References

Abelson, H., and G. Sussman, 1985, *Structure and Interpretation of Computer Programs*, The MIT Press, Cambridge, Massachusetts.

Chandrasekaran, B., 1986, Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design, *IEEE Expert*, **1**(3).

Chang, R., 1990, A Cliche-Based Reflective Architecture, *Proceedings of the ECOOP/OOPSLA '90 Workshop on Reflection and Metalevel Architectures in Object-Oriented Programming*, Ottawa, Canada.

Chang, R., and G. Novak, 1991a, Computational Reflection in Executions of Knowledge Level Models, *Proceedings of the 1991 ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) Workshop on Reflection and Metalevel Architectures in Object-Oriented Programming*, Phoenix, Arizona.

Chang, R., and G. Novak, 1991b, Components of Expertise for Knowledge Level Modeling, *Proceedings of the 1991 IEEE Third Conference on Tools for AI*, San Jose, California.

Chang, R., 1993, Cliche-Based Modeling for Expert Problem-Solving Systems, Ph.D. Thesis, The University of Texas at Austin.

Maes, P., 1987, Computational Reflection, Ph.D. Thesis, Laboratory for Artificial Intelligence, Vrije Universiteit Brussels.

Novak, G., 1983, "GLISP User's Manual," TR-83-25, Artificial Intelligence Laboratory, The University of Texas at Austin.

Steele, G., and G. Sussman, 1978, "The Art of the Interpreter or, the Modularity Complex (Part Zero, One and Two)," AI-Memo-453, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, 1991, *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, New Jersey.

Vanwelkenhuysen, J., and P. Rademakers, 1990, Mapping a Knowledge Level Analysis onto a Computational Framework, *Proceedings of the 1990 European Conference on Artificial Intelligence (ECAI-90)*, Stockholm, Sweden.

4. Analyst's Workbench

The concept of cliche-based modeling for problem solving has been demonstrated by a test-bed system, called Analyst's Workbench [Chang, 93], which is written in GLISP [Novak, 83]. This system is designed to help analysts to interactively select, construct and edit analysis models by a graphical user interface, and to compile these models into programs for answering a family of analysis questions involving geometric, mathematical, and database reasoning. An overall picture of this cliche-based modeling approach to problem solving is shown in Figure 1.

