IDEA GROUP PUBLISHING
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
ITB12859

This paper appears in the book, *Emerging Trends and Challenges in Information Technology Management, Volume 1 and Volume 2* edited by Mehdi Khosrow-Pour © 2006, Idea Group Inc.

CSpec: Constraint Specification for Data Modeling

Gillian S. Miller, Department of Computing, Macquarie University, Herring Road, North Ryde, NSW 2109 Australia, T: +61 2 9850 7372, F: +61 2 9850 9551, Gillian@ics.mq.edu.au

ABSTRACT

Data modeling allows us to elicit and visualize information about business objects and the relationships between them at a high level of abstraction. We argue that data modeling should be augmented to allow a detailed and rigorous specification of data constraints that represent structural invariants and business rules. We propose a language CSpec which extends OCL in a natural way to allow for constraint specification and advocate the use of a minimal but powerful set of notations to express constraint patterns. Novel features of our language are relational constraints, commuting constraints, concept formation, path predicates and safe transitive relationships.

1. INTRODUCTION

Data models such as Unified Modeling Language (UML) class diagrams [1] and Entity-Relationship (ER) models allow us to elicit and visualize information about business objects and the relationships between them at a high level of abstraction. These models are essentially simple, and offer a minimal but powerful set of constructs to represent this information. While the simplicity of the notation has contributed to the success of data modeling, the notation is limited in the capabilities for representing business data rules. Our interest is in how best to augment data modeling in order to allow for the specification of a greater range of semantic, structural and business rules related to the data model. We believe an appropriate framework should help modelers elicit and then document formally a rich range of structural and semantic rules which are otherwise not recognized, captured informally or are implicit in program code. In this paper we propose a language CSpec (Constraint Specification) for this purpose.

CSpec represents a synthesis of many ideas including the Object Constraint Language (OCL), Object Role Modeling (ORM), the Alloy model checker, the XPath query language, description logics and relational database constraints.

The Object Constraint Language (OCL) [2] allows software developers to write constraints and queries over object models in a declarative and relatively descriptive manner and arose because of the limited semantics of the graphical UML diagrams. Some of the limitations of OCL have been discussed in [3, 4]. Our concern is that OCL does not provide enough data semantics or constraint encapsulation.

There have been arguments about the relative strengths of UML class diagrams for conceptual modeling compared to other methodologies (see for example [5-7]). The methodology Object Role Modeling (ORM) [5] allows for a relatively richer expression of data invariants, ideas which we have incorporated into our framework.

The model checker Alloy [8] provides a rich logical language to express constraints over a class based data model and is not as mathematical as formal specification tools such as Z and B. Description logics [9] allows for the expression of concept formations using complex role paths. There has been considerable research in constraints for relational database theory. Good overviews can be found in [7, 10].

The rest of the paper is structured as follows. Section 2 discusses our requirements for a data modeling specification language. Section 3 summarizes the main language features of our proposed language. In section 4 we describe selected features in more detail, giving examples. Finally in section 5 we give our conclusions and discuss our related research.

2. CONSTRAINT SPECIFICATION REQUIREMENTS

We believe the main requirements for a specification language to augment data modeling are:

- to be usable by information systems practitioners – this audience is not necessarily well schooled in logic, and hence the specification language should provide familiar metaphors and *syntactic sugaring* for the expression of logical and query expressions
- provide a minimal but powerful set of constructs to encapsulate commonly occurring data constraint patterns – following arguments by [11] these notations should express fundamental concepts in a powerful and intuitive way, be *minimal* to protect the simplicity of the modeling process and *powerful* to substantially contribute to the semantics of the data model
- provide for expressive logical and query expressions at an appropriate level
- to serve as enhanced documentation during the analysis and design phase - this requires a balance of conciseness and readability and should include a text based encoding form
- to be declarative and independent of implementation
- to have formal underpinnings so that it can be effectively implemented for data validation and model checking.

