JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2003**

Vol. 2, No. 2, March-April 2003

Extending ODMG Metadata to Define External Schemas

Manuel Torres, Departamento de Lenguajes y Computación. University of Almeria, Spain
José Samos, Departamento de Lenguajes y Sistemas Informáticos. University of Granada, Spain

Abstract

Given that ODMG 3.0 specifications do not address the definition of external schemas, we are developing an external schema definition methodology for ODMG databases. In this paper, an extension of ODMG metadata is proposed to support the definition of external schemas. In particular, metadata for derived classes and derived interfaces are defined, as well as some modifications to define inheritance relationships depending on the schema. The extension to ODMG metadata proposed in this paper maintains the structure of the ODMG schema repository as an object-oriented schema, and can be used by most of the existing external schema definition methodologies. The proposed extension is illustrated using our methodology.

1 INTRODUCTION

One of the main drawbacks of object-oriented databases (OODB) was due to a lack of a standard for such databases, but with the emergence of the ODMG standard, this problem started to be solved. However, ODMG specifications do not address the definition of external schemas in ODMG databases. External schemas correspond to the view level of the well-known ANSI/SPARC architecture. In the ANSI/SPARC architecture, a database can be seen at three levels, known respectively as physical, logical, and view. For each level, there is a schema: internal, conceptual, and external. The internal schema describes the storage structures of the database. The conceptual schema describes the logical model of the database. External schemas describe different views of a database for particular users or groups of users, providing features such as logic independence, authorization, and integration of heterogeneous databases [Scholl91, Bertino92, Motschnig96].

The definition of external schemas in OODBs has been studied deeply since mid 1980s, but most of the proposals are not based on a standard object model [Abiteboul91, Scholl91, Bertino92, Rundensteiner92, Kim95, Samos95, Santos95, Guerrini97]. On the

Cite this article as follows: Manuel Torres, José Samos: "Extending ODMG Metadata to Define External Schemas", in *Journal of Object Technology*, vol. 2, no. 2, March-April 2003, pp. 183-202. http://www.jot.fm/issues/issue_2003_03/article5

other hand, existing proposals for ODMG either do not solve the problem completely [Dobrovnik93], or extend the object-oriented paradigm introducing a new dimension in external schemas [Garcia02]. So, we are currently developing an external schema definition methodolgy for ODMG databases [Torres00, Torres01a, Torres01b] without extending the object-oriented paradigm.

The development of such a methodolgy requires the existence of mechanisms for defining *derived classes* and *derived interfaces* to customize existing classes and interfaces (derived or not), which are known as their *base classes* and *base interfaces*, respectively. In addition, metadata corresponding to such concepts must also be defined. However, ODMG specifications do not include any definition for derived classes or derived interfaces, nor their corresponding metadata.

Besides, since several schemas may coexist in a database (the conceptual schema and some external schemas) and, given that inheritance relationships depend on the schema, these relationships should be stored in the repository in order to reuse them later in other schemas. However, current specifications of ODMG metadata only represent these relationships in a generic way, that is, without taking into account the schema where they occur.

In this paper, an extension of ODMG metadata is proposed in order to include the metaclasses corresponding to derived class and derived interface metadata, as well as an extension to model inheritance relationships by schema. In addition, to illustrate our proposal, our external schema definition mechanism is also briefly described.

The remainder of this paper is organized as follows. In Section 2, the main metadata included in ODMG specifications related to external schema definition are summarized. In addition, some considerations about current ODMG metadata are depicted. Section 3 summarizes briefly our proposal for defining external schemas in ODMG databases. In Section 4, modifications of current ODMG metadata adding new metadata to support the definition of external schemas in ODMG are presented, and an example to illustrate the proposed extension is also provided. Section 5 defines the proposed metadata in ODL. Finally, Section 6 summarizes the conclusions of this paper and discusses future work.

2 ODMG METADATA

In a database, metadata represent, among other information, descriptive information about the objects that make up the different schemas of the database. In ODMG, such information is stored in the Schema Repository [Catell00], which structure (metaschema) is also an object-oriented schema. In this section, the different kinds of objects stored in a database will be described. We will use this classification to describe the ODMG schema repository, and to propose its extension in order to define external schemas. Despite the proposed extension, we will show that our external schema definition mechanism does not extend the object-oriented paradigm.

Metaschema, schema, and data objects

A database can be said to store three kinds of objects depending on the level where they belong: metaschema objects, schema objects and data objects [Tresch92]:

- *Metaschema level objects*. These are objects that describe the database metaschema. Each object model has a set of metaobjects that allow the definition of database schemas. Such objects represent the different types and concepts of the object model, and are known as metaclasses.
- *Schema level objects*. These are objects that describe the schema of the application, and are the objects used to define the conceptual schema and external schemas. These objects are instances of metaclasses (metaschema level objects).
- *Data objects*. These are the objects that are really stored by the users in the database. They are instances of the classes defined at the schema level.

Objects are created top-down; initially, a database holds only metaschema level objects. When the database administrator defines the conceptual schema and external schemas, schema level objects are created. Finally, the user creates the data objects that are stored in the database.

ODMG metaschema

In ODMG, all objects of the repository are subclasses of three main interfaces: `MetaObject`, `Specifier` and `Operand`. In this paper, we only need to pay attention to `MetaObject` and, in particular, to metadata related to the support for defining external schemas. We will not study the metaclasses `Specifier` and `Operand`, because they represent concepts that do not affect the definition of external schemas. `Specifiers` are used to assign a name to a type in certain contexts. `Operands` form the base type for all constant values in the repository. Figure 1 illustrates the extract of the ODMG metaschema that represents the metadata we are considering in this paper. The illustration shows that metadata take part in a schema made up of metaclasses. The instances of such metaclasses are the different types, classes, relationships, and so on, which are used in the definition of schemas.

The main metadata of Figure 1 are briefly described below:

`MetaObject`. Metaobjects represent the elements of the schema that are stored in the repository. All the metaobjects have a name and take part in a relationship, named `definedIn`, with other metaobjects that establish their defining scope (e.g. the definition of a class in a module or the definition of an attribute in a class).

