

Dynamic Relationships in Object Oriented Databases: A Uniform Approach

Awais Rashid¹, Peter Sawyer¹

¹Computing Department, Lancaster University, Lancaster LA1 4YR, UK
{marash, sawyer} @comp.lancs.ac.uk

Abstract. In this paper we present a uniform approach to dynamic relationships in object oriented databases. We present our relationship categorisation based on dividing the object database into three *virtual* spaces each hosting *entities* of a particular type and show how relationships from the modelling domain map onto relationships in our categorisation. We present a relationship model and the semantics of relationships. The relationship model is complemented with a meta-model for implementing dynamic relationships in an object oriented database. The applicability of the dynamic relationships approach is explored by employing it to implement the database model for a system in order to achieve dynamic schema modification capabilities.

1 Introduction

Relationships are at the heart of the relational model [6] and semantic data models such as the entity relationship model [5]. They also play a key role in object oriented modelling techniques such as UML [2, 16], OMT [23] and SAMMOA [10]. Unlike their relational peers, relationships in object models exist among objects and not attributes. A relationship is: “an abstraction stating that objects from certain classes are associated in some way; the association is given a name so that it can be manipulated. It is a natural concept used in ordinary discourse” [21].

Object databases are considered to be suitable for supporting complex applications such as computer-aided design (CAD), etc. In order to address these complex applications involving correlations across data representations, object databases must incorporate functionality to create networks of semantically related objects. In addition, change propagation and referential integrity should be managed by the DBMS [17]. The highly interactive and evolutionary nature of these complex applications dictates the need for relationships to be dynamic. Facilities should be available to dynamically define new relationships in which instances of existing classes can participate. It should be possible to dynamically modify or remove any relationship definition and to dynamically add and remove participants from a specific instance of a relationship. Applications and objects created using existing classes for which a new relationship is defined or an existing relationship definition removed or modified should not become invalid.

In this paper we present our approach to dynamic relationships in object oriented databases. Object oriented modelling techniques identify several kinds of relationships such as *association*, *aggregation* and *inheritance*. Concrete instances of these relationships occur within an object oriented database. For example, the *defines/defined-in* relationship between a class and its members is an aggregation relationship. The *derives-from/inherits-to* relationship between a sub-class and its super-class is an inheritance relationship. Our relationships approach treats these concrete relationships uniformly regardless of whether they are association, aggregation or inheritance relationships.

2 Relationships: Objects and Meta-objects

Relationships in our approach are based on the observation that three types of entities exist within an object database. These are: **objects**, **meta-objects** and **meta-classes**.

Entities of each type reside within a specific *virtual space* within the database. Objects reside in the *object space* and are instances of classes. Classes together with defining scopes, class properties, class methods, etc. form the *meta-object space* [20]. Meta-objects are instances of meta-classes which constitute the *meta-class space*. Fig. 1 shows the object space, meta-object space and meta-class space in an object oriented database. For simplification we have only elaborated the *instance-of/has-instance* relationship which is represented by the solid arrows.

From fig. 1 we observe that relationships in an object oriented database can be categorised as: ***inter-space relationships*** and ***intra-space relationships***.

Entities participating in inter-space relationships reside in different virtual spaces. Inter-space relationships can be further classified as: **relationships among meta-objects and objects** and **relationships among meta-classes and meta-objects**.

The *instance-of/has-instance* relationship in fig. 1 is an example of inter-space relationships. Note that the *instance-of/has-instance* relationship is a one-to-many relationship directly since each class can have many instances. It is, however, a many-to-many relationship indirectly because the substitutability semantics of object orientation mandate that an object is not only an instance of a particular class but also an instance of its super-classes. The same applies to meta-classes and meta-objects. We regard the *instance-of/has-instance* relationship as a one-to-many relationship.

Intra-space relationships are those where the participating entities belong to the same virtual space. These relationships can be further classified as: **relationships among objects**, **relationships among meta-objects** and **relationships among meta-classes**.

