use of an object-oriented extension of Common Lisp, making it well suited for knowledge-based systems development, both as a programming and a knowledge representation tool.

TKO-B67
ISBN 951-22-0870-9
ISSN 0785-6601

TKK OFFSET 1991

1. Introduction

Although it is very difficult to completely and precisely define the object-oriented programming paradigm, a practical characterization can be given: we can say that central to the paradigm is the *definition of types* and the *creation of instances* of these types, and that types share descriptions of structure and behaviour via *inheritance*. This is not universally adequate (several good different definitions and characterizations exist [e.g. Pascoe 86, Meyer 88]), but will do for our purposes. In this paper an object-oriented programming system (OOPS) is understood to be a system with the above-mentioned characteristics. Examples of relevant OOPSs are the Lisp Machine Flavors system [Moon 86] and the Common Lisp Object System (CLOS) [Bobrow et al 88]. OOPSs are often extensions to conventional languages.

The concept of a frame system [Minsky 75] has gained ground as a basic mechanism for representing knowledge. The fundamental idea is very simple: *frames* are the system's basic objects, they represent real-world concepts and phenomena; frames can be given named attributes, *slots*, and slots can be assigned *values*. Inheritance allows slot values to be used as *defaults*. Examples of frame systems are the Carnegie Representation Language (CRL) [Carnegie 86] and the representation language of KEE [Fikes & Kehler 85, IntelliCorp 85]. Frame systems are often parts of hybrid development tools.

Frame systems satisfy the given characterization of the object-oriented paradigm: they typically allow the definition of types and the creation of instances; they also allow types to inherit information from other types. How, then, do OOPSs and frame systems differ. Could a frame system be used as an OOPS, or vice versa, and what would be the benefits of this?

2. Object-Oriented Programming vs. Frame-Based Programming

In OOPSs types usually form a taxonomy, a so-called is-a hierarchy. Inheritance is understood as the propagation of method and object structure information in this hierarchy. Often the propagation takes place at definition time, and when a program which was implemented with an OOPS is running, the inheritance hierarchy is more or less fixed.

In frame systems inheritance is usually used to propagate default slot values in the type hierarchy. Typically the inheritance is dynamic in nature, i.e. takes place at run time. This allows the default values to be changed during program execution. Frame systems may also allow the use of inheritance paths other than the usual is-a hierarchy.

The differences in inheritance are closely related to differences in program access to the type hierarchy. In an OOPS, only instances are objects of the system (some OOPSs, though, have been implemented with themselves, and offer facilities for *metaprogramming* [Goldberg & Robson 83, Bobrow et al 88]. In some frame systems types do not fundamentally differ from instances, hence program access to the type hierarchy is possible without any meta level.

Another useful feature of some frame systems is the ability to create and maintain multiple parallel and nested worlds, i.e., versions of the object universe. When a world is deleted, all changes made in it since it's creation are also destroyed. As an example, a world mechanism can be used when programming complex backtracking search algorithms (say, when the state of the whole frame system can be regarded as a search state).

OOPSs are usually designed for programming; the object-oriented features are useful in modularizing and structuring programs in an efficient manner. Frame systems, however, are usually designed for representing knowledge, with the programming aspect not necessarily considered very important. OOPSs in general are not adequate for knowledge-based systems programming. Frame systems may offer the extra features needed: dynamic default values, multiple worlds, knowledge bases, advanced inheritance etc. To tap this potential in the context of object-oriented programming, a frame language should also be made suitable for conventional programming.

3. BEEF

BEEF (BOSS's Extremely Elegant Frame System) [Lassila 90a] is a frame system implemented with Common Lisp [Steele 84]. It was designed with the intention of bringing object-oriented and frame-based programming together. Initially, BEEF was used in porting software from a Lisp Machine to a microcomputer: BOSS (Bunch of OPIS-like Scheduling Systems) [Hynynen & Lassila 89] is a distributed knowledge-based production scheduler, based on the OPIS scheduler [Smith et al 86]. OPIS and BOSS were implemented using Common Lisp and CRL (part of Knowledge Craft, a hybrid development tool). Knowledge Craft was not available in the target environment, hence another knowledge representation system was needed.

Object-oriented knowledge representation is often available in heavy knowledge-engineering tools. KEE and Knowledge Craft offer the programmer a rich selection of features, but their completeness can sometimes be viewed as their weakness: it takes a lot of memory and CPU cycles to run these tools, and this often degrades their expressiveness and power.

BEEF's design was influenced by constrained computational resources of the target environment. BEEF was implemented to facilitate the adaptation of a large AI program into a microcomputer, thus it was required that itself not be too large, nor it's dynamic memory consumption be too high. BEEF's possible use as a portable OOPS also guided the design and implementation.