`Scope`. Schema objects are defined in a scope. Scopes define a name hierarchy for the metaobjects of the repository. An instance of `Scope` has a list of the instances of `MetaObject` (metaobjects) that are defined in its scope. Therefore, a module, which in ODMG defines a scope for defining classes, interfaces and exceptions, contains a list of the classes, interfaces and exceptions defined in that module.

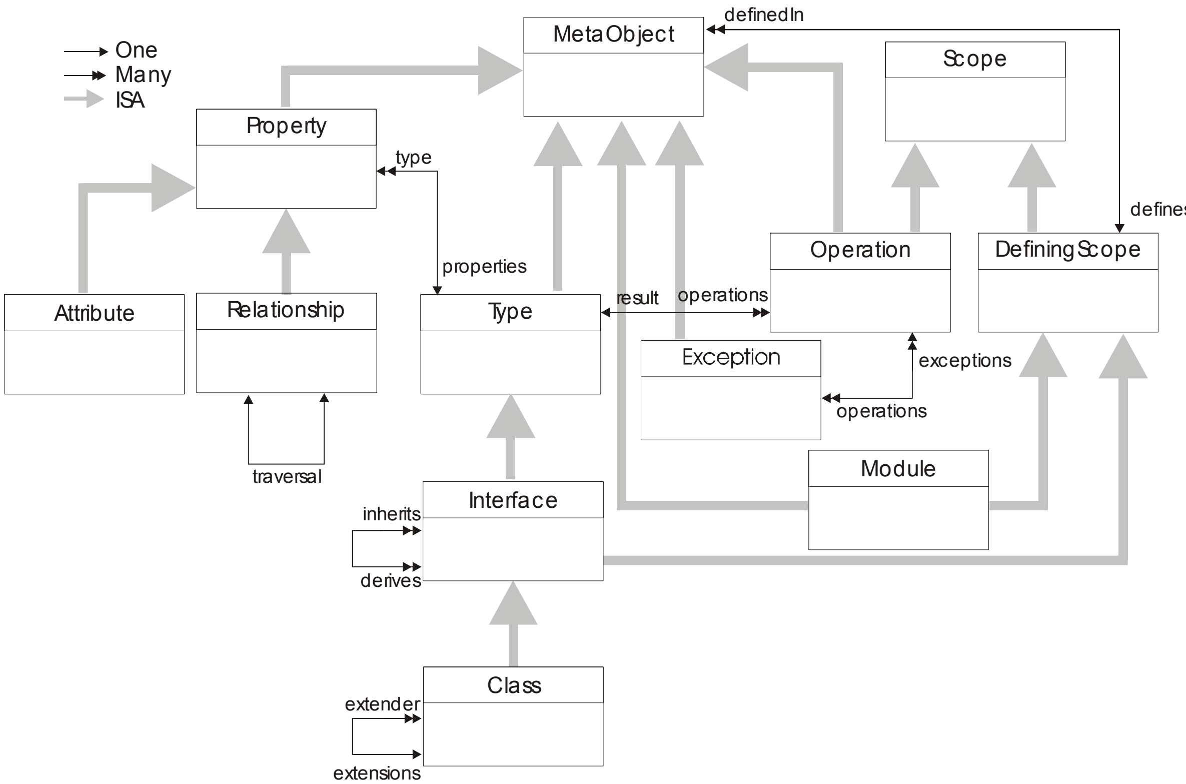

Fig. 1: An extract of the ODMG metaschema

`Module`. In ODMG, the modules and the schema repository itself, which is a specialized module, are defining scopes. These metaobjects include operations to add the modules, classes and interfaces that make up the modules. Modules are used to group certain instances of `MetaObject`, such as interfaces and classes. The instances of the metaclass `Module` are the topmost in the name hierarchy.

`Type`, `Interface` and `Class`. The instances of `Type` are used to describe types that are references to other objects. Instances of the metaclass `Interface` define the abstract behaviour of application objects. Interfaces are linked in a multiple inheritance graph with other interfaces by means of the `inherits` and `derives` relationships. The metaclass `Class` is a subtype of `Interface`. Its properties define the state of the objects stored in an OODB. Classes are linked in a single inheritance hierarchy by means of the `extender` and `extensions` relationships so that all the state and behaviour are inherited from the extended classes.

`Property`, `Attribute` and `Relationship`. The metaclass `Property` is an abstract class from which the metaclasses `Attribute` and `Relationship` are defined. Properties are defined in the scope of an interface or a class and describe the abstract state of an application object. The metaclass `Attribute` represents the properties that maintain the abstract state. The metaclass `Relationship` models bilateral associations between persistent objects.

Operation and Exception. The instances of the metaclass Operation model the abstract behaviour of application objects. Operations may raise exceptions (error conditions) that are stored in the metaclass Exception.

Issues about the current ODMG metaschema

The ODMG metaschema represents ODMG metadata as well as their relationships. However, in the current specifications of ODMG 3.0 the ODMG metaschema does not represent certain relationships that have to be defined in the repository in order to determine which classes or interfaces are included in a schema, or the relationship existing between two classes or interfaces of a given schema. In the current specifications of the standard, ODMG metadata only represent the schema where a class or an interface has been defined in. However, given that a class or interface may be used in another schema different from the one which it was initially defined in, a *usedIn* relationship must be represented in the repository. Moreover, the current specifications of ODMG consider the relationship existing between two classes (resp. interfaces) only in a generic way, without taking into account the schema where there exists in. Then, inheritance relationships between classes (resp. interfaces) must be stored in the repository to know the schema where they take place in.

As can be seen in Figure 1, all the metaclasses, except Scope and DefiningScope, are direct or indirect subclasses of MetaObject. Therefore, they inherit its state and its behaviour. As MetaObject is related to DefiningScope by means of the definedIn relationship, all the metaobjects of Figure 1, except Scope and DefiningScope have this property inherited from MetaObject. In other words, all the metaobjects, except Scope and DefiningScope, have a defining scope. This scope represents the object where an object has been defined in, which may be a class for a property, or a module for a class or a submodule. The definedIn relationship is 1:M. Therefore, instances of MetaObject are defined in a unique scope, and scopes can include several instances of MetaObject. This scope represents where the metaobjects were originally defined in, but it does not represent the scopes where they are used in. That is, with the current specification of ODMG metadata one can know only which module a class has been defined in, but not which module is used in. Likewise, this fact also happens with the properties and the operations of a class and other modules, classes and interfaces components.