Fig. 1 also shows the virtual spaces that will be of interest to different *actors* [2, 16]. A novice application developer is interested in achieving object persistence and manipulating relationships among the persistent objects. S/he, therefore, will be interested in the object space and its intra-space relationships. An experienced application developer or a database administrator, on the other hand, is interested in manipulating all three spaces and the various inter and intra-space relationships. For example, an experienced application developer might be interested in evolving the schema of the database by adding or dropping meta-objects or by modifying existing intra-space relationships for the meta-object space. S/he may also be interested in introducing some new meta-classes and their instances hence introducing new inter-space relationships between the meta-class space and the meta-object space. We will discuss the introduction of new meta-classes in section 4. We will now discuss how the conceptual division of the database into virtual spaces helps shield a novice application developer from information not of interest to him/her.

Fig. 3 shows a simple UML object model [2, 16] to be implemented in an object oriented database. Our approach allows the novice application developer to define the class hierarchy in the database at the same level of abstraction as UML and to transparently manipulate the persistent objects and the relationships among them. Once the class hierarchy has been defined the system extracts the various explicit and implicit relationships in the hierarchy and generates the corresponding meta-objects and the various inter and intra-space relationships.

Fig. 2 shows the database from an experienced application developer's viewpoint. For simplification the object-space has been omitted. Note that an implicit aggregation

relationship exists between a class and its members (methods and properties: attributes and relationships) which is extracted by the system in the form of *defines/defined-in*. Similarly, the inheritance relationship is extracted in the form of *derives-from/inherits-to*. The system also extracts the implicit recursive relationship *is-inverse-of* which exists between the two edges of a relationship. The implicit *has-type/is-type-of* relationship between a property and its data type is also extracted. Note that *defines/defined-in* is a many-to-one relationship, *derives-from/inherits-to* is a many-to-many relationship (our approach supports multiple inheritance) and *has-type/is-type-of* is a one-to-many relationship. We will discuss cardinalities, recursive nature of relationships and other semantics in the next section. Fig. 2 also shows the system generated *has-instance/instance-of* relationships between meta-classes and meta-objects. It is necessary for the system to extract the various types of relationships since they have different propagation semantics. The system can then provide propagation semantics specific to each type of relationship at the same time treating the various relationship types in a uniform fashion.

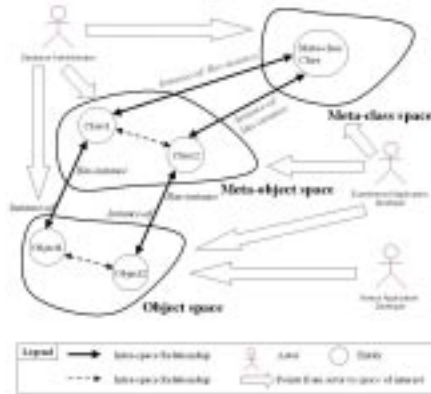


Fig. 1. Virtual Spaces in an object oriented database

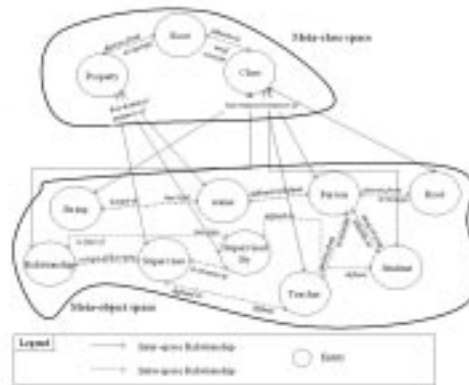


Fig. 2. An experienced developer's view of the database (the meta-class space and the meta-object space)

