JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2004**

Vol. 3, No. 10, November-December 2004

A Language to Define External Schemas in ODMG Databases

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

Abstract

ODMG is the de facto standard for object-oriented databases (OODB) but does not address the definition of external schemas. In order to provide such functionality, we have developed an external schema definition methodology that allows the definition of external schemas in ODMG databases. In this work, the language used by the schema definer to specify the schema components (e.g. classes, interfaces, and so on) is proposed. Besides, we describe the effect caused on the ODMG schema repository by the definition of external schemas using the proposed language.

1 INTRODUCTION

ODMG standard [Catell00] addresses many OODBs issues. However, a well-known functionality of databases as is the definition of *external schemas* is not addressed by the standard. External schemas match with the higher level of the ANSI/SPARC architecture, providing different views of the conceptual schema for particular users or group of users.

In order to provide ODMG with that functionality, we have developed an external schema definition methodology for ODMG databases [Torres02]. This methodology allows database administrators to define external schemas in a three-step process. First, schema definers specify the classes to be included in external schemas; second, schemas are closed so that, classes do not have references to classes that are not included in the schema; third, schemas are generated obtaining inheritance relationships instead of specifying them manually, avoiding the introduction of mistakes. Schema closure and schema generation are studied in [Torres01b, Torres01a] respectively.

The methodology also introduces an extension of ODMG metadata [Torres03] to give support for defining external schemas in ODMG introducing metadata for *derived classes* (classes defined from existing ones customizing them, like relational views for tables in relational databases), external schemas, and so on. Therefore, a mechanism

Cite this article as follows: Manuel Torres, José Santos: "A Language to Define External Schemas in ODMG Databases", in *Journal of Object Technology*, vol. 3, no. 10, November-December 2004, pp. 181-192. http://www.jot.fm/issues/issue_2004_11/article5

analogous to relational views must exist to define derived classes in OODBs. Many proposals have been defined for such a purpose: *virtual classes* [Abiteboul91, Rundensteiner92, Santos95], *view classes* [Bertino92, Guerrini97], and so on. However, to the best of our knowledge, only two proposals [Garcia02, Roantree99] have been defined in the ODMG framework because ODMG only has proposed the definition of named queries for that purpose. In our methodology, derived classes are defined using the mechanisms proposed in [Garcia02, Roantree99].

In this paper, a language to define external schemas in ODMG databases is proposed. The language allows the specification of external schemas selecting the components that make up an external schema. However, given that this specification process interacts closely with the repository, a brief overview of the extension to ODMG metadata introduced in [Torres03] is also described in this paper.

The remainder of this paper is structured as follows. Section 2 introduces our methodology for defining external schemas in ODMG. This section also summarizes the ODMG object model as well as an extension of ODMG metadata we have proposed to define external schemas in ODMG. In Section 3 the external schema definition language is proposed and for each operation, the effect on the extended ODMG repository is described. Finally, Section 4 summarizes the conclusions and discusses the future work.

2 OUR EXTERNAL SCHEMA DEFINITON METHODOLOGY AT A GLANCE

This section is divided in three subsections. First, the ODMG object model is summarized in order to introduce the concepts we use in our methodology. Then, the external schema definition methodology we have developed for ODMG databases is briefly introduced. Finally, an overview about the extension of ODMG metadata we have proposed to support the definition of external schemas in ODMG is put forward.

ODMG object model

In the ODMG standard, the basic modelling properties are the *object* and the *literal*. Objects and literals can be categorized by their *types*. All elements of a type have a common range of states (the same set of properties) and a common behaviour (the same set of operations). The state of an object is defined by the values it carries for its set of *properties*. These properties can be *attributes* (e.g. name, idEmployee) or *relationships* with other objects (e.g. relationship between teachers and students). In ODMG, there are several ways for specifying types: an `interface` defines only the abstract behaviour of an object type; a `literal` defines only the abstract state of a literal type; a `class` defines the abstract behaviour and the abstract state of an object type. Because of the existence of two different object types (i.e. classes and interfaces) two sorts of inheritance relationships may be defined in ODMG, known as `ISA` and `EXTENDS`. On the one hand, `ISA` is a multiple inheritance relationship for behaviour inheritance between object types.

