A Multiple View Definition System for Supporting Interoperability among Heterogeneous and Autonomous Databases

R. M. Duwairi+ N. J. Fiddian W. A. Gray

Department of Computer Science
University of Wales College of Cardiff, U. K.
Fax: +44 - 1222 - 874598
E-mail: R.Duwairi@cs.cf.ac.uk

+ Jordan University of Science and Technology, Irbid, Jordan.

Abstract

Users and application programs are increasingly requiring access to and manipulation of data distributed over the multiple sites of a computer network. Different users have different reasons for data integration, and the same user might want to integrate the same distributed information in various ways to suit different roles or tasks. Therefore, multiple integration views are needed to support this diversity. This paper describes the design and implementation of a Multiple View Definition System (MVDS). This system is concerned with the reconciliation of potential conflicts, duplication and redundancy of data descriptions and meaning in the context of supporting various integration views over sets of heterogeneous and autonomous object-oriented databases. MVDS is novel - in that it creates these views without first creating an entire global schema. Also, MVDS is DDL and DBMS-independent because of its flexible and extendible architecture.

1. Introduction

User data processing requirements and capabilities have changed dramatically due to recent progress in network and database technologies which are creating distributed rather than centralised information systems. These new applications often involve accessing and maintaining data from several pre-existing databases and software packages [HUR96, MAD95, VAS95], which are typically located on autonomous software and hardware platforms distributed over the multiple sites of a computer network. This usually means that there are heterogeneity problems to be resolved [DOG95, GAR96, OZS91], where heterogeneity may arise at the hardware, operating system, data model, DBMS, query language and/or semantic levels.

Conforming the component databases in such an environment to a common data model, query language and DBMS is very expensive and, more importantly, rarely acceptable to the users of these databases. Such databases and their users are usually autonomous and wish to remain so even after joining a multidatabase (MDB). In this environment, users want to access information located in any database without learning a new query language, data model or DBMS. Ideally, a user would like to employ his/her own local query language, data model and DBMS to access other databases distributed over the network. Furthermore, he/she would like to see a consistent view of all the distributed, overlapping and heterogeneous information available.

This paper describes our approach to supporting interoperability in a heterogeneous object-oriented (OO) MDB system. It is based on creating multiple integration views above the participating databases. These views support different perceptions of the distributed and heterogeneous information available. Our main focus here will be on the architecture of the software prototype, called MVDS for Multiple View Definition System, which we have developed to act as a schema integration workbench in a heterogeneous MDB context. Section 2 gives the background to our approach. Section 3 describes the architecture and operation of MVDS. Section 4 draws conclusions.

2. Background

2.1 Single versus Multiple Global Schemas for Multidatabase Interoperability

One way to support MDB interoperability is to define a single global schema above all the participating schemas. This schema is a consistent, non-redundant and complete representation of all the distributed and heterogeneous information available [OZS91, QUT92]. This approach, however, has a number of serious disadvantages. For example, the global schema may grow very large, especially if the participating databases are very numerous and/or individually large in schema terms. Also, it provides only a single semantic view[1] of the entire MDB which is decided on by the integrator, not the users, and all structural and semantic conflicts have to be identified and resolved even though some of them might never arise. Thus the creation of such a schema is complex, and it is ill-suited to applications that require frequent modifications to their local schemas and, consequently, to the composite global schema.

In [DUW96b] we argue that a better, more flexible approach to support heterogeneous MDB interoperability would result from creating several integrated views above the MDB - with each view tailored towards a certain group of users or application programs. These views are created by applying a set of semantically-rich integration operators, through a language we call MVDL (Multiple View Definition Language). A major goal in the design of MVDL was that it should be capable of reconciling the semantic and structural differences that are likely to arise when integrating heterogeneous and autonomous databases.