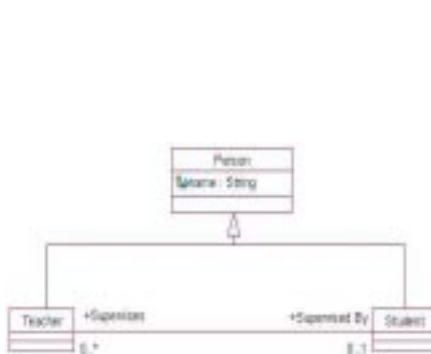


Fig. 3. UML object model for a simple application

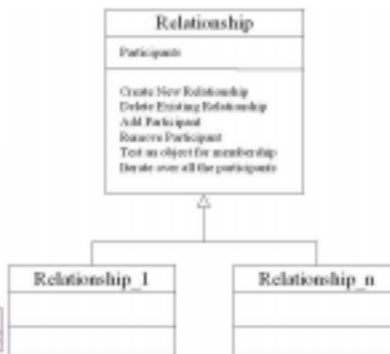


Fig. 4. The Relationship Model

3 THE RELATIONSHIP MODEL

This section first describes the various features desirable of relationships in an OODB. This is followed by a description of our relationship model and the meta-model used to

implement the relationship model in an OODB. Change propagation, referential integrity and the semantics of relationships are also discussed.

3.1 Desirable Relationship Features

Relationships in an ODBMS should satisfy the following set of requirements:

- Relationships should be dynamic i.e. it should be possible to dynamically define new relationships or remove existing relationship definitions. It should also be possible to dynamically modify a relationship definition and to dynamically add and remove participants from a specific instance of a relationship.
- Relationships can be n-ary.
- Relationships should have inverse traversal paths and these should be implicitly created by the DBMS except for uni-directional relationships. For a bi-directional relationship the application explicitly creates the relationship in one direction and the ODBMS implicitly sets the relationship in the opposite direction.
- Relationships can have a cardinality, either one-to-one, one-to-many, or many-to-many; many-to-many relationships cannot be uni-directional [17].
- Relationships can have ordering semantics [17].
- Relationships can be recursive. Participants of a recursive relationship are instances of the same class.
- Relationships can have different propagation semantics depending on whether they are aggregation, inheritance or association relationships.
- Change propagation and referential integrity for bi-directional relationships should be maintained by the ODBMS.

In section 3.3 we will discuss how closely relationships in our approach satisfy the above set of requirements.

3.2 The Relationship Model

Relationships in our approach are both semantic constructs and first class objects. Fig. 4 depicts the class Relationship used in our system. The model also outlines the various operations that can be applied to a relationship. This is the model visible to all actors and is of particular interest to a novice application developer.

The class *Relationship* is an abstract class. Relationships are created as instances of classes *Relationship_1* and *Relationship_n*. These instances act as edges of the relationship; an instance of class *Relationship_1* represents an edge cardinality of one while an instance of class *Relationship_n* represents an edge cardinality of many.

Fig. 8 shows an example scenario: the many-to-one *supervises/supervised-by* relationship from fig. 3. The domain of *participants* is *student* for the *supervises* edge of the relationship and *teacher* for the *supervised-by* edge. The traversal paths for the relationship are managed transparently of the application programmer. It should also be noted that the *Relationship* instances are *existence dependent* on the objects related through them; the *Relationship* instance representing the edge of a relationship ceases to exist once the object that employs it to participate in the relationship dies.

3.3 The Meta-Model

We now present our meta-model that can be employed to implement the relationship model described in section 3.1. This is the model that is visible to an experienced application developer or a database administrator and not to the novice developer. Relationships are treated in a uniform fashion regardless of whether they are inter-space or intra-space relationships. Intra-space relationships are treated in the same manner whether they exist within the object space, the meta-object space or the meta-

class space. No distinction is made between inter-space relationships; i.e. relationships among meta-objects and objects are treated the same way as those that exist between meta-classes and meta-objects.