Knowing which modules a metaobject is used in is important because an external schema definition mechanism would allow some kind of schema generation taking as input a selection of classes carried out by the schema definer. In [Torres01a], an external schema generation method for ODMG databases is proposed. This method uses the information of the repository to obtain the relationships existing between the classes of the schema. This information is used taking into account that if two classes are related by means of inheritance in one of the schemas stored in the repository, both classes will also be related by means of inheritance in that schema. Therefore, in order to reuse the information of the repository in the schemas, we have to store which inheritance relationships have a class with other classes in each schema where it is included in. Analogously, we can do the same for interfaces.

One might think that this "used in" relationship is the `definedIn` relationship itself. However, the relationship that we need is an M:N relationship, because a metaobject can be included in several scopes, and a scope can include several metaobjects (e.g. a class can be included in several schemas, and a schema can contain several classes). Therefore, the `definedIn` relationship existing in the current ODMG metadata specifications refers to the objects in which metaobjects are defined, but not to the different objects that use them.

Another failing to be found in the ODMG metaschema is that subtype relationships between interfaces, and subclass relationships between classes are established in a generic way, regardless of the schema. With the current specification of ODMG, all the instances of the metaclass `Interface` have a relationship defined, which indicates the superinterfaces or the subinterfaces of an interface. This relationship is an M:N relationship to support multiple inheritance between interfaces. From this relationship, we can identify the superinterfaces or subinterfaces of an interface, but not which schema this relationship exists in. This relationship cannot be obtained from the defining scope of the interface, because an interface is defined in a schema (module), which establishes its defining scope, but it does not determine all the schemas where the interface is included in. This problem also exists in the metaclass `Class`, which does not represent the subclass relationship existing between two classes in a given schema either. These situations are due to the fact that in the ODMG standard only a single schema is considered, but when several schemas coexist in a database, such as when external schemas are defined, the inheritance relationships depend on the schema.

Therefore, in order to know which instances of `MetaObject` have been included in a given scope as well as, which instances of `Interface` and `Class` with their respective relationships have been included in a `Module`, several modifications have to be carried out in the ODMG metaschema. This modification involves the revision of the metaclasses `MetaObject` and `DefiningScope` for the former, and the revision of the metaclasses `Module`, `Interface` and `Class` for the latter.

3 A MECHANISM FOR DEFINING EXTERNAL SCHEMAS IN ODMG DATABASES

The mechanism we are developing is based on the ODMG 3.0 object model, and it explicitly considers the ODMG schema repository for defining derived classes and external schemas, making it easier the reuse of previous definitions. Derived classes are defined using the mechanisms proposed in [Roantree99, Garcia02] because ODMG does not address the definition of derived classes. (In fact, ODMG proposes the use of *named queries*, which do not offer the expected functionalities). In our mechanism, derived classes and derived interfaces are integrated within the repository by means of the *derivation* relationship [Bertino92] as described in [Samos95]. This relationship is only used in the repository to relate derived classes to their base classes (resp. interfaces), making easier the integration, and avoiding the generation of intermediate unnecessary classes as in [Scholl91, Rundensteiner92]. However, end-user schemas do not use this kind of relationship in order to preserve the object-oriented paradigm. Therefore, existing inheritance relationships between classes and interfaces of the schema must be obtained when derived classes and derived interfaces are included in external schemas. For such a purpose an external schema generation algorithm [Torres01a] is used, which obtains the existing relationships between derived classes and the remainder set of classes of the external schema. External schema specification is carried out by means of the language proposed in [Torres00]. The language specifies the classes and interfaces to be included in the external schema. From this specification, we obtain a set of isolated classes and interfaces. Then, a closure process is applied to this set so that no references to classes or interfaces not included in the schema exist in the schema [Torres01b].

Figure 2 illustrates a repository for an ODMG database of people. People may be clients or employees, and both groups have vehicles. Information about temporaries is also stored. Temporaries and employees have several common properties and behaviour, but different from the issues common to employees and clients, that is modelled with the interface `Worker`. For the sake of simplicity, attributes and operations are not depicted in the figure. Figure 2 also illustrates an external schema (the surrounded area) which replaces the class `Employee` with a derived class `Employee'`. `Employee'` hides some instances and properties of `Employee`. The figure shows that the process for integrating derived classes is easy, and is related to connect derived classes with their base classes by means of derivation relationships. Figure 2 also shows how the derivation relationship is used in the repository but not in the external schema.

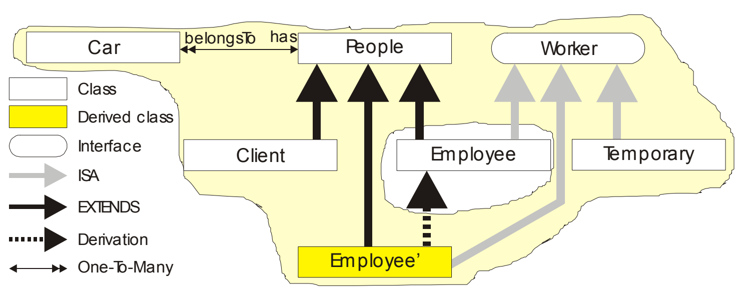

Fig. 2: Repository and external schema

Therefore, in our mechanism, unlike in [Bertino92, Kim95, Santos95, Guerrini97, Garcia02], external schemas only use relationships that are allowed in ODMG. So that, the object-oriented paradigm does not have to be extended, and unnecessary intermediate classes have not to be generated. However, metadata for derived classes and derived interfaces must be defined to define external schemas in ODMG databases.

4 EXTENSION OF THE ODMG METASCHEMA

The development of a mechanism for defining external schemas in ODMG involves the existence of mechanisms for defining derived classes and derived interfaces to customize existing classes and interfaces, respectively. However, the study of such mechanisms is beyond of the scope of this paper. Such details can be found in [Roantree99, Garcia02]. In this paper, only the issues concerning to ODMG metadata to give support for defining external schemas are studied, introducing the concepts of external schema, derived class and derived interface in the metaschema, as well as the relationships that represent the objects included in the schemas (classes, interfaces, and so on). In this section, the proposed metadata for derived classes and derived interfaces, as well as the metadata for obtaining the components of a schema are described.