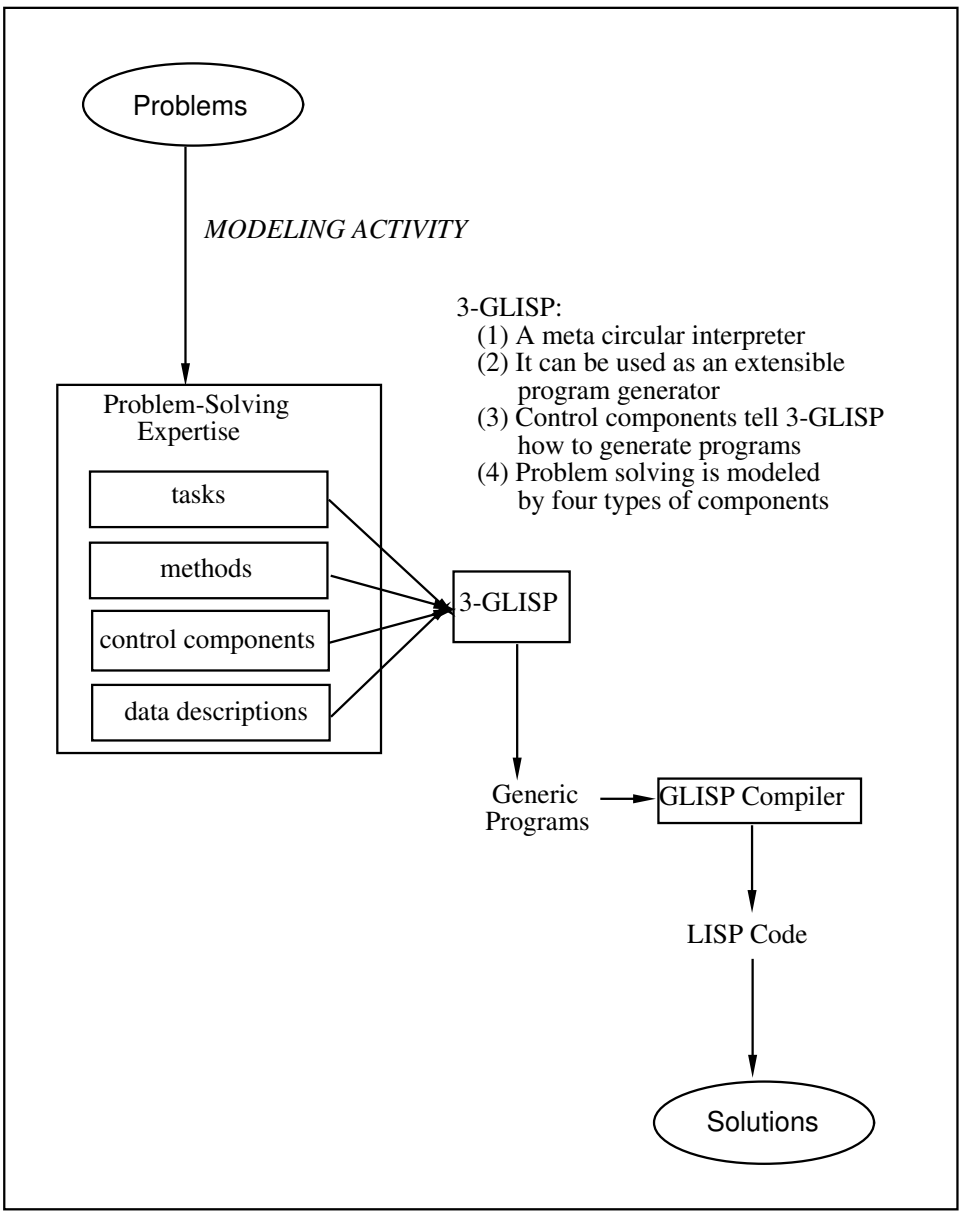

Figure 1 Cliche-based modeling for problem solving.

the other dynamically. Meta and object level systems can be causally connected by reflectors in the following sense. A reflector is accessible or modifiable at object level. Any change of a reflector affects behavior of the meta-level computation and, thus, the effects are *reflected* in the object-level computation (that is how it got the name). For instance, variable "env", representing a list of bindings in the evaluation environment of a Lisp interpreter, would be a reflector if Lisp programs could access and modify the variable at run-time. Since the Lisp interpreter is based on the variable "env" for program evaluation, any modification of it will be reflected in the Lisp program evaluation.

When a meta system reasons about or performs computation on the reflectors, the meta system is performing *reflective computation* on itself. Reflectors can affect behavior of the interpreter without modifying its code. This can be done either by dynamically modifying reflectors specified in object-level programs or by defining new reflectors for extending the system with new methods of interpretation.

A meta circular interpreter is an interpreter for a language, which is written in that language [Steele, 78] [Abelson, 85]. Due to the use of the same language in meta and object levels, it offers a natural way to provide reflective computation. A meta circular interpreter, called 3-GLISP, has been implemented in this reflective architecture to reason about reflectors. It is a cliche-based meta circular interpreter since it is designed for cliche interpretation in a cliche-based system [Chang, 90].

The capability of extensible interpretation offered by this reflective architecture allows flexibility in plugging together generic components for generating application-specific programs. This is realized by designing the *control components* as *reflectors* in our system. Because the control components provide control knowledge for building structures of problem-solving programs from different components of expertise, designing the control components as reflectors can avoid the fixed control strategy in a problem-solving system.

In this reflective architecture, the execution knowledge of a cliche is self-described and distributed over certain cliche property slots. Control components, specified in these cliche properties, are used to describe the execution knowledge of a cliche. Each control component represents a modular component of the cliche execution knowledge. This modularly represented execution knowledge can be easily shared and reused [Chang, 91a].