3. LANGUAGE OVERVIEW

The following describes our proposed language CSpec. We conceptualize our universe of discourse semantically. The principal building blocks are *concepts* and *association path*s. Concepts are formed from the classifiable elements (e.g. Class) with new concepts formed on the basis of predicate expressions. Association paths are formed by navigating over attributes and read-only methods using path composition, a range of path operators (such as inverses and transitive closures) and path predicates. Constraint primitives are used to describe the following :

- *Association constraints* -. Association constraints include *cardinalities*, *totality* and *partiality* constraints, *domain* and *range* constraints and *inverse* constraints.
- *Database constraints* – these allow the natural expression of typical database constraints such as uniqueness and key constraints, nulls and referential integrity constraints.
- *Association relational constraints* - These constraints allow the definition of ring constraints such as *irreflexiveness*, *reflexivity* or *acyclicity*
- *Commuting constraints* – These are a novel form of constraint that allow the simultaneous comparison of the ends of two association paths

Other novel features of our constraint language include the following :

- *Concept formation* – This allows multiple classification on the basis of predicate expressions
- *Logical sugaring* – The specification language is capable of expressing first order logical expressions in a "*sugared*" form. The use of *context* and *concept* forms together with the rich path

language means that we minimize the explicit use of variables from the underlying expression.

- *Elimination of recursion* – The use of inbuilt transitive operators means that we can safely deal with recursive types

4. LANGUAGE FEATURES

Concept Formation – Multiple Classification

We extend the notion of classifiable elements (e.g. Class) to allow for concept classes that allow categorization in terms of meaningful subsets (persons that are female, flights in a certain country). Each subset is determined by predicate expressions, and can carry a specialized set of constraints. This allows us to extend a rigid class hierarchy to allow for generalized multiple classification (especially if the intended implementation language only supports single inheritance). We provide constructs by which concepts may be named, or implicitly introduced such as the following:

context *classifierName* restricts *conceptNameExpr*
where *condition*
inv: *invEpression* ..

Concept formation are substantive to description logics [9]. Concepts are a natural extension of attribute-defined specialization used in extended entity relationship modeling as explained in [12]. Concepts provide a natural sugaring of the logical implication clause. Moreover, the use of concepts in an associated methodology should elicit a more systematic discovery of constraints

Association Relational Classification

These constraints allow the classification of relationships according to the elementary mathematical classifications. While this classification can be expressed using basic logic, we favor the encapsulation of such constructs as a direct notation. For example we provide constructs such as:

inv : assertReflexive (*featurenameExpr*)
inv : assertAcyclic (*featurenameExpr*)

There are also forms for irreflexive, symmetric, tree, inverse relations and so on. Acyclic associations represent a common constraint pattern. Examples of acyclic associations include management associations, course pre-requisites and meta-modeling inheritance hierarchies.

We believe the inclusion of such notation in CSpec contributes to the readability of the specification. This notation is absent in OCL. For example in [13] we observed at least four types of OCL expressions to express an irreflexive relationship, although these were consistently described in the English annotations.

Transitive Operators

Many associations between concepts have a transitivity property. Examples include employment hierarchies, course prerequisites, meta-modeling inheritance hierarchies, family relationships, connecting flights and so on.

We provide constructs for transitive closures, reflexive transitive closure and ordered closures over sequence based collection types. Furthermore by allowing these transitive closure operators, we can in most cases eliminate recursion from the specification language and hence the need for least fixed point operators.