Metadata for derived classes and derived interfaces

Current specifications of ODMG do not include a metaclass for storing the defined derived classes does not exist either. Before describing the definition of a metaclass for derived classes, a note is added to clarify the modifications proposed to ODMG metadata: an ODMG external schema consists of interfaces, classes, and exceptions. Interfaces and classes may be derived or not. Since in our external schema definition methodology, derived classes are fully-featured, without loss of generality one can say that an ODMG module consists of classes and interfaces, which can be derived or non-derived. This is an abstraction of the class and interface concepts that we term *generic class* and *generic interface*, which are specialized in non-derived class and derived class, and in non-derived interface and derived interface, respectively. Figures 3.a and 3.b illustrate this abstraction and depict the derivation relationship. With regard to classes, the derivation relationship represents the relationship existing between derived classes and their base classes, which can be derived or not, that is, generic classes. So, in this way is represented the fact that a derived class may be defined from existing classes, and the fact that an existing class may be a base class of some derived classes. Analogously, we can follow the same approach for derived interfaces, as Figure 3.b illustrates.

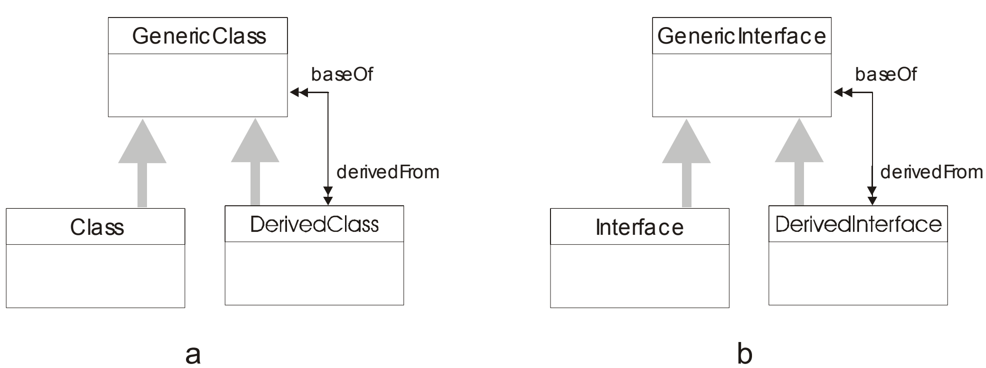

Fig. 3: a) Metadata for derived classes; b) Metadata for derived interfaces

As figures 3.a and 3.b show, the modification to the metaschema proposed to include metaclasses for derived classes and derived interfaces, the new abstractions for generic classes and generic interfaces, as well as the derivation relationship, retains the structure of the schema repository as an object-oriented schema. Therefore, the use of the derivation relationship in the repository to relate derived classes to their base classes, and derived interfaces to their base interfaces, does not involve an extension of the object-oriented paradigm. The derivation relationship is modelled in the repository by means of an ODMG compliant relationship.

Moreover, although in this paper we are focusing on our mechanism, the proposed extensions of ODMG metadata can also be used in most of the existing mechanisms because the abstraction defined in this paper to generalize derived and non-derived classes (i.e. generic classes), and to generalize derived and non-derived interfaces (i.e. generic interfaces) can be also carried out in other methodologies. This relationship between derived classes and their base classes, and between derived interfaces and their base interfaces is also used implicitly in other mechanisms, because derived classes and derived interfaces are always defined from other existing classes or interfaces (their base classes or interfaces), and this relationship exists regardless of the methodology used. The derivation relationship only describes how a derived class (resp. derived interface) is defined from other existing classes (resp. interfaces), derived or non-derived, that is, generic classes (resp. generic interfaces).

Therefore, the benefit is twofold. On the one hand, the ODMG standard is extended to allow the definition of external schemas. On the other hand, existing mechanisms may use the proposed extension in order to be defined on an object standard extended to allow the definition of external schemas.

Metadata for the components of a schema

External schemas are fully featured schemas and then, they can be considered as instances of the metaclass `Module`, that is, the different modules (groupings of classes, interfaces and exceptions) defined. Therefore, unlike derived classes and interfaces, it is not necessary to define an additional metaclass to model the concept of external schema. However, it is necessary to modify the current definition of the metaclass `Module` so that all the classes and interfaces included in a module, as well as their respective inheritance relationships in the module, are also known. With the current specifications of ODMG metadata, one can only know which module a class or interface has been defined in, but not which module are used in. Conceptually, these are two ternary relationships, one for interfaces, and one for classes. With regard to the ternary relationship concerning interfaces, the metaclasses involved are `Interface`, acting as superinterface, `Interface`, acting as subinterface, and `Module`, representing where the inheritance relationship takes place. Thus, the inheritance relationship between two interfaces in a schema can be determined. Similarly, this relationship has also to be established between classes and modules in order to identify the inheritance relationship between two classes in a module, giving rise to the other ternary relationship. To illustrate the need for this relationship, Figure 4.a illustrates a schema made up of four classes `A`, `B`, `C` y `D`. From this schema, two external schemas, illustrated in figures 4.b and 4.c, are defined. In Figure 4.a, `D EXTENDS C`, `C EXTENDS B`, and `B EXTENDS A`. However, in Figure 4.b, `D EXTENDS B`, and `B EXTENDS A`, and in Figure 4.c, `D EXTENDS A`. Then, inheritance relationships between classes (resp. interfaces) are not absolute, as can be followed from the current ODMG specifications for metadata, but relative to modules.

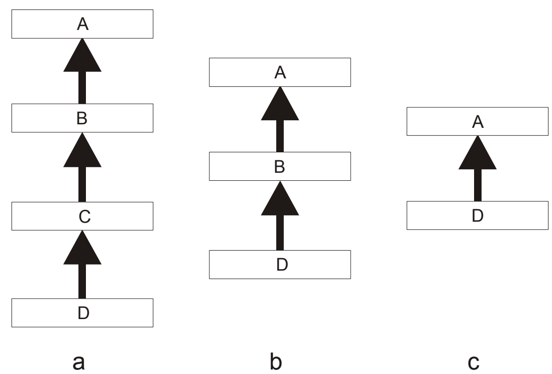

Fig. 4: Inheritance relationships depend on schemas

However, since ODMG only allows binary relationships, the two previous ternary relationships have to be transformed. In [Catell94, Blaha97], a transformation that can be generalized to n-ary relationships is proposed. The transformation creates a new class for the ternary relationship and establishes binary relationships between the new class and the three classes that take part in the ternary relationship.