The experience with CRL showed that a frame system offering a rudimentary provision for frame methods forces the programmer to a tedious programming style. CRL only allows separately defined functions to be assigned as slot values, and a clumsy way of calling these. Furthermore, the programmer has to obey a certain protocol as far as the formal parameters are concerned.

BEEF provides a simple efficient way of writing frame methods, facilitating object-oriented programming in practice, and not just theoretically. A special method definition form (á la defun) invisibly enforces the parameter protocol and also makes it easy to access the slots of the method's receiver. Inside the method's body, the receiver frame's slots can be accessed as local variables. The method definition mechanism resembles that of CLOS.

BEEF is based on a small set of concepts, the basic ones being *frames*, *slots* and *values*. Modeling and programming is quite possible with Common Lisp and these concepts alone. Since BEEF is an extension of Common Lisp, one basic goal was also to make it "culturally compatible" with CL.

From the modeling point of view, *frames* are used as equivalents of real-world objects and concepts. They can be named or anonymous. They can be given named attributes, called *slots*. To complete the framework, we also have *values*. A slot can have any number of Common Lisp data objects as values.

3.1. Relations and Inheritance

Slots provide a mechanism to attach values to frames. This notion is extended by introducing *relations*, defined recursively as follows:

1. A slot is a relation (a data object used as a value of slot x is said to be in relation x with the frame that has the slot x).

2. An expression composed of relations and one of so-called path grammar operators is a relation. The path grammar operators are **or**, **sequence**, **sequence+** and **repeat**.

As can be seen from the definition, relations are regular expressions of slots and path grammar operators. The behaviour of relations is best understood if we view the frame universe as a graph consisting of vertices (frames) and arcs (connections through slots). Path grammar expressions are used as match patterns in the graph when values are searched for. In other words, values are only searched from those parts of the frame graph that is matched by the regular expression.

Functions that access the values of slots can also be used to access data through relations. The function **value** finds the first value reachable through a relation, and the function **all-values** computes the closure of the relation provided. Cyclic paths are detected and traversed only once.

Relations can be named. This requires one to create a frame which describes the relation and its behaviour. Because slots are primitive relations, their behaviour can also be defined by creating a relation description. One can, for instance, give the slot an inverse relation which is automatically updated whenever the slot is updated.

In object-oriented programming and modeling, it is useful to organize the concepts into taxonomies called *class hierarchies*. Concepts are defined in terms of other concepts: this is done by specializing, refining and adding details. These taxonomies are often called **is-a** -hierarchies because the basic relation between the frames (that is, classes) is called **is-a**. BEEF supports multiple inheritance.

The frames of a class hierarchy are often thought of being data types, descriptions of data. The real data objects, so-called instances, are the leaves of the hierarchy. In BEEF there is no fundamental difference between the frames used as classes and the frames used as instances. To support the idea of instances, a special relation called instance is used to link instances to classes.

The inheritance of slot values is a special case of path grammar computation. It can be altered on a slot-to-slot basis: any path grammar expression can be substituted in place of the default **is-a** relation. This allows dynamic propagation of data in frame graphs.

3.2. Worlds

A *world* is a snapshot of the frame universe. Initially, all actions take place in a so-called root world. A new world can be created as a child of an old world. It is initially similar to its parent, but changes made in it cannot be seen in the parent. When a child is deleted, all changes made in it are undone. Changes can also be copied upward to the parent.

3.3. Methods

Programs can be organized by making frames communicate through *message passing*. A frame receiving a message needs to be able to handle it. This is done by *message handlers* attached to the frames. Handlers are Lisp functions assigned to slots as values, the names of the slots being the same as the messages. Messages are sent by calling a function which has the same name as the message, and passing the receiver as its first argument. This function chooses the actual function according to the type of the receiver.

3.4. Experiences with BEEF

BEEF was designed and implemented as a basic programming tool. This, and the requirement of portability led to the situation that BEEF itself doesn't have any interactive tools to aid its use. Based on the experience with CRL and BEEF itself, object-oriented programming and modeling is possible with the definitions being manipulated with a text-editor. This sounds worse than it actually is, because BEEF frame definitions are clear and simple and not too verbose.

BEEF was first used when the BOSS software was ported from a TI Explorer into a Macintosh II (running the Macintosh Allegro Common Lisp [Apple 89]). A CRL-compatibility shell written for BEEF made the task very easy. It was also found out that because of the small size and low memory consumption of the BEEF tool, the BOSS software could be used in the new much more constrained environment. Measurements have shown the dynamic memory consumption of BEEF to be smaller than that that of CRL (the consumption depends on the programming style used and the task being implemented, the average being somewhere between 10% and 50%). Accurate speed comparisons have not been made yet, but observations of the BOSS software show the Explorer and Macintosh versions to execute at about the same speed.