Supporting MDB interoperability through this approach offers *selectivity* and *customisability* (or *multiple semantics*). Selectivity means that an integrator need choose only the local information of interest when creating views at the global level. This has several advantages. For example, it reduces the complexity of the schema integration process because only the relevant parts of the participating schemas are involved in this process. The size of a generated global view is small compared with the case when a single, entire global schema is defined, this saves the user the burden of searching a large global schema to locate information. Providing a schema integration tool (i.e. MVDS) that aids in defining multiple views saves the user the effort of manually integrating the selected information to create his/her views as is the case with the multidatabase language approach [LIT86, LIT93, KRI91]. Finally, selectivity allows a large number of participating databases to join the MDB. Customisability, on the other hand, means that the integrator can specify exactly how he/she wants to view chosen information, and supports multiple semantic perspectives of the participating databases at the global level.

2.2 Multiple View Definition Language

In this section, we give a brief description of MVDL. For more details refer to [DUW96b].

The rationale behind the design of MVDL is that in a MDB environment operations such as data selection, focusing, pruning, fusion and mismatch processing are highly desirable. MVDL supports such operations via its operators. These operators are divided into five main groups depending on their method of integrating information. They include importation operators, generalisation operators, specialisation operators, merger operators and aggregation operators. Which operator to apply to a given set of integration units (i.e. local classes and/or their properties) depends on the relationship between these units (e.g. equivalence, homonymity, overlap, ..., etc.) and on integrator preferences.

The operands of MVDL operators (i.e. local integration units) are typically located in several autonomous databases. Therefore, establishing real world semantic relationships between these units is an essential part of the functionality of MVDL. Each application of an MVDL operator will create a set (typically a one element set) of virtual classes. These classes provide a consistent and integrated view of the distributed and heterogeneous local information.

In what follows, prop() is a function that returns the properties of its argument; ext() is a function that returns the extension of its argument; D, D1 and D2 are sets of derived properties; Gc, C1' and C2' are global classes; and Lc, C1 and C2 are their corresponding local classes.

1. Importation Operators: this group consists of the operators: Include, Include*, Include1, Include1*. It is responsible for importing local classes into a global view. These operators are useful if the integrator is interested in forming a set of local classes (typically from several databases) without actually integrating them. Table 1 shows the contribution of each importation operator.

2. Generalisation Operators: this group consists of the operators: Union, Union*, and Union1. They integrate local classes by linking them to a common superclass. The local classes have to be equivalent, synonymous or overlapping. Table 2 shows the contribution of each generalisation operator.

[1] Note that it is possible to define external schemas above a global conceptual schema in a MDB [SHE90]. However, this approach is limited in its power compared with supporting multiple global views. For more details see [DUW96b].

Operator	Semantics	Extension	Properties
Gc=Include (Lc)	imports a single class.	• ext(Gc)= ext(Lc).	• Prop(Gc)= prop(Lc) $\bigcup$ D.
Gc=Include1 (Lc) where ζ	imports the subset of instances of Lc that satisfy the selection predicate ζ.	• ext(Gc)=subset of ext(Lc).	• Prop(Gc)= prop(Lc) $\bigcup$ D.
Gc=Include* (Lc)	imports the class hierarchy rooted at Lc.	• ext(Gc)=ext(Lc) for all Gcs derived from Lcs.	• Prop(Gc)= prop(Lc) $\bigcup$ D for all Gcs derived from Lcs.
Gc=Include1* (Lc) where ζ	imports the subset of instances of the class hierarchy rooted at Lc that satisfy the selection predicate ζ.	• ext(Gc)= subset of ext(Lc) for all Gcs derived from Lcs.	• Prop(Gc)= prop(Lc) $\bigcup$ D for all Gcs derived from Lcs.

Table 1: Importation Operators

Operator	Semantics	Extension	Properties
Gc=Union(C1, C2)	generates a superclass for two local classes. All three classes are added to the view.	• ext(Gc)=ext(C1) $\bigcup$ ext(C2). • ext(C1')=ext(C1). • ext(C2')=ext(C2).	• prop(Gc) = prop(C1) $\bigcap$ prop(C2) $\bigcup$ D. • prop(C1' =prop(C1) $\bigcup$ D $\bigcup$ D1. • prop(C2')= prop(C2) $\bigcup$ D $\bigcup$ D2.
Gc=Union1 (C1, C2)	generates a superclass for two local classes. Only the super-class is included in the view.	• ext(Gc)=ext(C1) $\bigcup$ ext(C2).	• prop(Gc) = prop(C1) $\bigcap$ prop(C2) $\bigcup$ D.
Gc=Union* (C1, C2)	generates a common superclass for two local class hierarchies.	•ext(Gc)=ext(C1) $\bigcup$ ext(C2). •ext(rest)=ext(their corresponding local classes).	• the same as Union for the classes Gc, C1' & C2'. • prop(rest of Gcs) = prop(their corresponding Lcs) $\bigcup$ D.