On the other hand, `EXTENDS` is a simple inheritance relationship for state and behaviour inheritance relationships between object types.

Overview of our external schema definition methodology for ODMG databases

The methodology we have developed to define external schemas is based on the ODMG 3.0 object model [Catell00], 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 external schema definition language proposed in this paper. 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 1 illustrates a repository for an ODMG database of people. People may be clients or employees, and both groups have vehicles. In addition the adresses of people are stored as objects. 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 1 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 1 also shows how the derivation relationship is used in the repository but not in the external schema.

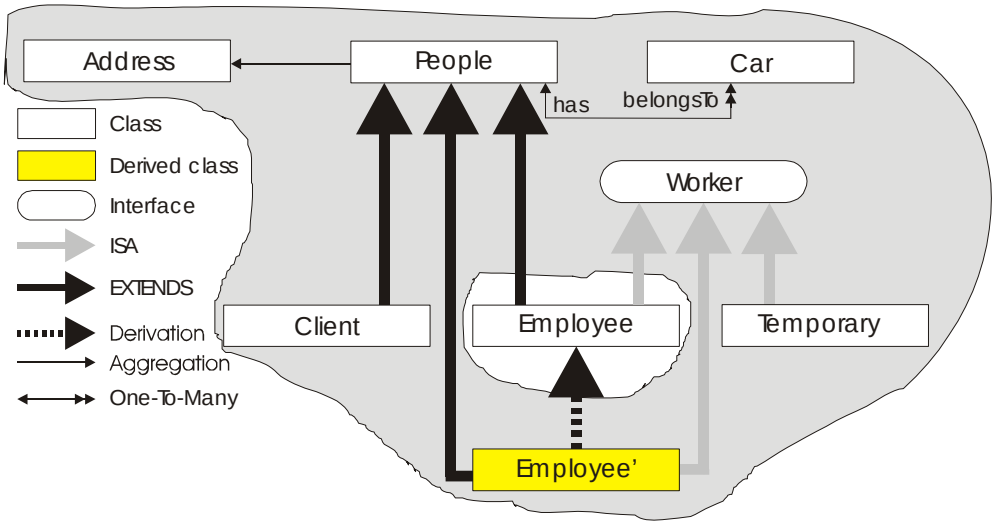


Fig. 1: Repository and external schema

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.

An extension of ODMG metadata to support the definition of external schemas

ODMG defines a *Schema repository* (metaschema) to store descriptive information about the objects that make up the different schemas of the database, that is, *metadata*. Figure 2 illustrates an extract of the ODMG metaschema. This extract depicts metaclasses involved in external schema definition. Instances of those metaclasses are the types, classes, relationships, and so on, used in each schema definition. Main metaclasses of Figure 2 are `MetaObject`, `DefiningScope`, `Module`, `Interface`, `Class`, `Attribute`, `Relationship`, `Operation` and `Exception`. `MetaObject` represents the elements of the schema that are stored in the repository. `DefiningScope` stores instances that represent definition scopes of other metaobjects. `Module` includes an instance for each defined module. `Interface` and `Class` contain all the class and interface definitions, respectively. Analogously, `Attribute` and `Relationship` store the definition of each attribute and relationship specified in a class or interface. Finally, in `Operation` defined operations are stored, and `Exception` includes instances of all the exceptions that can be raised by operations.