Fig. 5. The class *Relationship Structure*



Fig. 6. The class *Root* from which all classes and meta-classes inherit

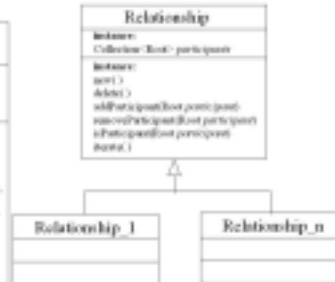


Fig. 7. The Relationship Model implemented

In order to treat relationships in a uniform manner, all classes and meta-classes in the system inherit from the class *Root* which is shown in fig. 6. As shown in fig. 2 the class has replicas existing in both the meta-class space and the meta-object space. The meta-classes inherit from the meta-class *Root* which exists in the meta-class space while the classes (which are meta-objects) inherit from the meta-object *Root* which resides in the meta-object space and is an instance of the meta-class *Class*. Inheriting from the *Root* replica in the meta-class space allows dynamic introduction, removal and modification of relationships in which meta-objects participate while inheriting from the *Root* replica in the meta-object space allows dynamic introduction, removal and modification of relationships in which the various objects participate.

We now discuss how the class *Root* allows dynamic introduction, removal and modification of relationship definitions. Note that the **class:** tag in fig. 6 indicates class-level (static) properties and methods while the tag class/meta-class indicates that the value of the parameter has to be a class (which is a meta-object) or a meta-class. Inheriting from the class *Root* allows the particular class or meta-class to use the class-level (static) methods *createNewRelationship* and *deleteExistingRelationship*. These methods allow the dynamic addition or deletion of relationships in which instances of the class or meta-class participate. It should be noted that the inverse traversal path is implicitly created when *createNewRelationship* is invoked. Uni-directional relationships can be created by specifying a null value for the last three parameters of *createNewRelationship*. Recursive relationships can be set up by specifying the same class or meta-class on which the method is being invoked as an inverse. The inverse traversal path is implicitly deleted when *deleteExistingRelationship* is invoked on one of the classes or meta-classes whose instances participate in the relationship. Later in this section we discuss how propagation of this change to the affected instances is managed by the system.

Fig. 6 also shows the class-level (static) attribute *Relationships* which is inherited by all the sub-classes of the class *Root*. This class-level attribute is a *List* of instances of the class *Relationship Structure*. For each relationship, which the instances of a class or meta-class participate in, a *Relationship Structure* instance is created and appended to the *List* of *Relationships*. These instances are used to store and dynamically modify the structure of the relationship. The class *Relationship Structure* is shown in fig. 5. The

instance: tag in fig. 5 indicates instance-level properties and methods. Note that the *Relationship Structure* class does not inherit from the class *Root*. The instance-level methods *setName*, *setInverseRelationship* and *setEdgeCardinality* can be invoked on instances of the class in order to dynamically modify the structures of relationships whose details they store.

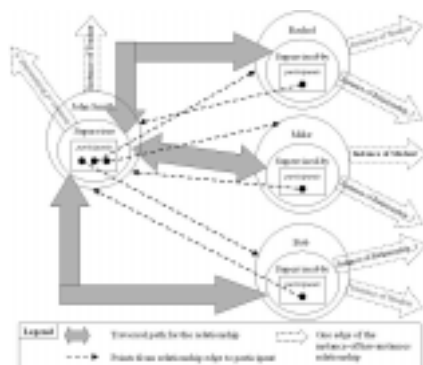


Fig. 8. Instances of *Relationship_1* and *Relationship_n* representing edges of a many-to-one relationship

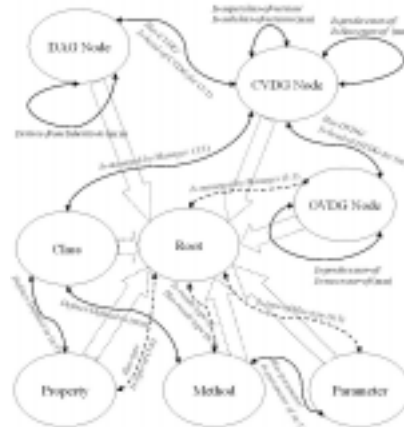


Fig. 9. Meta-classes in SADES and the relationships their instances participate in

Fig. 7 shows the relationship classes in the meta-model corresponding to the classes in the relationship model presented in fig. 4. Note that ordering semantics for relationships can be provided by using an ordered collection, such as a list or an array, for *participants*. Almost all object database management systems offer facilities to create ordered collections and provide methods such as *addAsFirst*, *addAsLast*, *addAtSpecifiedPosition*, etc. to operate on them. Note that only the instances of classes or meta-classes that inherit from the class *Root* can participate in a relationship. Also note that the relationship classes in fig. 7 do not inherit from the class *Root*.