Table 2: Generalisation Operators

3. Specialisation Operators: this group consists of two operators: Intersect and Intersect1. They integrate a set of overlapping local classes by generating a common subclass. Table 3 shows the effects of each of this group of operators.

4. Merger Operators: this group consists of two operators: Combine and Combine*. They integrate equivalent or synonymous local integration units by merging them into one global class. Table 4 shows the effects of each of this group of operators.

5. Aggregation operators: this group consists of just one operator: Aggregate. It is an overloaded operator that can be applied to classes, properties or both. It integrates local integration units by aggregating them into a higher degree aggregate class.

6. The Difference Operator: this is used to create a global class from two local classes. The global class extension is equal to the set of instances that belong to the first class but not to the second, and its properties are equal to the properties of the first class.

Operator	Semantics	Extension	Properties
Gc=Intersect (C1, C2)	generates a common subclass for two local classes. All three classes are added to the view.	• ext(Gc)=ext(C1) $\bigcap$ ext(C2). • ext(C1')=ext(C1). • ext(C2')=ext(C2).	• prop(Gc)= prop(C1) $\bigcup$ prop(C2) $\bigcup$ D $\bigcup$ D1 $\bigcup$ D2. • prop(C1') = prop(C1) $\bigcup$ D1. • prop(C2') = prop(C2) $\bigcup$ D2.
Gc=Intersect1 (C1, C2)	generates a common subclass for two local classes. Only the subclass is added to the view.	• ext(Gc)=ext(C1) $\bigcap$ ext(C2).	• prop(Gc) = prop(C1) $\bigcup$ prop(C2) $\bigcup$ D.

Table 3: Intersection Operators

Operator	Semantics	Extension	Properties
Gc=Combine (C1, C2)	merges two integration units into one global class.	¥ ext(Gc)=ext(C1 $\bigcup$ ext(C2).	• prop(Gc) = prop(C1) $\bigcup$ prop(C2) $\bigcup$ D.
Gc=Combine* (C1, C2)	merges two generalisation hierarchies into one global hierarchy.	• ext(Gc)=ext(C1) $\bigcup$ ext(C2). • the same holds for the rest of the classes in the global hierarchy.	• prop(Gc) = prop(C1) $\bigcup$ prop(C2) $\bigcup$ D. • the same holds for the rest of the classes in the global hierarchy.

Table 4: Merger Operators

7. The Connect Operator: this is used to integrate two local classes by generating an is-a link between them.

Note that with all the MVDL operators, it is possible to choose a subset of the default properties for the generated global classes. Here the user, rather than the system, is responsible for choosing these properties. This has two advantages. Firstly, the user can specify the set of properties he/she is interested in and this supports customisability. Secondly, MVDS only has to apply reconciliation transformations and derive instances for a subset of the possible properties. It is also important to note that $\bigcup$ and $\bigcap$ are not simple union and intersection, whether applied to class properties or extensions, due to the overlap and heterogeneity among local databases.

3. MVDS Architecture

The external architecture of MVDS, see figure 1, consists of three interfaces, namely: the *Source Schema Specification Interface,* the *Global View Definition Interface* and the *Target Schema Specification Interface*. It is implemented in Quintus Prolog [QUI91] coupled to OSF/Motif [HEL94].

An essential objective in the design of MVDS is adaptability, i.e. its functionality should *not* be limited to a particular set of DBMSs and DDLs. Therefore, all its internal processing and operations are founded on a canonical data model (CDM). The general usefulness of a CDM is based primarily on its capability to assist in mapping (translation) between logically heterogeneous local concepts and global concepts. The use of such a model reduces the number of translators that need to be developed. To this end, the OO model proves to be very suitable [BER94, PIT95], and therefore, MVDS's CDM is based on the Object Model of the ODMG-93 standard [CAT94]. It supports objects, classes, object identity (OID), persistence, aggregation and multiple inheritance.