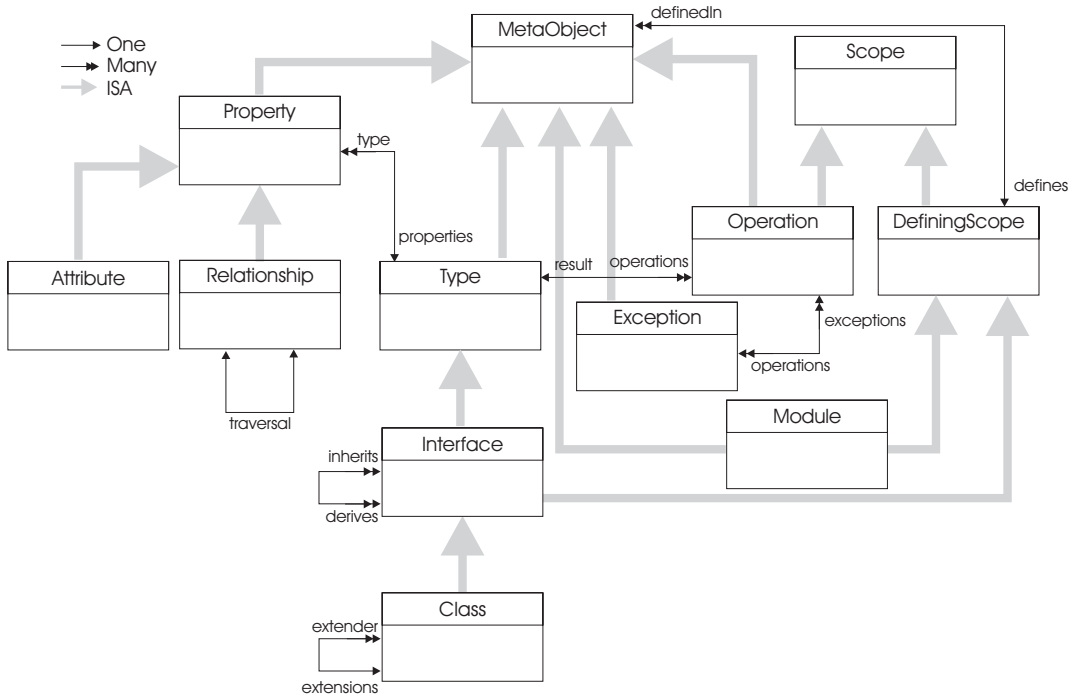


Fig. 2: Extract of ODMG schema repository related to external schema definition

However, current ODMG metadata have some limitations to define external schemas. On the one hand, ODMG specifications do not include metadata to define neither derived classes nor derived interfaces. On the other hand, `ISA` and `EXTENDS` relationships are established in a generic way, without taking into account the schema where they take place in. Therefore, we have proposed an extension to ODMG metadata in order to give support to the definition of external schemas [Torres03]. This extension is illustrated in Figure 3 where new metaclasses and modified ones are depicted with a shaded name.

The extension introduces i) metaclasses for derived classes and derived interfaces whose instances are the derived classes and derived interfaces included in user schemas; ii) metaclasses to model the interfaces and classes of a module in the ODMG schema repository, as well as the inheritance relationships by module (`EXTENDS` relationships are stored in the repository as instances of `ModuleClasses`, indicating for each class its superclass in each schema it is included in. Analogously, `ISA` relationships are stored as instances of `ModuleInterfaces`).

