Semantic Requirements for Sharing Behaviours in Federated Object-Oriented Database Systems

Mohamed B. Al-Mourad[*] William A. Gray Nick J. Fiddian

Department of Computer Science,
Cardiff University,
CF24 3XF, U. K.
e-mail {Mohamed.B.Al-Mourad, W.A.Gray, N.J.Fiddian}@cs.cf.ac.uk

Abstract: This paper is concerned with reusing and sharing behaviours of component Object Oriented Database Systems in a tightly coupled federated multidatabase system environment. In such a federation, users benefit from exploiting the behaviours available in the participating components that are not available in their local systems so they can capitalise on investment made in writing their code. To facilitate this, users need to know which behaviours can be reused and shared and what are the requirements of reusing and sharing these behaviours. This means that the users will be able to build their integrated federation views augmented with behaviours of their preference.

1 Introduction

Advances and developments in the area of communications and networking technology have created increasing interest from large organisations who wish to make their existing and currently disjoint databases interoperate efficiently and transparently in a way which guarantees that business is transacted more effectively and suitably to each user's needs. These databases are typically heterogeneous, running on a variety of database management systems which may be incompatible in syntax and semantics due to differences in their respective data models and corporate usage. The creation of an environment that permits the controlled sharing and exchange of information among multiple heterogeneous databases has been identified as a key issue in the future of database integration research [SSU90], [HM93]. In such environments, it is necessary that each component database system is not only able to communicate with other databases but that it can also reuse and share the entire functionality of services already provided for other existing databases [Dr93], [PTR96]. In the context of *object oriented database systems* (OODBSs), services are the set of methods that describe the behaviour of a particular class in a database system. Sharing these services saves effort, cost and time where the investment made in developing them can be exploited again by the original owner of these services and also by new users in the interoperation realm. For example, when two bank databases are integrated due to merger, management might

[*] This work has been made possible by Ph.D. studentship funding from University of Aleppo, Syria, whose support is gratefully acknowledged.

require that the loan-granting services/procedures in one bank will be adhered to by both databases.

In order to make the independently developed database systems concerned interoperable and able to reuse and share their behaviours, they must agree on the meaning of their data, services, etc. In other words, the heterogeneities that arise due to independent design must be reconciled in different ways to fit new requirements.
Numerous approaches have been proposed to integrating database systems. Systems that support database interoperability are called *Multidatabase Systems* (MDBS). Their architectures range from tightly coupled to loosely coupled [SL90], and they employ static [DH84], [Mo87], [CHS91], [QFG92], [DFG96b] or dynamic [Li93], [GL98] views in accessing data from multiple databases.

Recent research on integrating OODBs has focused mainly on exploiting and understanding the semantics of the structural part of their schema components (structural semantics). However, a schema is a combination of both a structural part, represented by attributes, and a behavioural part, represented by methods or operations that can be executed on or by an object. In our past research [QFG92], [DFG96b], [DFG96a] we focused on exploiting and understanding the structural part. We continue here to investigate the importance of the behavioural part (behavioural semantics) of the schema in the integration process. We consider both structural and behavioural semantic heterogeneity, showing that they are both important [Dr93] in this context.

Section 2 of this paper presents the basic characteristics of the *Object Model* (OM) used in our research. In section 3 we explain the concept of reusing and sharing behaviours at the global level. In section 4 we explain the systematic and semantic requirements of reusing and sharing behaviours. Finally, section 5 contains the conclusion of this paper and a brief discussion of further work.

2 Overview of our Object Oriented Database Model

The basics of the database model used here are very similar to the OM proposed in the ODMG standard [Ca97]. Real world entities are modelled as a set of objects. Let O be an infinite set of object instances. An object o_i O consists of *structure properties* (attributes of the object) and *behaviour properties* (the methods or operations that can be executed on or by the object). We will refer to the attributes and behaviours of an object as *object properties* [Fa88], [AB91], [CAG98] and together they identify the object's *Type*. Each object has an independent unique system-generated value called its *object identifier* (OID). Two objects can be either identical (that is, they are the same objects) or equal (i.e. they have the same value). Objects that have the same properties (Type) are grouped into sets called classes.
Let C be the set of all classes in a database. A class c   C has a unique class name, properties (Properties(c)) and type (Type(c)), and a set of instances indicated by class extent or (Ext(c) = o | o   c)[1].