Translators that map between MVDS's CDM and a new user-specified DDL are attached to MVDS to adapt it to a new DBMS and DDL. This explains the purpose of the first and third interfaces listed above, which we consider next.

3.1 Source Schema Specification Interface

This has two phases: specification and application. During the specification phase, the syntax of the source schemas (i.e. their DDL description) is recorded in a file using Prolog's [BRA90] Definite Clause Grammar (DCG) [PER80] which is a BNF-compatible format. This syntactic description is complemented by a special-purpose semantic language whose role is to specify the mapping of source schema syntactic constructs and corresponding semantic contents into the MVDS intermediate CDM. The following is an extract (simplified for exposition purposes) from the Source Schema Specification file for a source schema written in ODL[2]:

```
interface_declaration -->
        [interface], [Name],
        optional_inheritance_specification,
        optional_type_property_list,
        optional_persistence_declaration,
        optional_interface_body,
        {create_odmg_frame}.
```

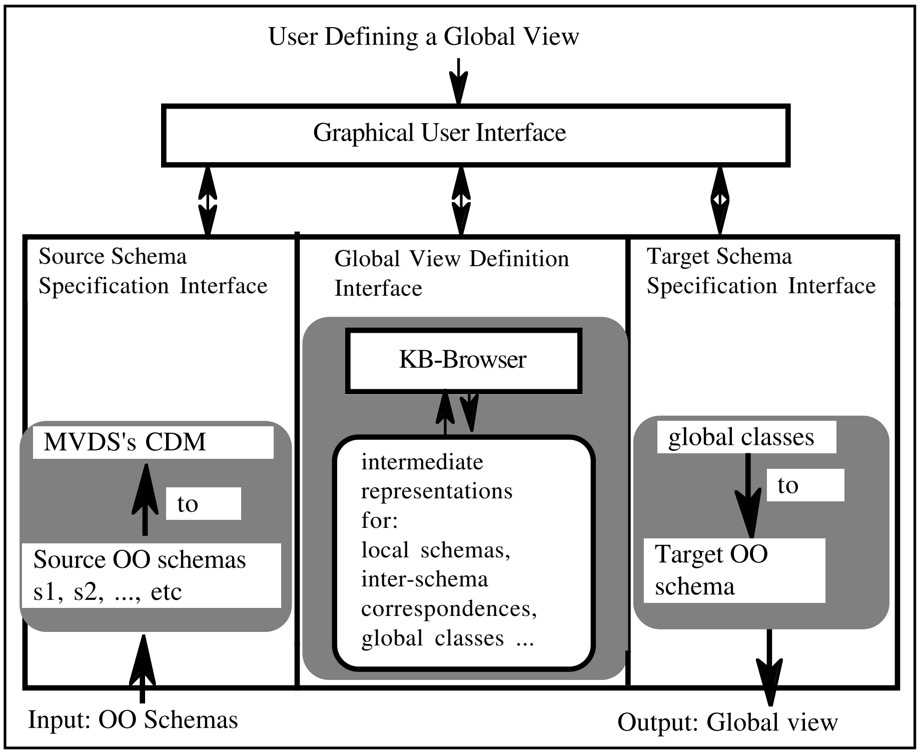

Figure 1: MVDS Architecture

In this extract, DDL tokens are placed between square brackets. O*ptional_inheritance_specification* is responsible for parsing the inheritance specification for an interface (class[3]), *optional_type_property_list* is responsible for parsing the extent and the key specifications for a given ODMG interface, and so on. Complementing the syntactic specification, *create_odmg_frame* will transform the syntactic description of an interface into a Prolog frame that subsequently is asserted into the MVDS knowledge base. The specification phase is invoked once per DDL.

The application phase of the Source Schema Specification Interface, by contrast, is responsible for introducing a source schema to MVDS, i.e. the file that contains its syntactic

[2] ODL (Object Defintion Language) is the DDL of the Object Model of the ODMG-93 standard.

