Constraint analysis is a design tool for object-oriented database environments that makes use of knowledge about constraints to support the design of user views and the specification of propagation actions. The constraint analysis process is supported by the formal, uniform representation of schema constraints. The formal representation supports the ability to analyze and explain constraints; the uniform representation allows both inherent and explicit constraints to be analyzed in the same manner. Using constraint analysis, the effects of constraints can be made explicit at design time to support the flexible specification of database operations that maintain object integrity. This paper presents the constraint representation that supports the analysis and explanation process.

1 Introduction

An important advantage of object-oriented data modeling is the increased level of semantics that can be directly captured within the data description. Viewing each class in an object-oriented schema as an abstract data type, the operations on each class should protect the semantics of the objects involved [Ditt86]. One problem with object-oriented data modeling, however, is that the increased level of semantics associated with a schema tends to complicate the specification of operations on objects. Furthermore, new applications for object-oriented database systems often emphasize the need for expressing and propagating explicit constraints (e.g., [Du87]), further increasing the complexity of specifying object manipulation operations. As a result, there is a strong need for a constraint management tool to assist in the formation of update operations.

The effect of schema semantics on object operations has previously been addressed in the context of update propagation [Brod84, Hech81, Abit85, Mark86, Brod84]. Previous propagation algorithms, however, perform in a pre-defined manner for all users based on propagation rules for structural schema constraints, ignoring the possibility that different users may require different propagation actions. Furthermore, propagation actions for explicit constraints are rarely addressed [Morg84]. Ideally, an object-oriented design environment should support flexibility in specifying propagation actions for inherent and explicit constraints and should also provide tools to insure that propagation actions are correctly specified.

This paper presents a support tool for an object-oriented design environment known as constraint analysis [Urba87]. Constraint analysis is a component of a constraint management facility that provides a way of actively using constraints as a source of knowledge for the support of object-oriented design activities. In [Shep84], Shepherd and Kerschberg identify constraint management as an essential aspect in the development of more intelligent environments for managing both data and knowledge.

There are five primary contributions of constraint analysis to object-oriented design. First of all, constraint analysis relies on a formal, declarative representation of both inherent and explicit schema constraints by transforming all constraints into Horn clause form\(^1\). Schema constraints are

\(^1\)Constraint analysis is currently applied only to constraints that are expressible in Horn clause form.
therefore uniformly represented in a knowledge base that supports the ability to automatically reason about constraints. A second contribution of constraint analysis is that related clauses are logically ordered to provide an organized approach to the analysis and explanation of constraints. With all constraints collectively represented in a logical, analyzable manner, a third contribution is that the analysis process helps a designer understand the consequences of stating constraints. As a result, errors in the expression of constraints can be detected early in the design phase.

The fourth contribution of constraint analysis is also a direct result of the ability to explain constraints. Through the explanation process, knowledge about constraints can be used to define the scope and content of user views of abstract objects. The explanation of constraints also enhances the specification of update propagation actions, the fifth contribution of constraint analysis. By making the effects of constraints explicit at design time, constraint analysis supports flexibility in specifying update propagation actions so that different user views can respond to constraint violations in different ways. Flexibility in the design of update operations supports a more creative environment for application design.

This paper concentrates on presenting the uniform representation scheme that supports the correct analysis and explanation of constraints. A detailed presentation of using constraint analysis to form user views and update propagation specifications is beyond the scope of this paper and can be found in [Urba87]. The remainder of the paper is organized as follows. Additional motivation for constraint analysis is presented in Section 2. The general analysis process and the representation of constraints is presented in Section 3, followed by an example of the analysis and explanation of constraints in Section 4. Conclusions and future research are presented in Section 5.

2 Motivation

As an example of the utility of constraint analysis, consider the university subschema in Figure 1 and a view of the STUDENT class that allows modification of the major, gpa, and advisor properties. The following constraint states that a STUDENT object with a status of "honor" must have an advisor from the same dept as his or her major and the advisor_status value must also be "honor":

C1: for all S in student (where S.status = 'honor'), there is some F in faculty, such that: (F is S.advisor and S.major = F.dept and F.advisor_status = 'honor');

If an honor student, S1, previously satisfied C1, a number of different propagation actions can be taken to satisfy the constraint when S1's major property is modified. For example, C1 can be satisfied for S1 by modifying the advisor's dept value, or by changing S1's status to a value other than "honor". A more likely action is to assign a new advisor to S1 having the appropriate dept and advisor_status values. Considering the same constraint for a view of the FACULTY class, modifying a faculty member's dept value implies a modification of the major for that faculty member's