IDEA GROUP PUBLISHING **ITB13691**

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

This paper appears in the publication,
Artificial Intelligence and Integrated Intelligent Information Systems: Emerging Technologies and Applications
edited by X. Zha © 2007, Idea Group Inc.

Chapter XIV

Development of an Intelligent Information System for Object-Oriented Software Design

Gary P. Moynihan, The University of Alabama, USA

Bin Qiao, The University of Alabama, USA

Matthew E. Elam, The University of Alabama, USA

Joel Jones, The University of Alabama, USA

Abstract

The purpose of this research was to apply an artificial intelligence approach to improve the efficiency of design pattern selection used in the development of object-oriented software. Design patterns provide a potential solution to the limitations occurring with traditional software design approaches. Current methods of design pattern selection tend to be intuitive and based on the experience of the individual software engineer. This expertise is very specialized and frequently unavailable to many software development organizations. A prototype expert system was developed in order to automate this process of selecting suitable patterns

to be applied to the design problem under consideration. It guides the designer through the pattern selection process through inquiry regarding the nature of the design problem. The prototype system also provides the capabilities to browse patterns, view the relationship between patterns, and generate code based on the pattern selected. The routine application of such a system is viewed as a means to improve the productivity of software development by increasing the use of accepted design patterns.

Introduction

Software design is an iterative process, such that the requirements are transformed into a model for constructing the software. First, the design is represented at a very high level of abstraction. At this level, the design can be directly traced to specific data, functional, and behavioral requirements. Being an iterative process, the design undergoes changes over the duration of the design phase. Eventually, the design representations are expressed at a much lower level of abstraction. At this level, the connection between the requirements and the design is less obvious (Vliet, 2001).

Object-oriented design differs from traditional (i.e., procedural) software design methods, by achieving a number of different levels of modularity. This modularity can be defined as dividing software into separately named and addressable modules. These components then can be assembled to form the complete system. Although object-oriented software design is frequently viewed as easier than alternative approaches, designing any software is a difficult task, regardless of technique. For example, Jones, Hilton, and Lutz (1998) point out that "the promise of the object-oriented approach to the systems analysis hinges on correctly partitioning the problem domain into essential classes and objects. Most developers agree that this is no easy task." Despite these difficulties, experienced object-oriented designers produce good designs that are hard to achieve for a novice designer. Frequently, they will use recurring patterns of communicating objects and classes. Such patterns are applicable to specific design problems, and thus make the object-oriented designs more elegant, flexible, and ultimately reusable. These patterns allow experienced designers to reuse successful designs by basing new designs on their prior experience.

A software design pattern book by Gamma, Helm, Johnson, and Vlissides (1995) discusses 23 fundamental design patterns carefully selected from numerous object-oriented systems. It is regarded by many as the accepted baseline for this type of approach (e.g., Budinsky, Finnie, Vlissides, & Yu, 1996). Each of these fundamental patterns can be applied only under certain circumstances, so it has its own applicability criteria. Each pattern is also associated with its consequences. A subset of these patterns is related to each other, such that the application of one pattern makes the subsequent application of relevant patterns beneficial to the overall design. These patterns provide a valuable tool for the practicing software design professional. As the benefits of design patterns are becoming more and more apparent, the utilization of these patterns to develop components of reusable object-oriented software has become an emerging trend. Previous research done in this area is now being incorporated for commercial applications. Some of the associated advantages are discussed by Cline (1996). However, of these 23 fundamental design patterns that can be used to develop reus-