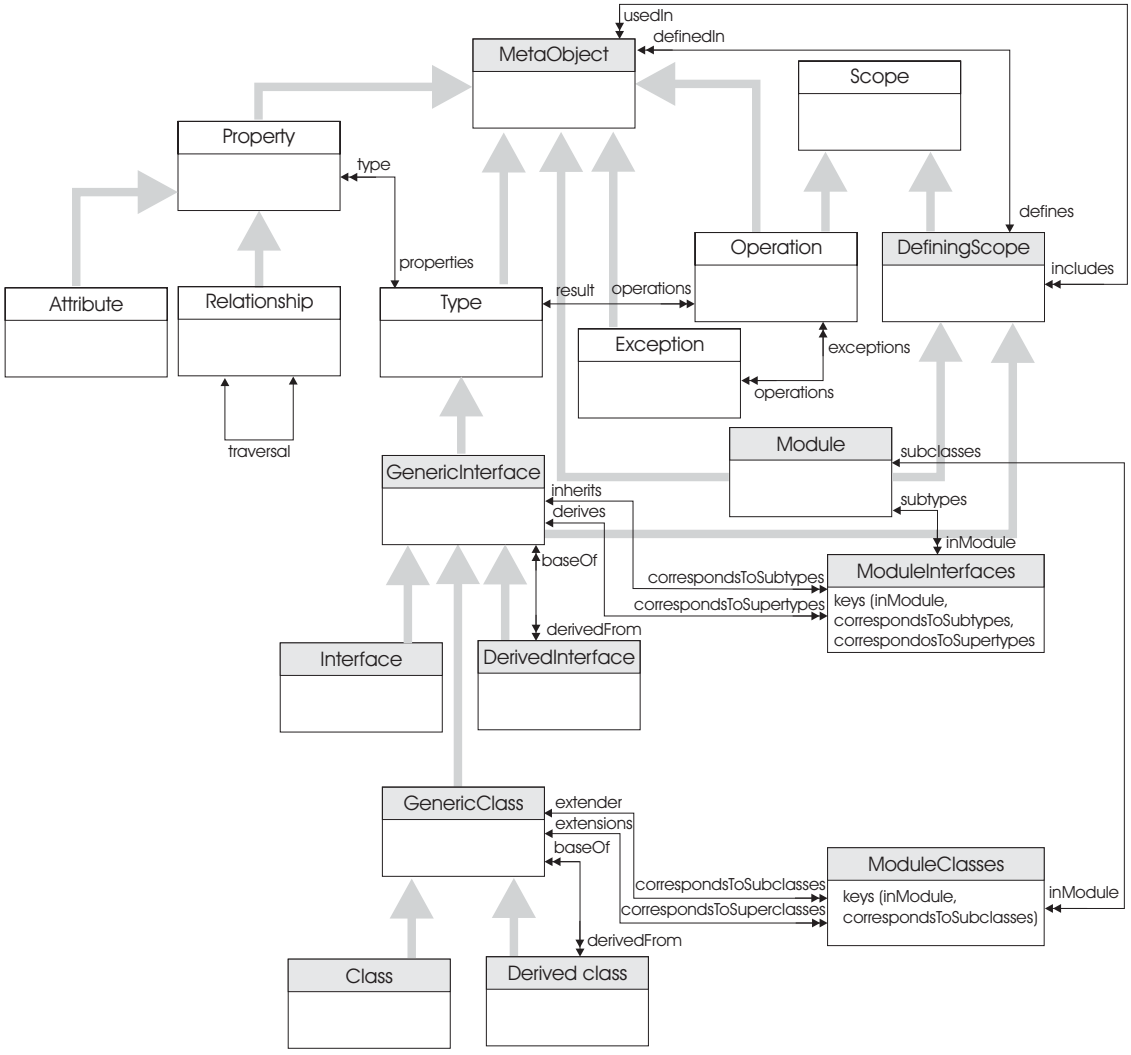
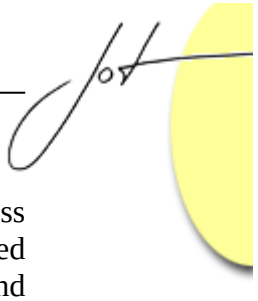


Fig. 3: Extended ODMG schema repository to enable the definition of external schemas

Once we have introduced briefly the ODMG metadata extension to define external schemas in ODMG databases, next section describes the language proposed in this paper to define external schemas showing the effect produced in the extended repository when language operations are processed.