The first large software project based entirely on BEEF is the prototype of a distributed knowledge-based scheduler for a steel-milling plant [Lassila et al 90]. In BOSS, BEEF was only used for modeling, but the new software has been written as a collection of frame classes and methods. The project has inspired and influenced further development of BEEF in the form of a distributed version of the frame system which allows frames to reside in different machines in a local-area network.

BEEF has also been used as a prototyping tool in smaller projects, such as in the implementation of a frame-based substitute for the Lisp Machine **defsystem**-tool, and in the design of a knowledge-based system for group technology [Lassila 90b].

4. Conclusions

OOPSs and frame systems have many similar characteristics, but are usually designed for different purposes. OOPSs are used programming, while frame systems are mainly used for representing knowledge. The development of knowledge-based systems involves both kinds of tasks, and it would be beneficial if both could be carried out with the same tool.

The BEEF frame system was designed to amalgamate ideas of object-oriented programming and frame-based knowledge representation. The first software projects carried out with BEEF as their basic programming tool have provided support to this line of thinking. BEEF has also proved efficient in environments with limited resources.

5. Literature

[Apple 89] Apple Computer, *Macintosh Allegro Common Lisp™ version 1.3 Reference*. APDA™ #M0067LL/C, Apple Computer, Cupertino, California (U.S.A.), 1989.

[Bobrow et al 88] Daniel G. Bobrow, Linda G. DeMichiel, Richard P. Gabriel, Sonya E. Keene, Gregor Kiczales and David A. Moon, *Common Lisp Object System Specification*, X3J13 Document 88-002R, SIGPLAN Notices, volume 23, Association for Computing Machinery, September 1988.

[Carnegie 86] Carnegie Group, *Knowledge Craft User's Manual*. Carnegie Group, Inc., Pittsburgh (Pennsylvania, U.S.A.), 1986.

[Fikes & Kehler 85] Richard Fikes and Tom Kehler, *The Role of Frame-Based Representation in Reasoning*. Communications of the ACM, 28:9, Association for Computing Machinery, September 1985.

[Goldberg & Robson 83] Adele Goldberg and Dave Robson, *Smalltalk-80: The Language and its Implementation*. Addison-Wesley, Reading, Massachusetts (U.S.A.), 1983.

[Hynynen & Lassila 89] Juha Hynynen and Ora Lassila, *On the Use of Object-Oriented Paradigm in a Distributed Problem Solver*. In: Hannu Jaakkola and Seppo Linnainmaa (eds), Proceedings of the Second Scandinavian Conference on Artificial Intelligence (SCAI'89), pp. 657-669, Tampere (Finland), June 1989.

[IntelliCorp 85] IntelliCorp, *KEE™ Software Development System User's Manual*. IntelliCorp, Mountain View, California (U.S.A.), July 1985.

[Lassila 90b] Ora Lassila, *Knowledge-Based Algorithm for Group Analysis*. Accepted to the APMS'90 -conference, Otaniemi (Finland), August 1990.

[Lassila 90a] Ora Lassila, *The BEEF Reference Manual - A Programmer's Guide to the BEEF Frame System*. Report TKO-C43, Laboratory of Information Processing Science, Helsinki University of Technology, Otaniemi (Finland), February 1990.

[Lassila et al 90] Ora Lassila, Markku Syrjänen and Seppo Törmä, *Coordinating Mutually Dependent Decisions in a Distributed Scheduler*. Accepted to the APMS'90 -conference, Otaniemi (Finland), August 1990.

[Meyer 88] Bertrand Meyer, *Object-oriented Software Construction*. Prentice Hall International Series in Computer Science, Great Britain, 1988.

[Minsky 75] Marvin Minsky, *A Framework for Representing Knowledge*. In: Patrick Henry Winston (ed.), The Psychology of Computer Vision. McGraw-Hill, New York (U.S.A.), 1975.

[Moon 86] David A. Moon, *Object-Oriented Programming with Flavors*. Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'86), SIGPLAN Notices, vol 21 number 11, Association for Computing Machinery, November 1986.

[Pascoe 86] Geoffrey A. Pascoe, *Elements of Object-Oriented Programming*. BYTE, August 1986, pp.139-144.

[Smith et al 86] Stephen F. Smith, Mark S. Fox ja Peng Si Ow, *Constructing and Maintaining Detailed Production Plans: Investigations into the Development of Knowledge-Based Factory Scheduling*, The AI Magazine, Vol. 7, No. 4, ss. 45-61, The American Association for Artificial Intelligence (AAAI), Fall, 1986.

[Steele 84] Guy L. Steele Jr., *Common LISP: The Language* (first edition). Digital Press, Digital Equipment Corporation, U.S.A., 1984.