Figure 5 illustrates the proposed modifications to the metaclasses of Figure 5 to model the interfaces and classes of a module in the schema repository, as well as the inheritance relationships by module. For the sake of simplicity, the modifications proposed in the previous section concerning derived classes and derived interfaces are not depicted in the illustration. So that, in the illustration, instead of showing the metaclasses `GenericClass` and `GenericInterface` as well as their corresponding updates, the ODMG metaclasses `Class` and `Interface` are showed. In order to model the *usedIn* relationship, the proposal also includes a new relationship between `MetaObject` and `DefiningScope`. With this new relationship, we can identify the metaobjects included or used in a given metaobject, as well as the objects which a certain metaobject is used in, because with the current `definedIn` relationship we can only know where it was defined in but not where it is used in, as described in Section 2.3. Likewise, Figure 5 also illustrates how the ternary relationship between `Interface` (which had a reflexive relationship in the original metaschema) and `Module` has been transformed, generating a new metaclass `ModuleInterfaces`. The instances of this new metaclass are each of the relationships existing between the interfaces of a module. This new metaclass has as properties a module, an interface (acting as subinterface), and another interface (acting as superinterface). In this way, the subinterfaces and superinterfaces of an interface in each schema can be represented. These three properties, which are relationships, comprise a key to express the semantics of the ternary relationship (in a module, an interface may have several superinterfaces).

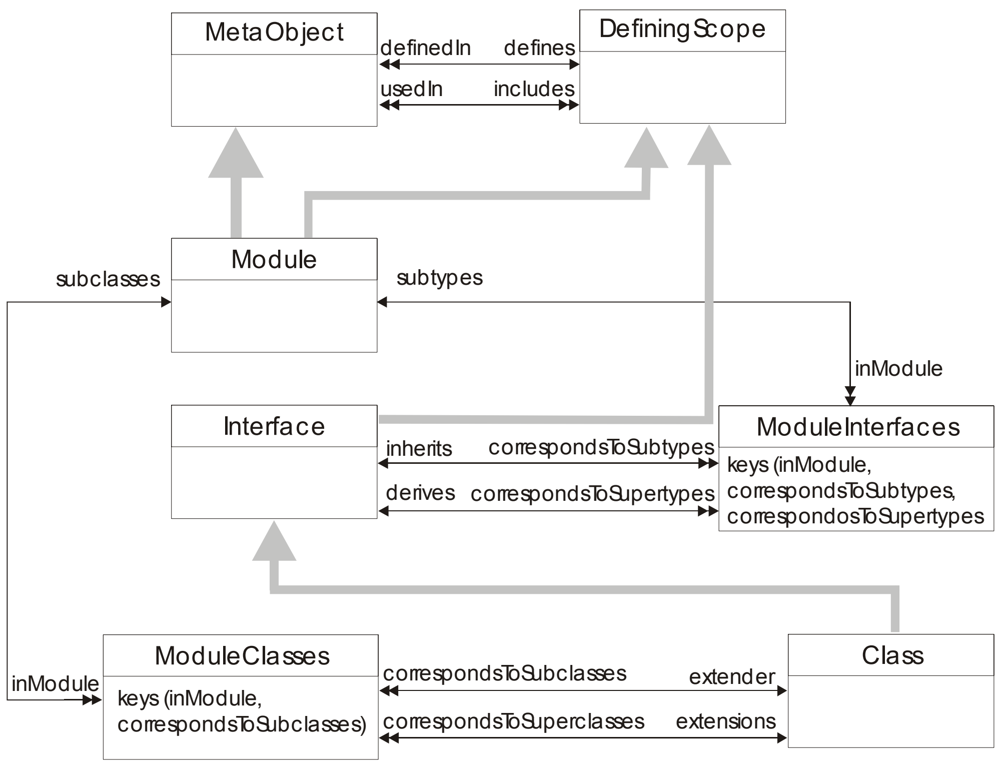

Fig. 5: Modelling the components of a schema as well as their relationships by schema

A module may have several relationships with several instances of this metaclass, which is expressed by means of the relationship `subtypes-inModule` existing between `Module` and `ModuleInterfaces`. In addition, `ModuleInterfaces` has two relationships with `Interface` to express the interface which the instances of such metaclasses are referred to, and to represent the ternary relationship. These are the two relationships existing between `ModuleInterfaces` and `Interface`.

Analogously, it can be seen that another metaclass has been created, named `ModuleClasses`, to express the ternary relationship existing between classes, subclasses and modules. Nevertheless, because of classes can only have single inheritance relationships, in each module a class has only a superclass, and then, the key is comprised of the class acting as subclass and the module where the relationship exists in.

A metaclass for inheritance and derivation relationships

If we analyse how inheritance relationships are represented in the metaschema we conclude that it does not exist any metaclass for these relationships as in other models

[Tresch92, Saltor95]. However, in ODMG there exists a metaclass for relationships (i.e. `Relationship`). This disagreement is because in ODMG, the metaclass `Relationship` is considered as a subtype of `MetaObject`, and therefore inherits a name and a defining scope (the name of the relationship, and the interface or class where it is defined in). However, these characteristics are not applied to inheritance relationships (an inheritance relationship does not have a name), and hence ODMG decides not to create a metaclass for such relationships. In ODMG, inheritance relationships are represented by means of the `extender` and `extensions` relationships for classes, and by means of the `inherits` and `derives` for interfaces. These relationships are defined in the metaclasses `Class` and `Interface`, respectively.