Furthermore, in this reflective architecture, a customizable cliche execution process can be provided. Users are able to create their own cliche execution process in order to meet their special needs in different problem domains. Thus, this object-oriented reflective architecture can provide an open-ended framework for exploratory modeling in problem-solving systems.

2. A Cliche-Based Modeling Approach to Problem Solving

A cliche-based modeling methodology is proposed in this research to develop methods for automating the construction of computer programs to answer analysis questions. In this cliche-based modeling paradigm, a problem-solving process is specified by selecting and connecting generic models of problem-solving components, which are represented as *cliches*. In our system, a cliche is a high-level object-oriented construct for representing executable problem-solving components. Through knowledge level analysis of how problem-solving components can be abstracted and explicitly represented for reuse [Chandrasekaran, 86] [Vanwelkenhuysen, 90], four kinds of cliche components have been identified in this research [Chang, 91b]. Task components are used to model target problems, which may have alternative solution structures composed of method components. Method components provide either direct solutions or decompositions into subtasks. Control components specify connections and ordering of cliche components for interpretation. Data description components describe data in application domains.

An object-oriented reflective architecture has been designed in our system [Chang, 93] to provide an execution mechanism for plugging together components of expertise to generate problem-solving programs. The generated programs are constructed by executing a series of related components. Initially, a task component is executed for solving a target problem. The task execution invokes the execution of its control components, which will select a candidate method component for execution. The execution of a method component invokes the execution of its control components, which will direct the method either to solve a task immediately or to decompose the task into subtasks for further execution. During such a chain of executions, related components are plugged together to generate programs for solving the problem. In this way, problem solving process can be viewed as a modeling activity rather than a programming activity.

3. An Object-Oriented Reflective Architecture

This object-oriented reflective architecture is designed to support (a) an extensible mechanism for plugging knowledge level components together, (b) modularity of the execution knowledge for sharing or reuse, and (c) user-definable interpretation process in an open-ended environment.

According to [Rumbaugh, 91], *"Reflective* is a property of a system such that it can examine its own structure dynamically and reason about its own state." A system is said to have a *reflective architecture* if the system stresses the importance of being reflective and thus supports building blocks to achieve that [Maes, 87]. Definitions of some basic terms used in describing a reflective architecture are given below.

A *reflector* is a meta level entity about which a meta system can reason for doing the meta level computation; it also provides causal connection between meta and object level systems. Two systems are *causally connected* [Smith, 82] if they are linked in such a way that changes in one will lead to effects upon

Submitted to 1993 OOPSLA Workshop on Object-Oriented Reflection
and Metalevel Architectures

Object-Oriented Reflection for Plugging Together Problem-Solving Components in an Open-Ended Software Architecture

Ruey-Juin Chang
AI Laboratory
The University of Texas at Austin
cschang@cs.utexas.edu

Abstract

An object-oriented reflection architecture has been designed for an open-ended software environment for automating the construction of computer programs to answer analysis questions. Analysis programs can be specified by selecting and connecting high-level, generic models of problem-solving components. Reflection provides great flexibility in adapting the generic components to be specialized into executable programs in different applications.

1. Introduction

A human analyst who wishes to answer a question such as "What is the total energy production of counties within 100 miles of Austin?" must use several types of reasoning. Geometric reasoning is required to find the counties that are within 100 miles of Austin. One or more databases may contain energy production data in different forms. Physics reasoning may be required to convert one form of data (*e.g.*, barrels of oil, tons of coal) into the other (*e.g.* the energy equivalent).

In traditional programming approaches to problem solving, the control knowledge for generating problem-solving programs is usually embedded in an interpreter or a program generator. The hardwired control knowledge of an interpreter limits its capability in solving related but variant problems, due to the difficulty of predetermining all possible control and composition structures in the problem-solving processes. The more control knowledge is embedded in an interpreter, the less extensibility the interpreter has to handle the variations of related problems. In contrast, the interpreter in our system is generic and extensible in the sense that the kernel of this interpreter is reduced to the minimum with no hardwired control knowledge.