3 EXTERNAL SCHEMA SPECIFICATION

The specification of an external schema deals with the selection of classes and interfaces to be included in external schemas. In the proposed external schema definition mechanism, schema specification is carried out in the repository, in order to reuse existing classes or interfaces. The specification must allow the selection of schema components, derived or not, returning the set of classes and interfaces that initially make up an external schema. It is an initial set because that set may be modified later in schema



closure [Torres01b] and schema generation [Torres01a] steps. The specification process updates the repository adding or modifying the corresponding instances in the related metaclasses (e.g. some metaclasses must be updated to reflect which classes and interfaces make up the defined schema). So, the effect caused in the repository by each schema specification operation of the proposed language must be addressed, illustrating how instances are added, modified o deleted from their metaclasses. The proposed language allows the schema definer to create new external schemas from scratch, or to modify and delete existing external schemas.

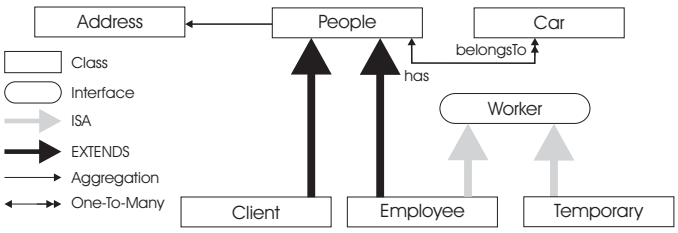
External schema definition language

Next, the EBNF syntax of the proposed external schema definition language is described. This language consists of schema-level operations and component-level (classes and interfaces) operations. Schema-level operations allow defining, modifying or deleting schemas, whereas component-level operations allow adding or deleting components from the schemas.

```
<define_es> ::= <create_es>; | <modify_es>; | <delete_es>;

<create_es> ::= DEFINE ES <name> {
      [<added_component>] }

<modify_es> ::= MODIFY ES <name> {
      [<added_component>]
      [<deleted_component>] }

<added_component> ::= <component_addition> |
      <component_addition> <added_component>

<deleted_component > ::= <component_deletion> |
      <component_deletion> <deleted_component>

<component_addition> ::= ADD CLASS <name>; |
      ADD DERIVED CLASS <name>; |
      ADD INTERFACE <name>; |
      ADD DERIVED INTERFACE <name>; |
      ADD SUBSCHEMA ROOTED BY <class_name> IN SCHEMA <schema_name>; |
      ADD ES <name>;

<component_deletion> ::= REMOVE CLASS <name> [AND NOT USED]; |
      REMOVE DERIVED CLASS <name> [AND NOT USED]; |
      REMOVE INTERFACE <name> [AND NOT USED]; |
      REMOVE DERIVED INTERFACE <name> [AND NOT USED]; |
      REMOVE SUBSCHEMA ROOTED BY <name> [AND NOT USED]; |
      REMOVE ES <name>;

<delete_es> ::= UNDEFINE ES <name> [AND NOT USED]
```

In order to illustrate the operations of this language, we will use an example about an external schema that hides all the data related to employees from the people database.

Figure 4.a illustrates the ODMG schema of the database. Figure 4.b shows the definition of an external schema, which hides data related to employees using the proposed language. The definition of this schema in the proposed language specifies the schema name in the schema definition operation (`DEFINE ES`) and class names in the class addition operations (`ADD CLASS`).




```
DEFINE ES example {
  ADD CLASS People;
  ADD CLASS Address;
  ADD CLASS Car;
  ADD CLASS Manufacturer;
  ADD CLASS Client;
};
```

(a) (b)

Fig. 4: a) People database; b) External schema definition

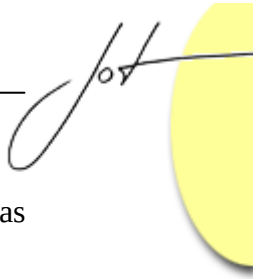
Schema-level operations

In this subsection, the three schema-level operations `DEFINE ES`, `MODIFY ES` and `UNDEFINE ES` are described, which allow creating, modifying and deleting external schemas, respectively.