If we were interested in creating a metaclass for inheritance relationships and include it in the metaschema, so that inheritance relationships of the schemas are instances of this metaclass, as in [Tresch92, Saltor95], that metaclass would not be a subclass of `MetaObject`, because it would inherit the `name` attribute, which does not make sense in inheritance relationships. However, in the ODMG C++ binding there exists a metaclass `d_Inheritance`. This mismatch is due to the fact that, unlike to the ODMG metaclass `Class`, the metaclass `d_Class` of the C++ binding does not include a relationship to model the subclass relationships between classes. Therefore, in order to model these relationships in the binding, a metaclass `d_Inheritance` is defined, which instances are the inheritance relationships existing between instances of `d_Class`. Nevertheless, as stated above, in the C++ binding `d_Inheritance` is not considered as a subclass of `d_MetaObject` (the equivalent to `MetaObject` in the C++ binding) because it has neither a name nor the `definedIn` relationship. So, in the C++ binding `d_Inheritance` is another metaclass but it is not defined from `d_MetaObject`.

However, since the bindings are not the scope of this paper, we follow the approach used in ODMG metadata. Therefore, given that ODMG metadata does not include a metaclass for the inheritance relationship, we follow the same approach for the derivation relationship, because although it is a relationship with a different semantic, it is also a relationship between classes that is not identified with a name either. Therefore, instead of creating a new metaclass for derivation relationships, we have decided to create a relationship between derived classes and their base classes following a similar approach to the one used for inheritance relationships.

In the extension proposed in this paper to model the components of a schema, two metaclasses, `ModuleInterfaces` and `ModuleClasses`, have been created for such a purpose. However, this fact should not be interpreted as a mismatch between the consideration carried out for the derivation relationship. Such classes have been created as a result of the transformation of ternary relationships, because the ODMG object model only allows binary relationships. Nevertheless, as in C++ binding, those metaclasses are not defined from `MetaObject`, because the relationships, which their instances represent, do not have a name.

An example

Figure 6 illustrates the external schema defined in Section 3, as well as the proposed extension for the ODMG metaschema. In the figure, the proposed relationship between `MetaObject` and `DefiningScope` to model the *usedIn* relationship is illustrated, as well as the proposed metaclasses for derived classes and derived interfaces. Likewise, needed classes to represent inheritance relationships by module have been also depicted in the illustration. However, for the sake of simplicity, attributes and operations are not depicted in the figure.

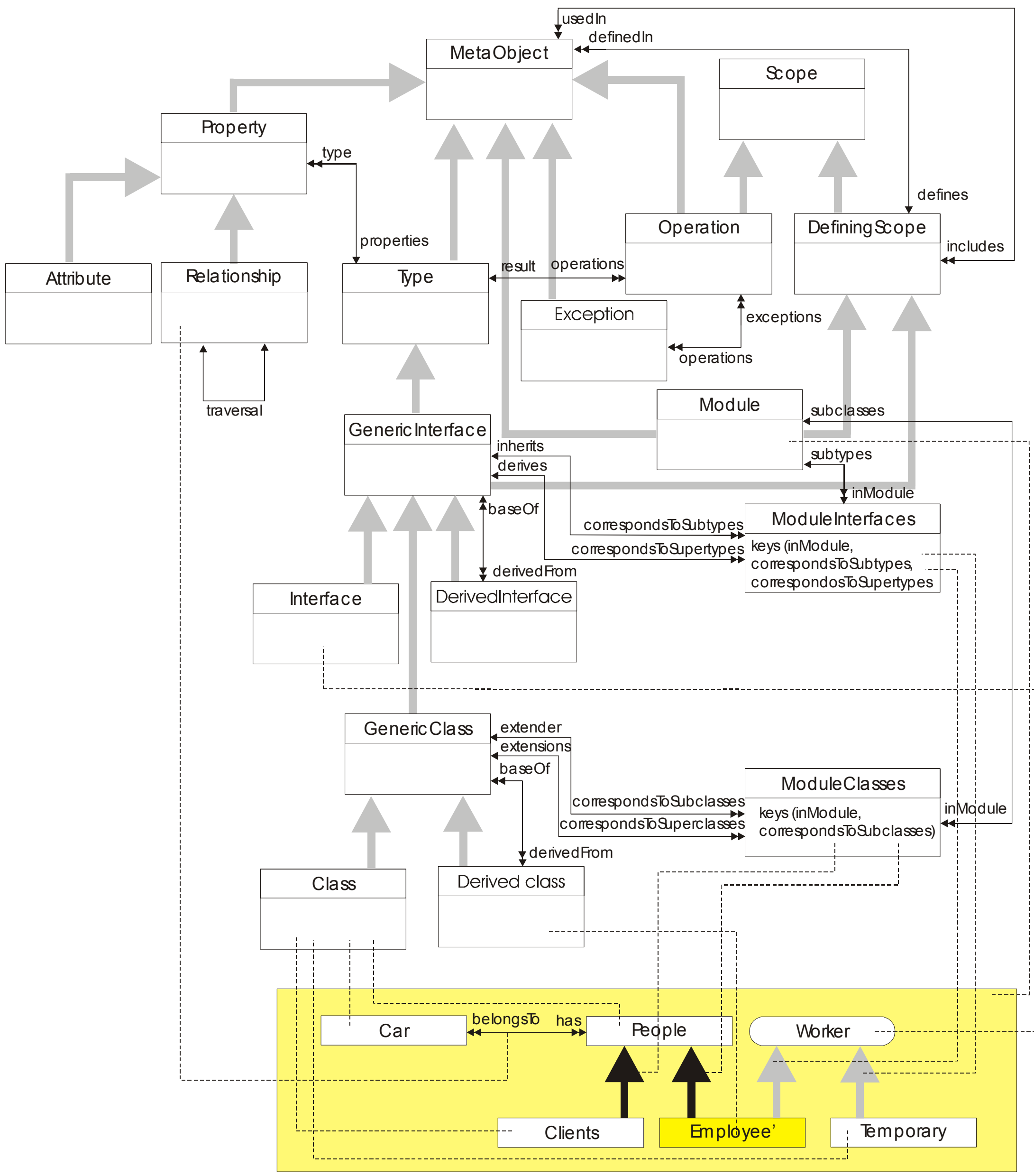

Fig. 6: Schema repository and an external schema definition

Attributes of classes and interfaces are instances of the metaclass `Attribute`, while operations of classes and interfaces are instances of the metaclass `Operation`. The conceptual schema and the definition of the derived classes and interfaces have not been depicted either. However, we can imagine the schema repository as a set of layers, which represent the schemas defined and stored in the repository. In addition, as Figure 6 shows, each schema component (classes, interfaces, relationships, and so on) is related to its metaclass by means of an `InstanceOf` relationship, which is depicted as a dashed line.