We now describe the semantics of relationships in our approach and discuss how closely they meet the requirements set out in section 3.1. We also discuss how we address the various evolution problems arising from dynamically defining new relationships or modifying or deleting an existing relationship definition.

Our approach allows all actors to introduce, modify and delete relationship definitions dynamically and in a uniform fashion regardless of whether they are interested in the object space only or the meta-object space and meta-class space as well. Furthermore, it is possible to dynamically add and remove participants from existing instances of a relationship. Again, this is managed in a uniform fashion regardless of the nature of relationships whether inter-space or intra-space. Inverse traversal paths are created and managed transparently. The cardinality semantics described in section 3.1 are fully supported besides allowing ordering semantics for relationships. Our approach also allows the creation of recursive relationships. Change propagation and referential integrity is managed transparently of the user. Our approach, however, only supports binary relationships as n-ary relationships can be implemented as a layer on top of binary relationships.

We now discuss the propagation semantics for inheritance, association and aggregation relationships. It should be noted that the user does not need to concern himself/herself

with these propagation semantics. The three types of relationships can be manipulated by the user in a uniform manner.

Aggregation: [26] provides a discussion of relationship semantics. For aggregation relationships it provides a number of propagation alternatives to choose from. We have chosen to propagate operations from aggregate to part, with deletion of the aggregate resulting in deletion of the parts, but not vice versa. It should be noted that the aggregation relationship is transitive in nature; i.e. an aggregate has parts which may in turn have parts [26].

Association: An association relationship requires that changes to one edge of the association are automatically propagated to the other edge. Cardinality semantics need to be preserved. Our approach transparently propagates changes from one edge of the association to the other, so preserving the cardinality.

Inheritance: Inheritance has special propagation semantics. Changes to the structure of a class need to be propagated to its sub-classes. We have chosen a class versioning [12, 13, 19, 24] approach to propagate changes along inheritance relationships. Class versioning avoids invalidating objects and applications created using class definitions prior to the change. It, therefore, addresses the issues that arise when adding a new relationship definition to a class or modifying or deleting an existing relationship definition. Whenever a change is made to the structure of a class a new class version is created for the class and all its sub-classes. Objects and applications created using the older class versions remain accessible through their respective version. They can then be attached to the new class version. [19] proposes a framework for such re-association. Our class versioning taxonomy is the subject of a forthcoming paper.

4 Application to Evolution

[9] characterises relationships among classes as static since these are fixed at compile-time. Relationships among instances are comparatively dynamic in nature and can be changed at run-time. Our approach differs from the viewpoint presented by [9]. Relationships among classes do not need to be fixed at compile-time. These can be dynamic in nature and can be changed at run-time. Therefore, the schema of an object-oriented database can be dynamically modified if the various meta-objects (classes, etc.) that form the schema are interconnected through dynamic relationships. Dynamic schema changes can be made by dynamically modifying the various relationships in which the meta-objects participate. These can be the *derives-from/inheritance-to* relationships between classes or the *defines/defined-in* relationships between classes and their members. If relationships exist among meta-objects and objects they can be used to propagate schema changes to the affected objects.

We have employed our dynamic relationships approach to implement the database model and achieve dynamic schema modification capabilities for our **Semi-Autonomous Database Evolution System**, SADES [18]. SADES is being built as a layer on top of the commercially available object database system *Jasmine* from *Computer Associates International* and *Fujitsu Limited*. It provides support for: class hierarchy evolution, class versioning, object versioning and knowledge-base/rule-base evolution.