Schema definition. The `DEFINE ES <name>` operation creates a new external schema identified as `name` provided that in the repository does not exist a schema with that name. The definition of an external schema specifies by means of component-level operations the classes and interfaces that the schema definer wants to include in the schema. The result of this operation is the creation of a new instance in the metaclass `Module`. Therefore, regarding to the previous external schema, a new instance example in the metaclass `Module` is created.

Schema modification. The `MODIFY ES <name>` operation is applied to existing defined external schemas, and allows the incorporation of existing components and the deletion of previously added components. In order to modify a schema named as `name`, a schema with this name must exist in the repository, that is, an instance corresponding to `name` must exist in the metaclass `Module`. If the schema definer wants to add a component to the schema, such a component must be stored in the repository, so that an instance corresponding to that component must be included in the metaclass `MetaObject`. The `includes-definedIn` relationship defined between `DefiningScope` and `MetaObject` is used to avoid the inclusion of a component that already exists in that schema. The same relationship is used to avoid the deletion of a schema component that does not exist in that schema.

Schema elimination. The `UNDEFINE ES <name> [AND NOT USED]` operation deletes from the repository an external schema named as `name`. Optionally, if parameter `AND NOT USED` is specified, each class defined for the schema `name` that is not being used in other schemas is also deleted. This is an optional feature because the schema



definer may want to preserve derived classes that are not used in other schemas as shorthand in queries [Guerrini97] or to be included in new external schemas.

3.3. Component-level operations

The component-level operations allow the addition of classes or interfaces to new schemas or existing ones, and allow the deletion of classes or interfaces from existing schemas. After running these operations, the `usedIn-includes` relationship must be updated adding or deleting the added or removed components, respectively.

Class addition. The `ADD CLASS <name>` operation adds the class `name` to a new schema or to an existing one that is being modified. Such a class must be stored in the repository, and after running this operation, the class `name` belongs to the set of components of the schema that is being defined.

Derived class addition. The `ADD DERIVED CLASS <name>` operation is analogous to the `ADD CLASS <name>` but for derived classes.

Interface or derived interface addition. The operations related to interfaces `ADD INTERFACE <name>` and `ADD DERIVED INTERFACE <name>` are similar to the corresponding ones for classes, except that they deal with interfaces.

Subschema addition. The `ADD SUBSCHEMA ROOTED BY <class_name> in SCHEMA <schema_name>` operation adds to the schema that is being created or modified each class stored in the repository as direct or indirect subclass of the class `class_name` in the schema `schema_name`. The schema `schema_name` must be specified because the class `class_name` may be included in several schemas, and then, its set of subclasses may change from one schema to another. In order to include in the schema the subschema rooted by `class_name` in `schema_name`, `class_name` must be included in `schema_name`, and this fact may be found analysing the `usedIn-includes` relationship.

Schema addition. The `ADD ES <name>` operation adds to the external schema that is being defined each class of the schema `name` stored in the repository.

Class and derived class deletion. The `REMOVE CLASS <name> [AND NOT USED]` and `REMOVE DERIVED CLASS <name> [AND NOT USED]` operations delete the class `name` from the schema that is being modified if such a class is included in that schema. If the option `AND NOT USED` is specified, the class `name` is also deleted from the repository if it is not included in another schema.

Interface and derived interface deletion. The operations related to interfaces, `REMOVE INTERFACE <name> [AND NOT USED]` and `REMOVE DERIVED INTERFACE <name> [AND NOT USED]`, are similar to the corresponding ones for classes, except they are related to interfaces.

Subschema deletion. The `REMOVE SUBSCHEMA ROOTED BY <name> [AND NOT USED]` operation deletes from the external schema the subschema whose root is `name`, that is, deletes from the schema all the subclasses of `name`. However, this operation is executed if the class `name` exists in the schema that is being modified. This operation is

equivalent to execute as many `REMOVE CLASS` as subclasses has the class `name` in the schema that is being modified. Similarly, like happens with previous operations, if the option `AND NOT USED` is specified, deleted classes are also deleted from the repository if neither are used by other classes nor are included in other schemas.