5 ODL DEFINITION OF PROPOSED METADATA

ODMG metadata are described in ODL, the object definition language proposed by ODMG. ODL is a definition language for specifying objects, and in ODMG databases, schemas are defined in ODL.

Once described the proposed metaclasses to define external schemas, the corresponding ODL definitions are presented in this section, distinguishing the new definitions from the modified ones. The modifications are depicted in italics, and only attributes and relationships are specified, leaving out operations and exceptions.

Metadata definition is carried out in ODMG 3.0 style; therefore, metaclasses are then defined by means of `interface` definitions. However, the metaclasses `ModuleInterfaces` and `ModuleClasses` are defined as classes because they include a key definition, and in ODMG keys are defined on classes but not on interfaces. Hence, these metaclasses are defined using a `class` specification instead of an `interface` one.

Metaobjects and defining scopes

The metaclasses `MetaObject` and `DefiningScope` have been modified in order to include the *usedIn* relationship of metaobjects in a given scope so that, one can know which classes, interfaces and exceptions have been used in a module definition.

```
interface MetaObject: RepositoryObject {
   attribute string name;
   attribute string comment;
   relationship DefiningScope definedIn
      inverse DefiningScope::defines;
   relationship set<DefiningScope> usedIn
      inverse DefiningScope::includes;
   ...
} ;
```

Interfaces

In ODMG 3.0, the metaclass `Interface` is defined from the metaclass `Type`. However, the modification proposed in this paper includes an abstraction of the interface concept motivated by the introduction of the derived interface concept. Next, the current definition existing in ODMG is shown.

```
interface Interface: Type, DefiningScope {
   struct ParameterSpec {
      string param_name;
      Direction param_mode;
      Type param_type;};
   relationship set<Interface> inherits
      inverse Interface::derives;
   relationship set<Interface> derives
      inverse Interface::derives;
};
```

The abstraction proposed for the interface concept, `GenericInterface`, plays the role of the current metaclass `Interface`, although it modifies its inheritance relationships in order to represent the inheritance relationships by module. However, the definitions proposed in this section for interfaces are open to the inclusion of properties and operations, because its definition must be guided by the derived interface definition mechanism. Therefore, in this paper only a draft is proposed as a basis for the metadata to be included by the definitive derived interface definition mechanism. Next, the ODL definitions of `GenericInterface`, `Interface` and `DerivedInterface` are proposed.

```
interface GenericInterface: Type, DefiningScope {
   struct ParameterSpec {
      string param_name;
      Direction param_mode;
      Type param_type;};
   relationship set<ModuleInterfaces> inherits
      inverse ModuleInterfaces::correspondsToSubtypes;
   relationship set<ModuleInterfaces> derives
      inverse ModuleInterfaces::correspondsToSupertypes;
   relationship set<DerivedInterface> baseOf
      inverse DerivedInterface::derivedFrom;
   ...
};

interface Interface: GenericInterface {
   ...
};

interface DerivedInterface: GenericInterface {
   relationship set<GenericInterface> derivedFrom
      inverse GenericInterface::baseOf;
   ...
};
```

Classes

As well as the metadata related to interfaces, metadata related to classes also include an abstraction that allows the class concept to be generalised to represent the concepts of derived and non-derived classes. As well as in derived interfaces, the definition proposed is open until a consensus is reached on the definition of derived classes in ODMG. The current metadata definition for classes is depicted as they appear in ODMG specifications.

```
interface Class:Interface {
   attribute list<string> extents;
   attribute list<string> keys;
   relationship Class extender
      inverse Class::extensions;
   relationship set<Class> extensions
      inverse Class::extender;
};
```

The abstraction proposed in this paper, named `GenericClass`, has a definition inspired in the current definition of `Class`, as the next definition shows. Moreover, the proposed definition for metaclasses `Class` and `DerivedClass` is also depicted, but must be completed when derived class definition mechanisms for ODMG are fully developed.

```
interface GenericClass: GenericInterface {
   attribute list<string> extents;
   attribute list<string> keys;
   relationship set<ModuleClasses> extender
      inverse ModuleClasses::correspondsToSubclasses;
   relationship set<ModuleClasses> extensions
      inverse ModuleClasses::correspondsToSuperclasses;
   relationship set<DerivedClass> baseOf
      inverse DerivedClass::derivedFrom;
   ...
};


interface Class: GenericClass {
   ...
};

interface DerivedClass: GenericClass {
   relationship set<GenericClass> derivedFrom
      inverse GenericClass::baseOf;
   ...
};
```

Components of a module

Finally, the ODL definitions of the metaclasses `ModuleInterfaces` and `ModuleClasses` are proposed, as well as the modification to the definition of the metaclass `Module`. Unlike the previous definitions, `ModuleInterfaces` and `ModuleClasses` are specified by means of `class` instead of being defined as an `interface`. This is due to the fact that in ODMG, only types with extension (i.e. classes) may have keys in their definition.

```
interface Module: MetaObject, DefiningScope {
   relationship set<ModuleInterfaces> subtypes
      inverse ModuleInterfaces::inModule;
   relationship set<ModuleClasses> subclasses
      inverse ModuleClasses::inModule;
   ...
};

class ModuleInterfaces
(  extent TheModuleInterfaces) {
   relationship GenericInterface correspondsToSubtypes
      inverse GenericInterface::inherits;
   relationship GenericInterface correspondsToSupertypes
      inverse GenericInterface::derives;
   relationship Module inModule
      inverse Module::subtypes;
   keys inModule, correspondsToSubtypes, correspondsToSupertypes
   ...
};

class ModuleClasses
(  extent TheModuleClasses) {
   relationship GenericClass correspondsToSubclasses
      inverse GenericClass::extender;
   relationship GenericClass correspondsToSuperclasses
      inverse GenericClass::extensions;
   relationship Module inModule
      inverse Module::subclasses;
   keys inModule, correspondsToSubclasses
   ...
};
```

6 CONCLUSIONS AND FUTURE WORK

In this paper, an extension of ODMG metadata to support the definition of external schemas has been proposed. This extension is necessary because an external schema definition mechanism for ODMG involves the existence of facilities to define derived classes and derived interfaces that allow customizing existing classes and interfaces, respectively, as well as the existence of the underlying metadata. This paper is only focused on the extension of ODMG metadata to give support to the ODMG standard to define external schemas, but it is not focused on the mechanisms for defining derived classes and derived interfaces themselves.