The following example is taken from [13] where we are meta-modeling the inheritance relationships among Classifiers and the operations parent, allParents and allFeatures. This is encoded in CSPec as follows:

context Classifier
def : parent : Set(Classifier) = generalization.parent
inv : assertAcyclic (parent)
def : allParents : Set(Classifier) = transClosure (parent)
def : allFeatures : Set(Feature) = union (feature, allParents.feature)

We believe that the above specification is much more readable than the equivalent recursive form encoded in OCL as follows.

context Classifier
parent : Set(Classifier);
parent = self.generalization.parent;

allParents : Set(Classifier);
allParents = self.parent->union(self.parent.allParents);

allFeatures : Set(Feature);
allFeatures = self.feature->union(self.parent.oclAsType(Classifier).allFeatures)

Commuting Constraints

Our specification language advocates the use of strong navigational forms. This paves the way for what we call *commuting constraints.* These allow the simultaneous comparison of the ends of two association paths formed relative to some object. We follow two navigational paths which form a cycle (best visualized graphically), and then specify a constraint about the pair of associated sets of objects. The associated sets may be the same, they may be disjoint or related by subset inclusion. The commuting constraints are little known in the literature. The construct to specify an equals commuting constraint is:

context *classifierExpr*
inv : assertEquals (*featureNameExpr1*, *featureNameExpr2*)

The following is an example adapted from [11]. Assume a model that contains Teacher, Class and Department. There are associations teaches between Teacher and Class, offeredBy between Class and Department, and faculty between Teacher and Department. A university rule is that teachers can only teach classes offered by the department in which they are faculty. This is encoded in CSpec as follows:

context Teacher
inv: assertSubset(teaches.offeredBy, faculty)

5. CONCLUSION

We believe that data modeling should be augmented to allow for the specification of a greater range of semantic, structural and business constraints. We propose a language CSpec which has the expressiveness of a query and logical language, but with a primary concern of being able to concisely express logical expressions, invariants and structural constraints. We advocate the use of a minimal but powerful set of high level constructs to encapsulate common constraint patterns. Novel features of our proposed language are concept formation and multiple classification, navigational expressions, database constraints, association constraints, ring association constraints, safe transitive closure operators and commuting constraints. CSpec extends the industry standard OCL language with considerably enhanced data, structural and semantic specification capabilities.

This paper summarizes our research at a high level as it relates to practitioners working in data modeling. As part of our research we are working on an implementation that extends an existing OCL toolkit to

incorporate our language. We also have related theoretical research to investigate formal properties and reasoning over the model.

6. BIBLIOGRAPHY

1. OMG, *OMG Unified Modeling Language Specification; Version 1.5*. 2003, Object Management Group Inc.
2. OMG, *UML 2.0 OCL Specification*. 2003, Object Management Group, Inc.
3. Jackson, D., *A comparison of Object Modelling Notation : Alloy, UML and Z*. 1999, MIT Lab for Computer Science.
4. Cook, S., et al., *The Amsterdam manifesto on OCL UML 2.0 RFI response. Technical Report ad/99-12-22*. 1999, Object Management Group.
5. Halpin, T.A., *Information Modeling and Relational Databases; from conceptual analysis to logical design*. 2001, San Francisco.
6. Halpin, T.A. *Augmenting UML with Fact-orientation*. in *HICCS-34*. 2001. Mauri.
7. Thalheim, B., *Entity-Relationship Modeling; Foundations of Database Technology*. 2000: Springer.
8. Jackson, D., *Alloy: a lightweight object modelling notation*. ACM Transactions on Software Engineering and Methodology (TOSEM), 2002. 11(2): p. 256-290.
9. Baader, F., *The description logic handbook : theory, implementation, and applications*. 2003, New York: Cambridge University Press.
10. Abiteboul, S., R. Hull, and V. Vianu, *Foundations of Databases : the logical level*. 1995, Reading, Massachussetts: Addison Wesley Publ. Co.
11. Badia, A., *Entity-Relationship modeling revisited*. SIGMOD Record (ACM Special Interest Group on Management of Data), 2004. 33(1): p. 77—82.
12. Dietrich, S.W. and S.D. Urban, *An advanced course in database systems : beyond relational databases*. 2005: Pearson Prentice Hall.
13. OMG, *Common Warehouse Metamodel: Version 1.1*. 2003, Object Management Group, Inc.