Dynamic relationships are needed to provide the above-mentioned evolution facilities. This serves a two-fold purpose. First, evolution can be achieved dynamically in a uniform fashion. Second, a coherent view of the conceptual structure of the database is provided making maintenance easier.

The SADES conceptual schema [19] is a fully connected directed acyclic graph (DAG) depicting the class hierarchy in the system. SADES schema DAG uses *Version derivation graphs* [11]. Each node in the DAG is a *class version derivation graph*. Each node in the class version derivation graph (CVDG) keeps: *reference(s) to predecessor(s)*, *reference(s) to successor(s)*, *reference to the versioned class object*, *descriptive information about the class version such as creation time, creator's identification, etc.*, *reference(s) to super-class version(s)*, *reference(s) to sub-class version(s)*, and a set of *reference(s) to **object version derivation graph(s)***.

Each node of a CVDG keeps a set of *reference(s) to some object version derivation graph(s) (OVDG)*. An OVDG is similar to a CVDG and keeps information about various versions of an instance rather than a class. Each OVDG node [11] keeps: *reference(s) to predecessor(s)*, *reference(s) to successor(s)*, *reference to the versioned instance*, and *descriptive information about the instance version*.

Since an OVDG is generated for each instance associated with a class version, a set of OVDGs results when a class version has more than one instance associated with it. As a result a CVDG node keeps a set of references to all these OVDGs.

Fig. 9 identifies the various meta-classes in the system and the relationships in which their instances, the meta-objects, participate. The dashed block arrows indicate that all meta-classes inherit from the class *Root*. The line arrows, both solid and dashed, represent the various relationships meta-objects participate in. A solid arrow pointing from a meta-class back to itself indicates a recursive relationship. The dashed line arrows representing relationships among certain meta-classes and the class *Root* indicate that the relationship exists among instance(s) of the particular meta-class and instance(s) of a sub-class of the class *Root*. An instance of the class *OVDG Node*, for example, manages an object which is an instance of a sub-class of the class *Root* (since all classes inherit from the class *Root*). Similarly, a class *Property* or a method *Parameter* will normally have a sub-class of the class *Root* as its type.

The various relationships among instances of meta-classes shown in fig. 9 can be dynamically modified to achieve dynamic schema modifications. It is possible to introduce new relationships and even new meta-classes if desirable. For example, it might be desirable for an experienced application developer trying to extend the database model with active features [7] to introduce a meta-class *Rule* which can then be used to define classes (meta-objects) of rules which in turn will be used to create the various rule objects. Schema changes are propagated to the instances using the *has-OVDG/is-head-of-OVDG-for* relationship that exists between a class version and its associated OVDGs. Class versioning allows dynamic schema modifications without rendering existing instances invalid. Instances associated with one class version can be associated with another one using the framework we described in [19].

5 Related Work

Pioneering work to extend OO models with explicit relationships has been carried out by [1, 21]. [21] discusses incorporating relationships as semantic constructs in an object oriented programming language. [1] proposes inclusion of relationships and declarative constraints on relationships in an object oriented database programming language. [22] addresses propagation of operations by defining propagation attributes on relationships. [26] aims at incorporating relationship semantics into object-relational data models. Although the approach by [26] is applicable to pure object oriented data models the applicability has not been explored. All the work concentrates on semantics

and propagation. The dynamic nature of relationships has not been considered. Neither is there an attempt to treat the various types of relationships in a uniform fashion or to extract the implicit relationships that exist within an object oriented model.