[3] Throughout this paper, the term class is used to refer to the schematic description of objects.

and semantic descriptions is connected to MVDS. This process is performed once for each source schema. When using MVDS with a known DDL, the process of loading a source schema specification file is reduced to clicking the appropriate button in the system's main window. This specification is used to translate the source schema into intermediate (i.e. CDM) form.

3.2 Target Schema Specification Interface

This is responsible for allowing a user to stipulate to MVDS his/her preferred DDL specifications for the output global view, therefore its specification phase (see below) is dealt with before performing the integration process. This interface also has two phases: specification and application. During the specification phase, templates that facilitate the interpretation of a view class's internal representation into a target DDL are specified. Each template contains parameters which refer to the necessary information for generating the syntax of the output DDL. The following extract depicts a template that will transform a virtual class from its internal form into ODL (nl stands for newline):

```
write_interface(InterfaceName, Extent, Keys, Superclasses, PropertyNames,
          PropertyTypes):-
         write('interface '), write(InterfaceName), write(' '),
         write_optional_inheritance(Superclasses), nl,
         write('('), write_extent(Extent), nl,
         write_keys(Keys), write(' )'), nl, write('{ '),nl,
         write_properties(PropertyNames, PropertyTypes), nl, write('};').
```

This specification phase is performed once per target DDL. In MVDS, a target schema contains its interfaces (classes) description and materialisation rules that are responsible for deriving its instances from its origins. These rules have a POSTGRES-rule like syntax (cf. Section 3.3.4). The specification of virtual class materialisation rules depends on the MVDL operator used to create the class, its origin correspondences (i.e. their semantic relationship types and property mappings) and its global properties. These rules are effected automatically and, therefore, their specifications are hard-wired into MVDS. So the specification phase of the Target Schema Specification Interface is limited to defining the templates necessary to produce a target DDL description for the virtual classes of a given global view.

During the application phase of this Interface, the templates defined during the specification phase are connected to MVDS and are used to transform the current view into a user-specified DDL. This process is undertaken once for each generated global view.

3.3 Global View Definition Interface

This central component of MVDS is responsible for constructing a global view either from scratch by applying MVDL operators to local integration units, or by evolving an existing global view. The algorithm followed in creating a global view from scratch is described in this paper, while constructing a global view by evolution is the subject of a forthcoming paper.

3.3.1 Global View Definition Algorithm

This subsection gives an overview of the algorithm followed to create a global view using MVDS, the subsequent subsections give details of major parts of this algorithm:

1. Connect the relevant Source and Target Schema specifications to MVDS.
2. Transform the participating local schemas from their native DDL definitions into MVDS's CDM representation by applying the relevant Source Schema specification that was defined in step-1 above.
3. Create the required virtual classes by applying MVDL operators to the local integration units[4]. This step is repeated until the definition of the global view is complete.
4. Close the completed view by importing all the classes that it refers to.
5. Transform the view from MVDS's CDM representation into a user-specified DDL by applying the Target Schema specification that was defined in step-1 above. View materialisation rules are created during this transformation.

[4] This step involves detecting possible similarities and conflicts between the local schema contents, it also involves reconciling these conflicts once determined (see Section 3.3.2).

3.3.2 Inter-schema Correspondence Establishment

Schema integration represents the merging and restructuring of data stored in autonomous databases. As such, this data is very likely to be duplicated and inconsistent. A trivial way of performing schema integration is simply to concatenate the relevant information at the global level (i.e. the global schema contents are the union of the local schema contents). However, this solution is unacceptable as the essential problems caused by heterogeneity, notably duplication and inconsistency, are not solved [OZS91]. Therefore, during schema integration it is essential to analyse the participating local schema contents in order to identify common types of information, detect and reconcile potential conflicts, and eliminate data duplication and inconsistency.

Descriptions of data (i.e. meta-data) play an important role in analysing and understanding the semantic content of databases. These descriptions include database schemas and data dictionaries. However, they are not enough to totally understand the semantic contents of a given database. This is true partly because as we move from the real world to the represented world (i.e. the database world), some of the real world semantics are lost [GAR96], and partly because of the limited expressiveness of current data models [SAL91]. Consequently, it is not possible to fully automate the process of analysing and understanding the semantic contents of participating databases. Performing that process manually, on the other hand, is very tedious, error-prone and time-consuming. Therefore, MVDS follows a graphical and semi-automatic approach to analysing local schema contents that depends on a probabilistic lexical matching between schema pair elements and also on user advisory interaction.