The extension proposed to the ODMG metadata in this paper establishes the necessary metadata to define derived classes and derived interfaces. Moreover, since current ODMG specifications are focused on a unique schema, the extension needed for metadata to represent the subclass and subtype relationships depending on the schema has been also introduced.

The proposed extension of ODMG metadata establishes the necessary support to represent in the metaschema the corresponding metadata for derived classes, derived interfaces and external schemas, which are common issues that can be applied to most of the existing mechanisms based on ODMG. Therefore, this extension can be considered as a generic proposal that can be applied to the external schema definition mechanism we are developing and to other mechanisms based on the ODMG standard, although other methods for integrating derived classes and interfaces are used.

Our mechanism is characterized by the use of the derivation relationship in the repository to relate derived classes and derived interfaces to their base classes and interfaces, respectively. The use of the derivation relationship in the repository preserves its structure as an object-oriented schema because it is modelled as an ODMG `relationship`, so the repository schema is still an object-oriented schema. In addition, this relationship is not used in the user schemas preserving the object-oriented paradigm in user schemas.

In this paper, we have used existing mechanisms to define derived classes. However, such mechanisms are not fully-featured because they allow only object-preserving semantics and cannot define capacity-augmenting derived classes. Therefore, we are working in the definition of such mechanisms and its utilization in schema evolution. This requirement is necessary because if ODMG wants to promote its specifications to an accepted standard, the definition of external schemas must be allowed, and the existence of fully-featured mechanisms to define derived classes and derived interfaces is the basis for such a purpose.

Currently, we are also working on the extension of the C++ binding according to the model proposed in this paper. Similarly, other extensions for the remainder ODMG bindings have to be defined.

ACKNOWLEDGEMENTS

This work has been supported by the Spanish CICYT (project TIC 2000-1723-C02-02).

REFERENCES

[Abite91] Abiteboul, S., Bonner, A. 'Object and Views'. *In Proc. ACM SIGMOD International Conference on Management of Data*. pp. 238-247. 1991

[Berti92] E. Bertino. "A View Mechanism for Object-Oriented Databases" *In Proc. of the 3rd EDBT*. pp. 136-151. 1992

[Blaha97] M. Blaha, W. Premerlani. Object-oriented Modeling and Design for Database Applications. Prentice Hall. 1997.

[Catell94] R.G.G. Catell. Object Data Management. Addison Wesley. 1994.

[Catell00] R.G.G. Catell. The Object Database Standard: ODMG 3.0. Morgan Kaufmann. 2000

[Dobro93] M. Dobrovnik, J. Eder. "Adding View Support to ODMG-93" *In Proc. of the 1st Internat. Workshop on Advances in Databases and Information Systems*. pp. 62-73. 1994

[Garcia02] J.García Molina, M.J. Ortín Ibáñez, G. García Mateos. "Extending the ODMG standard with views" *In Information and Softare Technology*. Vol 44. pp. 161-173. 2002

[Guerr97] G. Guerrini, E. Bertino, B. Catania, J. Garcia-Molina. "A Formal View of Object-Oriented Database Systems" *TAPOS*. Vol. 3(3). pp. 157-183. 1997

[Kim95] W. Kim, W. Kelley, "On View Support in Object Oriented Database Systems". *In Modern Database Systems*. pp. 108-129. 1995

[Motsch96] R. Motschnig-Pitrik, "Requirements and Comparison of View Mechanisms for Object-Oriented Databases" *In Information Systems*, Vol. 21(3). pp. 229-252. 1996

[Roant99] M. Roantree, J.B. Kennedy, P.J. Barclay. "Providing Views and Closure for the Object Data Management Group Object Model" *In Information and Software Technology*. Vol. 41. pp. 1037-1044. 1999

[Runde92] E.A. Rundensteiner. "Multiview: A Methodology for Supporting Multiple Views in Object-Oriented Databases" *In Proc. of the 18th VLDB*. pp. 187-198. 1992

[Saltor95] F. Saltor, M. Castellanos, M. Garcia-Solaco, Th. Kudrass "Modelling Specialization as BLOOM Semilattices" *In 4th European-Japanese Seminar on Inf. Modelling and Knowledge Bases*. pp. 447-467. 1995.

[Samos95] J. Samos. "Definition of External Schemas in Object Oriented Databases" *In Proc. of OOIS 1995*. pp. 154-166. 1995

[Santos95] C.S. Santos, "Design and Implementation of Object-Oriented Views", *In Proc. of 6th DEXA*. pp. 91-102. 1995

[Scholl91] M.H. Scholl, C. Laasch, M. Tresch. "Updatable Views in Object-Oriented Databases", *In Proc. of the 2nd DOOD*. pp. 189-207. 1991

[Torres00] M. Torres, J. Samos "Definition of External Schemas in ODMG Databases" *Proc. of OOIS 2000*. pp. 3-14. 2000.

[Torres01] M. Torres, J. Samos "Generation of External Schemas in ODMG Databases" *In Proc. of IDEAS 2001*. pp. 89-98. 2001.

[Torres01] M. Torres, J. Samos "Closed Schemas in Object-Oriented Databases" *In Proc. of DEXA 2001*. pp. 826-835. 2001.

[Tresch92] M. Tresch, M.H. Scholl "Meta Object Management and its Application to Database Evolution" *In Proc. 11th ER*. pp. 299-321. 1992.

About the authors

Manuel Torres is associate lecturer at the Departamento de Lenguajes y Computación of the University of Almeria. He recently got his Ph.D about the definition of external schemas in ODMG databases. His interests include object-oriented databases, especially view mechanisms and their application to schema evolution. He can be reached at mtorres@ual.es.

José Samos is assistant teacher at the Departamento de Lenguajes y Sistemas Informáticos of the University of Granada. He heads a research group about object-oriented databases and data warehousing. He can be reached at jsamos@ugr.es.