The object database standard, ODMG [3, 4], mandates both uni-directional and bi-directional binary relationships as semantic constructs in a database schema. The ODMG specification requires the DBMS to maintain the referential integrity of these semantically related objects. Relationships are treated in a uniform manner whether they exist among meta-objects or objects. Although relationships in ODMG are semantic constructs, they are not first class objects. Neither are they dynamic in nature. ODMG compliant bindings of commercially available object database management systems POET [15], Versant [25] and O2 [14] support binary relationships and maintain the referential integrity for bi-directional relationships. Referential integrity for uni-directional relationships is left to the application. Although these ODBMSs offer dynamic modifications to the database schema [20] dynamic creation, deletion and modification of relationships are not possible as relationships are not semantic constructs in the schema. Instead, they are mapped onto class attributes. Meta-objects in these systems are not semantically related either. As a result an actor such as an experienced application developer or a database administrator tends to lose sight of relationships in the system. Also, it makes maintenance a problem because it is easy to make errors.

In contrast to ODMG which mandates binary relationships among the various objects and meta-objects in the system, the Iris object oriented database management system [8] uses n-ary, inverse traversable relationships to model information about objects and meta-objects residing in the database. Referential integrity is maintained by the DBMS. In contrast to the ODMG compliant systems discussed earlier, both objects and meta-objects in Iris are semantically related. Types are treated as first class objects. Therefore, a conceptual equivalence exists between relationships among meta-objects and relationships among objects. Each type bears relationships to its sub-types, super-types and instances. However, relationships are not semantic constructs in Iris. Furthermore, relationships in Iris are not highly dynamic in nature. For example, new sub-type/super-type relationships among existing types cannot be created.

Relationships in our approach are both semantic constructs and first class objects. In contrast to the above-mentioned approaches it is possible to introduce, modify and delete relationship definitions dynamically. Relationships are treated in a uniform fashion regardless of their type. Participants can be added and removed dynamically from specific instances of a relationship with referential integrity and change propagation managed by the system. Propagation semantics for aggregation, inheritance and association relationships are provided by the system transparently of the various actors.

6 Summary and Conclusions

We have presented an approach to incorporate dynamic relationships in an object oriented database. The novelty of the work is in the dynamic nature of relationships and on treatment of the various relationships in the system in a uniform fashion. We have divided the database into virtual spaces in order to present an actor with relationships of interest to him/her. We have also presented a relationship categorisation on the basis of this virtual division and have automated the extraction of the various explicit and implicit relationships in the modelling domain in order to map

them onto our relationship categorisation. Our approach allows new relationships to be defined dynamically and to remove or modify existing relationships definitions. We have used a class versioning approach to ensure that applications and objects created prior to the introduction, removal or modification of a relationship definition do not become invalid. Participants can be added and removed dynamically from instances of the various relationships.

We have also presented the semantics of relationships in our system and the various operations that can be applied to them. A meta-model has been presented which is used to implement dynamic relationships in an object oriented database. The meta-model can be layered on top of any existing object database management system hence incorporating dynamic relationships into the underlying system.

We have explored the practicability of our relationships approach by implementing the database model for the SADES system and exploiting the dynamic nature of relationships to achieve dynamic schema modification facilities.

Future directions include incorporating n-ary relationships into the system. These will be implemented as an additional layer on top of binary relationships. We plan to explore the applicability of the approach to object-relational data models and to employ learning mechanisms to extract implicit and explicit constraints on relationships in a high level object oriented design and incorporate these constraints into our system without affecting its ability to treat the various types of relationships in a uniform manner.