Deletion of schemas included in a schema. Finally, `REMOVE ES <name>` deletes the schema `name` from the external schema that is being modified, deleting therefore, each class and interface included in the schema `name`.

Effect of schema definition in the repository

Figure 5 illustrates the effect that produces the definition of the previous external schema in the repository. In addition, the figure also illustrates the definition of the conceptual schema depicted before (Figure 4.a). It can be seen that `Module` has two instances: one for the conceptual schema, and another for the external schema of the example. Besides, a new defining scope for the external schema with their components, as well as instances corresponding to `EXTENDS` relationships has also been created. In [Torres01a] the schema generation algorithm for obtaining the `EXTENDS` relationship existing between `Client` and `People` in the external schema is described. It can also be seen that the external schema includes all the classes specified by the schema definer but no class is defined in such a schema.

Class

OID	name
#300	Object
#301	Address
#302	People
#303	Car
#304	Client
#305	Employee
#306	Temporary

Interface

OID	name
#200	ObjectT
#201	Worker

Module

OID	Name
#100	ConceptualSchema
#101	**example**

ModuleClasses

OID	schema	class	superclass
#500	#100	#301	#300
#501	#100	#302	#300
#502	#100	#303	#300
#503	#100	#305	#302
#504	#100	#306	#302
#505	#100	#307	#300
#506	**#101**	**#301**	**#300**
#507	**#101**	**#302**	**#300**
#508	**#101**	**#303**	**#300**
#509	**#101**	**#304**	**#302**

ModuleInterfaces

OID	schema	component	superinterface
#600	#100	#201	#200
#601	#100	#306	#201
#602	#100	#307	#201

DefiningScope

OID	name	defines	includes
#100	ConceptualSchema	#201, #301, #302, #303, #304, #305, #306	#201, #301, #302, #303, #304, #305, #306
#101	**example**		**#301, #302, #303, #304**

Fig. 5: Repository once defined the external schema

4 CONCLUSIONS AND FUTURE WORK

In this paper, an external schema definition language for ODMG databases, and its effect in the schema repository have been put forward. The language allows schema definers to create new schemas as well as to update and delete existing ones. Schema specification selects which classes, interfaces, or even schemas, are included in external schemas. The specification is based on the information stored in the repository. After defining schemas, repository information is updated to reflect the changes. The effect on the ODMG schema repository extended for enabling the definition of external schemas of each language operation has been also described. The proposed language is part of an external schema definition methodology that we have developed for ODMG databases. Currently we are working in an extension of this methodology for schema evolution in ODMG databases.

ACKNOWLEDGEMENTS

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

REFERENCES

[Abiteboul91] Abiteboul, S., Bonner, A. 'Object and Views'. *In Proc. ACM SIGMOD International Conference on Management of Data*. pp. 238-247. 1991

[Bertino92] E. Bertino. “A View Mechanism for Object-Oriented Databases” *In Proc. of the 3rd EDBT*. pp. 136-151. 1992

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

[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

[Guerrini97] 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

[Roantree99] 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

[Rundensteiner92] E.A. Rundensteiner. "Multiview: A Methodology for Supporting Multiple Views in Object-Oriented Databases" *In Proc. of the 18th VLDB*. pp. 187-198. 1992

[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

[Torres01a] M. Torres, J. Samos "Generation of External Schemas in ODMG Databases" *In Proc. of IDEAS 2001*. pp. 89-98. 2001.

[Torres01b] M. Torres, J. Samos "Closed Schemas in Object-Oriented Databases" *In Proc. of DEXA 2001*. pp. 826-835. 2001.

[Torres02] M. Torres. Defining external schemas in ODMG databases. PhD Thesis. University of Almeria, Spain (2002)

[Torres03] M. Torres, J. 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

About the authors

Manuel Torres is assistant professor at the Departamento de Lenguajes y Computación of the University of Almeria. 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 associate proffesor 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.