In general, classes can be related to each other by one of the following real world semantic relationships: equivalence, synonymity, homonymity, overlap, containment or disjunction. To determine the type of the real world semantic relationship that may exist between a pair of potentially mergeable classes, a probabilistic lexical matching process is carried out on their properties. Two classes are considered *related* if their property equivalence percentage is greater than or equal to a minimum threshold value specified by the user. MVDS employs a set of heuristics to deduce types of inter-class semantic relationship. As this comparison process is carried out on the database representation of real world objects, the set of heuristics cannot guarantee complete accuracy. Therefore, all MVDS findings are presented to the user, who has the final say in accepting or rejecting the system's suggestions.

MVDS incrementally analyses participating local schema contents - its analysis process is triggered whenever a user applies an MVDL operator to the required local integration units. If these units have already been compared on a previous occasion or if it is possible to automatically deduce some knowledge about their semantic contents (see [DUW96a]), then it is unnecessary to carry out the comparison process in the first case, or it is greatly simplified in the second case. Any previous knowledge can be reused as all established correspondences are stored in the MVDS knowledge base. Analysing the local schema contents in this manner significantly reduces the number of comparisons necessary. Traditionally, it has been carried out by comparing every integration unit in a given schema with every integration unit in all other schemas being integrated.

Reconciliation transformations, that mediate between conflicting representations of the same real world objects, are also defined during the above process.

3.3.3 Inter-object Equivalence

Once the semantic contents of the participating schemas are clearly understood by the identification of their semantic inter-relationships, it is necessary to establish their instance equivalences. For example, if it has been established that class EMPLOYEE from one database is equivalent to class EMPLOYEE from another database, then the next major step is defining the rules that will determine when a given employee from the first database (say Sam) is the same Sam in the other database, i.e. both Sams refer to the same real world Sam. OIDs work fine within the boundaries of a single database, but in the case of multiple databases an alternative method is needed [KEN91].

MVDS, as a schema integration tool, is responsible for defining the rules that will determine inter-object equivalences. However, the construction of global objects and the assignment of their global OIDs is a query processor task. MVDL operators are divided into two groups with respect to their role in inter-object equivalence establishment. The first group strictly maintains a one-to-one correspondence between global objects and local objects within the boundaries of a single global view. It consists of the operators: Include, Include1, Include* and Include1*. So, there is no need to establish inter-object equivalence rules for the virtual classes generated using these operators.

The second group does not maintain such a correspondence between global and local objects[5], and it is then necessary to define inter-object equivalence rules. This group consists of the operators: Combine, Combine*, Union, Union*, Union1, Intersect, Intersect1, Connect, Difference and Aggregate. ODMG-93 allows users to define keys for interfaces. These keys may be utilised to establish inter-object equivalence rules. For example, proxy employees with the same social security number represent the same real world employee. Note that the relationship between the classes DB1-EMPLOYEE and DB2-EMPLOYEE[6] is established during the semantic heterogeneity detection and reconciliation phase of MVDS. If there are no keys defined or these keys are insufficient to identify inter-object equivalences for a given pair of classes, then their common properties are used to establish their instance equivalence (these sometimes are referred to as *same predicates* [SCH94]). MVDS defines inter-object equivalence rules by creating a function called '=g' (for global equivalence). Each =g function will state conditions that determine when two local objects are equivalent[7]; =g utilises class keys or common properties. If class keys are defined and these have been identified as equivalent properties during the semantic heterogeneity detection and reconciliation phase, MVDS automatically generates =g. Otherwise the user is responsible for defining that function.

