12

Core objects required for a generic CASE repository

Gordon Manson, Siobhán North and Abdullah Alghamdi
Department of Computer Science, University of Sheffield, Regent Court,
211 Portobello Street, Sheffield, UK. Tel. 0044-114-2825597
EMail: A.Alghamdi@dcs.shef.ac.uk

Abstract

An extendible CASE tools environment is currently being researched at the Department of Computer Science, University of Sheffield. This environment is designed primarily for developing parallel system software but is configurable for other applications. It requires an underlying generic data repository to represent information about the system under development in a consistent and complete form. The representation must be independent of the source, and intended use, of its data.

The paper starts by explaining the importance of a CASE data repository and the overall hierarchical structure of our repository data model. Then, it goes on to suggest a meta-meta data model for a generic CASE repository.

Keywords

CASE tools, MetaCASE Environment, Data Repository

1. INTRODUCTION

The software developer now expects a full CASE tools environment; not just a collection of unrelated CASE tools to support software development, but rather an assembly of integrated CASE tools that link together and provide automated support for all phases of the software life cycle . The data repository is a core technology in reaching high levels of software integration and automation because it provides the underlying framework upon which all the other structures depend (Chen, 1991) (Forte, 1989).

The software engineering environments are perceived, in our system, as a collection of methodologies, each methodology consists of a number of tools (or techniques) and each tool comprises a number of graphical and non-graphical objects used for constructing part of the real life application. In view of this perspective, the overall structure of our CASE repository contents has been divided into a number of different layers of abstraction.

S. Brinkkemper et al. (eds.), *Method Engineering*

The first and most important step in developing a central CASE repository is to design its meta-meta data model. The abstract entities representing the real diagrammatic objects must be chosen very carefully if the repository is intended to be a generic one because whatever is chosen must be able to support all the currently popular methodologies and, hopefully a few that have not yet been invented (Alderson, 1991). To this end a number of methodologies, representing different paradigms and approaches to software engineering, as well as a number of configurable CASE tools have been reviewed. This allowed the development of an abstract meta-meta data model which could represent all of the diverse structures used in the different methodologies.

2. REPOSITORY META DATA LAYERS

To achieve the integration of various CASE tools and to make the environment open to multiple software engineering approaches, the repository data has been divided into a number of different levels of abstraction.

The "Meta-Meta Data Model" is a highly conceptual layer describing what components and capabilities are available for creating meta models. It provides a sufficient degree of abstraction to deal effectively with many CASE environments. This layer provides another degree of freedom for extending the meta model and also simplifies the definition of design rules and integrity checking.

The next level down is the "Methodology Generic Meta Data Model" layer. In our system a software engineering methodology is perceived as a process for the organised production of software using a collection of predefined tools and notational conventions. This layer is concerned with describing the rules required for building a new methodology, its tools and graphical objects.

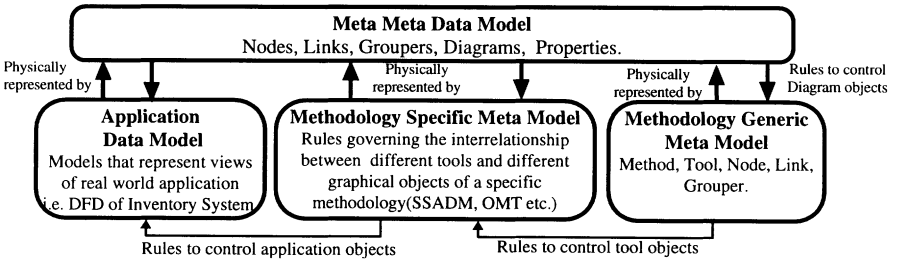

Figure 1: Meta-Meta data model interaction to other meta models in the system

The next layer down is the "Methodology Specific Meta Data Model" layer which describes the interrelationship between the tools and graphical objects of a specific. It contains descriptions of object types such as "process", "data flow" and "control flow", the types of modelling notations and the connectivity rules available to the developer in specifying an application.

The lowest layer is the "Application Data Model" layer which contains the specific models of an application. This layer defines the data flow diagrams, Entity Relationship diagrams, action diagrams, state transition diagrams, etc. that describe the system of interest.

3. THE REPOSITORY META-META DATA MODEL

The meta-meta data model is the collection of primitives used to represent everything stored in a repository because it provides the overall conceptual view of the entire repository contents.

The conceptual objects that have been defined as common to most methodologies are nodes, links and groupers. A *node* represents any diagrammatic concept which can exist independently from any other objects in the diagram, a *link* represents a connection between two (and only two) nodes and cannot exist without both of these two nodes and a *grouper* represents a grouping of interrelated objects (nodes and links).

A real diagram contains a number of diagrammatic objects connected together. These objects are not the basic CASE objects mentioned above but are rather, instances of them. The CASE instance objects store information about the shapes appearing in a diagram. That means that one instance object will be stored in the CASE data repository for each shape in the diagram. Therefore in one diagram there may be many instances of a basic CASE object and other instances of this object may occur in other diagrams.

Sometimes the basic information stored about a CASE object is not sufficient, additional information is needed for a particular application (say code generation). This can be represented by what is called an "Object Property". It allows the analyst to add multi-level structured information to the object, thus making its definition extensible.

Our repository meta-meta data model is represented using Entity-Relationship-Attribute (ERA) model. All the objects outlined above are defined in terms of entities participating in relationships.