References

- [1] Albano, A., Ghelli, G., Orsini, R., "A Relationship Mechanism for a Strongly Typed Object-Oriented Database Programming Language", Proc. of the 17th Int. Conf. on Very Large Databases, Sept. 1991, pp. 565-575
- [2] Booch, G., Jacobson, I., Rumbaugh, J., "The Unified Modelling Language Documentation Set", Version 1.1, Rational Software Corp., c1997
- [3] Cattell, R. G. G., *et al.*, "The Object Database Standard: ODMG-93 Release 1.2", Morgan Kaufmann, c1995
- [4] Cattell, R. G. G., *et al.*, "The Object Database Standard: ODMG 2.0", Morgan Kaufmann, c1997
- [5] Chen, P. P., "The Entity-Relationship Model - Toward a Unified View of Data", ACM Transactions on Database Systems, Vol.1, No.1, Mar. 1976, pp.9-36
- [6] Codd, E., "A Relational Model for Large Shared Data Banks", Communications of the ACM, Vol.13, No.6, Jun. 1970, pp.377-387
- [7] Dittrich, K. R. *et al.*, "The Active Database Management System Manifesto: A Rulebase of ADBMS Features", Proc. of the 2nd Workshop on Rules in Databases, Sept. 1995, LNCS, T. Sellis (ed.), Vol. 985, pp. 3-20
- [8] Fishman, D. H. *et al.*, "Iris: An Object Oriented Database Management System", ACM Transactions on Office Information Systems, Vol.5, No.1, 1987, pp.48-69
- [9] Gamma, E. *et al.*, "Design Patterns - Elements of Reusable Object-Oriented Software", Addison Wesley, c1995
- [10] Hesse, W., Mayr, H. C., "Highlights of the SAMMOA Framework for Object Oriented Application Modelling", Proc. of 9th Int. Conf. on Database & Expert Systems Applications, Aug. 98, LNCS 1460, pp. 353-373
- [11] Loomis, M. E. S., "Object Versioning", Journal of Object Oriented Programming, Jan. 1992, pp. 40-43
- [12] Monk, S., Sommerville, I., "Schema Evolution in OODBs Using Class Versioning", SIGMOD Record, Vol. 22, No. 3, Sept. 1993, pp. 16-22
- [13] Monk, S., "A Model for Schema Evolution in Object-Oriented Database Systems", PhD Thesis, Computing Department, Lancaster University, 1993
- [14] "The O2 System - Release 5.0 Documentation", Ardent Software, c1998
- [15] "POET 5.0 Documentation Set", POET Software, c1997
- [16] Quatrani, T., "Visual Modelling with Rational Rose and UML", Addison Wesley, c1998
- [17] Rashid, A., "An Object Oriented View of the Department of Computer Science", MSc Dissertation, University of Essex, September 1997
- [18] Rashid, A. & Sawyer, P., "SADES - A Semi-Autonomous Database Evolution System", Proc. of 8th Int. Workshop for PhD Students in Object Oriented Systems, Jul. 98, ECOOP '98 Workshop Reader, LNCS 1543
- [19] Rashid, A. & Sawyer, P., "Facilitating Virtual Representation of CAD Data through a Learning Based Approach to Conceptual Database Evolution Employing Direct Instance Sharing", Proc. of 9th Int. Conf. on Database and Expert Systems Applications, Aug. 1998, LNCS 1460, pp. 384-393
- [20] Rashid, A. & Sawyer, P., "Evaluation for Evolution: How Well Commercial Systems Do", Accepted at the 1st Int. Workshop on OODBs to be held in conjunction with ECOOP '99, June 14-18, Lisbon, Portugal
- [21] Rumbaugh, J., "Relations as Semantic Constructs in an Object-Oriented Language", SIGPLAN Notices, Vol.22, No.12, 1987, pp.466-481
- [22] Rumbaugh, J., "Controlling Propagation of Operations using Attributes on Relations", SIGPLAN Notices, Vol. 23, No. 11, 1988, pp. 285-296
- [23] Rumbaugh, J., *et al.*, "Object Oriented Modelling and Design", New Jersey, Prentice-Hall Inc., 1991
- [24] Skarra, A. H. & Zdonik, S. B., "The Management of Changing Types in an Object-Oriented Database", Proc. of the 1st OOPSLA Conference, Sept. 1986, pp.483-495
- [25] "Versant Manuals for Release 5.0", Versant Object Technology, c1997
- [26] Zhang, N., Haerder, T., Thomas, J., "Enriching Object-Relational Databases with Relationship Semantics", Proc. of the 3rd Int. Workshop on Next Generation Information Technologies and Systems (NGITS), Israel, 1997