3.3.4 Global View Materialisation Rules

The instances of the generated global classes are derived from their corresponding local ones. This means that a global instance might be constructed by aggregating several local instances, and the same instance may have heterogeneous representations in different databases. Specifying the rules for deriving global instances and reconciling heterogeneous representations is the task of the set of materialisation rules that accompany each global view. MVDS creates such rule sets *automatically*; to do this it utilises the MVDL operators, inter-schema correspondences, and the generated virtual classes. When a query is posed against a global class, the query processor will utilise its corresponding materialisation rules to decompose the global query into local subqueries. Currently, these rules have a POSTGRES-rule like syntax [STO90]:

```
Rule <name>
on retrieve to <virtual-class-properties>
do instead retrieve <local-corresponding-properties>
where <condition>
```

Where:

<virtual-class-properties>:	are the properties listed in the virtual class interface;
<local-corresponding-properties>:	are the actual properties to be retrieved - note that a virtual property may draw its values from several possibly heterogeneous local properties;
<condition>:	is the condition under which the rule is executed.

3.3.5 Global Object Composition and OID Assignment

Global object generation comes after the integration system has succeeded in providing the end-user with a common interface to the heterogeneous MDB system. Generally, this may be achieved by either creating the global objects on demand or explicitly populating the global view with all its instances in advance. MVDS views can be used in either case.

Maintaining a MDB identity for real world objects is essential if component databases are to interoperate successfully. The purpose of global identification is to provide MDB users with unique, persistent and unambiguous references to objects. This is not a trivial task when a user requires access to objects stored in several databases or when object identification methods in local databases are different [ELI91].

One way of achieving this goal is to maintain OID mapping tables or expressions at the global level. Whenever a global object is created for the first time, a new OID is assigned to it, and an entry is inserted in the OID mapping table (or expression) which maps between this new

[5] i.e. a single global object can be created by merging several local objects in different databases.

[6] DB1-EMPLOYEE and DB2-EMPLOYEE are local classes in different databases.

[7] =g is defined at the meta-level (i.e. at the class level) not at the instance level.

OID and its corresponding local OIDs. Using this technique, each time a global object is created by integrating the same set of local objects, it will be assigned the same OID (i.e. OID persistence is achieved at the global level) [ELI91].

4. Conclusions

Our paper has described the design and implementation of a multiple view definition system (referred to as MVDS) that facilitates the efficient production of multiple integration views from the structural components and semantic contents of logically heterogeneous OO databases. These views provide a consistent and non-redundant representation of distributed participating database information. They are also customised to satisfy the needs of different user groups or application programs.

The development of MVDS has been carried out entirely in terms of logic programming/Prolog language technologies. In the context of a complete heterogeneous OO database environment, MVDS offers the potential for users to work in terms of integrated and customised global views - where each global view is designed specifically to meet the needs of a particular user group or application program. Using this tool a user can formulate queries against his/her customised global view and these queries will be executed by means of MVDS-mapped requests against the underlying local schemas. The mapping of queries against global concepts to queries against local concepts is facilitated by the set of materialisation rules that accompany each global view. These rule sets are generated automatically by MVDS.

MVDS's architecture is novel - in that it incrementally analyses the participating local schema contents. This permits the reuse of the knowledge already accrued from earlier comparison processes [DUW96a]. MVDS is not confined to a particular DBMS or DDL. This is achieved by employing the three-interface architecture we have described and a rich CDM. Finally, the availability of a rich set of integration operators (MVDL) and the declarative nature of Prolog substantially increase the degree to which the schema integration process is automated.

References

[BER94] Bertino E. et al, "Applications of Object-Oriented Technology to the Integration of Heterogeneous Database Systems", Distributed and Parallel Databases, Vol. 2, pp. 343-370, 1994.

[BRA90] Bratko I., "Prolog Programming for Artificial Intelligence", 2nd Edition, Addison-Wesley, 1990.

[CAT94] Cattell R. G. G. (ed.), "Object Database Standard: ODMG-93, Release 1.1", Morgan Kaufmann Publishers, 1994.

[DOG95] Dogac A. et al, "METU Interoperable Database System", SIGMOD RECORD, Vol. 24, No. 3, pp. 56-61, 1995.

[DUW96a] Duwairi R. M., Fiddian N. J. and Gray W. A., "Schema Integration Meta-Knowledge Classification and Reuse", Proceedings of BNCOD14, UK, pp. 1-17, 1996.