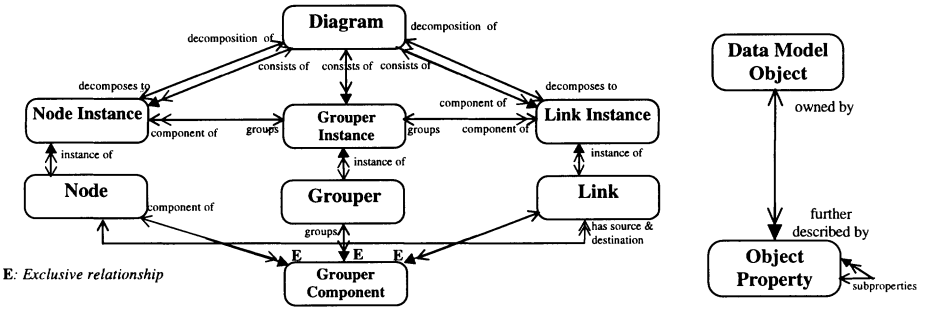

Figure 2: The Repository Meta-Meta Data Model

Two main types of relationships are defined; mandatory and optional. Diagrammatically a mandatory relationship is represented by an arrow with solid triangular arrow head and an optional one is represented by ordinary arrow head. A mandatory relationship implies that the destination entity is dependant on the relationship for its existence.

As can be seen from figure 2, the model consists of a diagram entity, which is a collection of graphical instances, a number of basic objects (node, link and grouper) and their instances. Each one of these entities is known as a data model object. Each data model object can be further described by a number of multi-level properties as shown in the same figure (right).

The model does not show all the entities represented in the repository. A number of entities were omitted for simplicity. Some of these entities support reusability, some of them are to support object versioning and configuration management and some are for query optimisation, object-symbol mapping and object security.

Basic CASE objects

Any concept used during the analysis and design stages and represented in a diagram will be stored as a CASE basic object in the CASE repository. The advantages gained from applying such an approach include data persistency, reusability and reduction of data redundancy.

Data persistency is an important concept and, in the context of CASE systems, has two different, but related, aspects. Firstly, it allows information stored in the repository to outlive the process that created it, and secondly, it provides for the possibility of basic information surviving in the repository after the deletion of its instances.

If an object is to be reused by another diagram it must not have any visual or relationship information tying it to a specific diagram and the only objects which have these characteristics are the basic ones. So we also need basic objects to achieve re-usability.

In some applications, more than one instance of the same object might share a number of similar properties. If those properties are duplicated with a copy in each of the instances then the usual problems of data redundancy will occur. This difficulty can be overcome by having a central basic object containing properties common to all instances and allowing each instance to have its own extra properties.

Object Instances

Each basic object may have more than one instance object which represents it visually in a diagram; that means one instance object will be stored in the CASE data repository for each shape in a diagram. The idea behind this approach is to enable the CASE tools to perform consistency checking between diagrams, retrieve graphical information about the CASE objects and optimise the reusability of the basic objects.

Object Property

The object property provides additional, multi-level, explanatory information about an object in the repository data model. By applying this new technique, we have come up with a highly flexible hierarchy of properties. In this hierarchy, a property (or even a group of properties) can be moved around, copied, added to or deleted from any position in any level in the hierarchy without affecting the whole structure.

These characteristics are not offered by most of the available commercial meta-CASE tools environments. The absence of these characteristics in a similar project (Manson, 1994) made automatic code generation very difficult because of the lack of detailed information about the data and programming constructs. It is for this reason we have adopted such a flexible structure.

During the very early stages of designing our repository, the properties were exclusive to the basic objects, but after going through a number of real case studies (Alghamdi, 1994) we discovered that the instances need some extra visual and non-visual information which could only be stored as properties. For that reason we have decided that the instances should have their own properties in addition to those stored in their original basic objects.

4. CONCLUSION

The meta-meta data model described here is the overall conceptual view of the entire repository contents where the CASE objects are defined in terms of entities participating in binary relationships. The conceptual basic objects that have been defined as common in most methodologies are nodes, links and groupers. A diagram consists of a number of interrelated diagrammatic objects connected together. These objects are not the basic CASE objects but are rather instances of them. The CASE instances store information about the shapes appearing in a diagram.

Sometimes the basic information stored about a CASE object is not enough, additional information is needed to further describe the object in order to be sufficient for some application purposes. In fact this can be done through what is called an "Object Property". The multi-level object property allows the analyst (or designer) to add more than one level of structured information to the object. This makes the object definition extensible.

The fundamental advantage of this approach to CASE system design is that it permits reusable basic data objects to be defined and instances of these objects to inherit their properties and methods. Moreover, it allows a flexible hierarchy of properties to be constructed for each data model object in the system. It is this flexibility that gives our approach an advantage over more traditional approaches.

5. REFERENCES

Alderson, A. Meta-CASE. (1991)Lecture notes in computer sciences 509, Springer-Verlag

Alghamdi, A. (1994) An extendible MetaCASE repository. Transfer Report, Department of Computer Science, University of Sheffield

Chen, Minder and Edgar Sibley, (1991)Using CASE Based Repository for Systems Integration. Proceedings of The Hawaii International Conference on Systems Sciences,.

Forte, Gene, (1989) Inside the CASE Repository. CASE Outlook, No 4 Dec.

Manson, G. A. Sahib, S. and Elamvazauthi, C.(1994) "Design and code derivation in the PCSC methodology" Information and Software Technology journal, July.