[DUW96b] Duwairi R. M., Fiddian N. J. and Gray W. A., "Multiple Views for Interoperability in a Heterogeneous Multidatabase System", Submitted for publication.

[ELI91] Eliassen F. and Karlsen R., "Interoperability and Object Identity", SIGMOD RECORD, Vol. 20, No. 4, pp. 25-29, 1991.

[GAR96] Garcia-Solaco M., Saltor F. and Castellanos M., "Semantic Heterogeneity in Multidatabase Systems", in "Object-Oriented Multidatabase Systems: A Solution for Advanced Applications", ed. by Bukhres O. A. and Elmagarmid A. K., Prentice Hall, 1996.

[HEL94] Heller D. and Ferguson M., "The Definitive Guides to the X Window Systems: Motif Programming Manual for OSF/Motif Release 1.2", Vol. 6A, O'Reilly & Associates, 1994.

[HUR96] Hurson A. R. and Bright M. W., "Object-Oriented Multidatabase Systems", in "Object-Oriented Multidatabase Systems: A Solution for Advanced Applications", ed. by Bukhres O. A. and Elmagarmid A. K., Prentice Hall, pp. 1-36, 1996.

[KEN91] Kent W., "The Breakdown of the Information Model in Multidatabase Systems", SIGMOD RECORD, Vol. 20, No. 4, 1991.

[KRI91] Krishnamurthy R., Litwin W. and Kent W., "Interoperability of Heterogeneous Databases with Semantic Discrepancies", Proceedings of the First International Workshop on Interoperability in Multidatabase Systems, Kyoto, Japan, pp. 144-151, 1991.

[LIT86] Litwin W. and Abdellatif A., "Multidatabase Interoperability", IEEE Computer, Vol. 10, No. 12, pp. 10-18, 1986.

[LIT93] Litwin W., "O*SQL: A Language for Object Oriented Multidatabase Interoperability", in "Interoperable Database Systems (DS-5)", ed. by Hsiao D. K., Neuhold E. J. and Sacks-Davis R., Elsevier Science Publishers B. V., North Holland, pp. 119-137, 1993.

[MAD95] Madnick S. E., "From VLDB to VMLDB (Very MANY Large Databases): Dealing with Large-Scale Semantic Heterogeneity", Proceedings of the 21st VLDB Conference, Zurich, Switzerland, pp. 11-16, 1995.

[OZS91] Ozsu M. and Valduriez P., "Principles of Distributed Database Systems", Prentice Hall, 1991.

[PER80] Pereira, F. C. N. and Warren D. H. D., "Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks", Artificial Intelligence, Vol. 13, pp. 231-278, 1980.

[PIT95] Pitoural E., Bukhres O. and Elmagarmid A., "Object-Orientation in Multidatabase Systems", ACM Computing Surveys, Vol. 27, No. 2, pp. 141-195, 1995.

[QUI91] Quintus Prolog, Release 3, Quintus Corporation, 1991.

[QUT92] Qutaishat M. A., Fiddian N. J. and Gray W. A., "A Schema Meta-Integration System for A Heterogeneous Object-Oriented Database Environment - Objectives and Overview", Proceedings of the NordDATA'92 Conference, Tampere, Finland, pp. 74-92, 1992.

[SAL91] Saltor F., Castellanos M. and Garcia-Solaco M., "Suitability of Data Models as Canonical Models for Federated Databases", SIGMOD RECORD, Vol. 20, No. 4, pp. 44-48, 1991.

[SCH94] Scholl H. and Schek H. J., "Object Algebra and Views for Multi-Objectbases", in "Distributed Object Management", ed. by Ozsu T. M., Dayal U. and Valduriez P., Morgan Kaufmann Publishers, 1994.

[SHE90] Sheth A. and Larson J., "Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases", ACM Computing Surveys, Vol. 22, No. 3, pp. 183-236, 1990.

[STO90] Stonebraker M. et al, "On Rules Procedures, Caching and Views in Database Systems", Proceedings of the ACM-SIGMOD Conference on Management of Data, Atlantic City, 1990.

[VAS95] Vaskevitch D., "Very Large Databases How Large? How Different", Proceedings of the 21st VLDB Conference, Zurich, Switzerland, pp. 677-685, 1995.