[1] Where the symbol   is defined based on the object identities of the object instances.

Each attribute (a) of a class (c) has a name and a domain dom(a), where the domain of an attribute is a set of values from which the attribute can take its value, and its type can be either a primitive object (integer, string, etc.) or a non-primitive object which is an instance of one of the database classes.
Each method (m) has a signature, which defines the name of the method, the name and type of each of its arguments and the types of value(s) returned[2]. We will refer to the properties of a class by (*classname.propertyname*).

Formally, We denote the properties of class c by Properties(c)={A, M}, where:

A is the set of all attributes of class c: A= $\{a_1: D_1, \ldots\ldots, a_n: D_n\}$; where D_i (i = 1, n) is the type of an attribute and is either primitive or non-primitive.

M is the set of all methods of class c: M= $\{m_1, m_2, \ldots\ldots, m_j\}$; where m_i (i = 1, j) is the signature of method i and has the form:

$$\text{Name}\ (Arg_1{:}T_1, \ldots\ldots, Arg_k{:}T_k) \quad (R_1{:}S_1, \ldots\ldots, R_p{:}S_p)$$

where Name is the method name; Arg_i is an argument (input parameters) and T_i is the type of Arg_i ((i = 1, k) , k 0); and R_j is a return value and S_j its type ((j = 1, p), p 0).
The domains of Arg_i and R_j may be either a primitive or a non-primitive object.
The domain of each method is a cross-product of the domains of its result values:

$$\text{dom}(m) \quad \{ \quad _{i=1,p}\ \text{dom}(R_i)\}$$

The domain of a class c is: $\text{dom}(c) \quad \{ \quad _{i=1,n}\ \text{dom}(a_i)\} \quad \{(\quad _{i=1,j}\ \text{dom}(m_i)\}$
For two classes c_1 and c_2 C:

We call c_1 a *subset* of c_2, denoted as (c_1 c_2), iff: ((o O and o c_1) o c_2) the subset relationship definition is based on object identity, it ignores the type description of the classes.

We call c_1 a *subtype* of c_2, denoted as (c_1 c_2), iff: Properties(c_1) Properties(c_2) and (p Properties(c_2) dom_{c2}(p) dom_{c1}(p))

From the subset and subtype definitions we call c_1 a *subclass* of c_2, denoted by (c_1 is a c_2), iff (c_1 c_2) and (c_1 c_2).

Methods of a class can do several tasks, basically we can classify them as follows:

1. Constructor Methods: create a new instance of a class and (probably) initialise its attributes. Such constructors are available to all classes.
2. Update Methods: alter the state of an object.
3. Access Methods: retrieve and query an object state in the database.
4. Computation Methods: are the operations that access an object state but have no side effects on objects. The reason for accessing the object state is to provide information that is implicit and can be derived by calculating it from the object state. For example, the age() method can get the age of a person from the date_of_birth attribute. Some authors call this type of method application-specific methods [VA97], or derived attributes [MHP99].

While global query and transaction processing in multidatabase systems is concerned with the task of the first three types of methods, we are interested in computation

[2] The type of arguments and value(s) returned could be either primitive or non-primitive.

methods, which are designed to provide services that are able to perform an efficient computation on an object of a class.

3 Sharing Behaviour - The Scope

Behaviour in OODB systems represents a valuable resource of code (services), which can offer additional features for organisations to participate in a federated MDBS. If an MDBS provides the environment to reuse and/or share these services, this saves effort, cost and time as the investment made in developing these behaviours can now benefit the global users. For example, let us consider the classes DOCUMENT and PAPER in DB1 and DB2, respectively (Fig. 1). Assume that both classes are semantically related and ideal for integration by merging into a global class GLOBAL-ARTICLE (say) where its extension is the union of both DOCUMENT and PAPER extensions [MB81], [Mo83], [Mo87], [DFG96a]. Typically, reusing the count-word() method defined in local class PAPER at the global level offers an additional feature to the global users. So they can share this method by applying it to instances derived from the DOCUMENT class in DB1 as well as instances derived from class PAPER in DB2.
From the previous example we can differentiate between two different meanings, *reusing* and *sharing* behaviour at the global level. Reusing behaviour is the ability to use the behaviour defined in a component database class on instances of this class but at

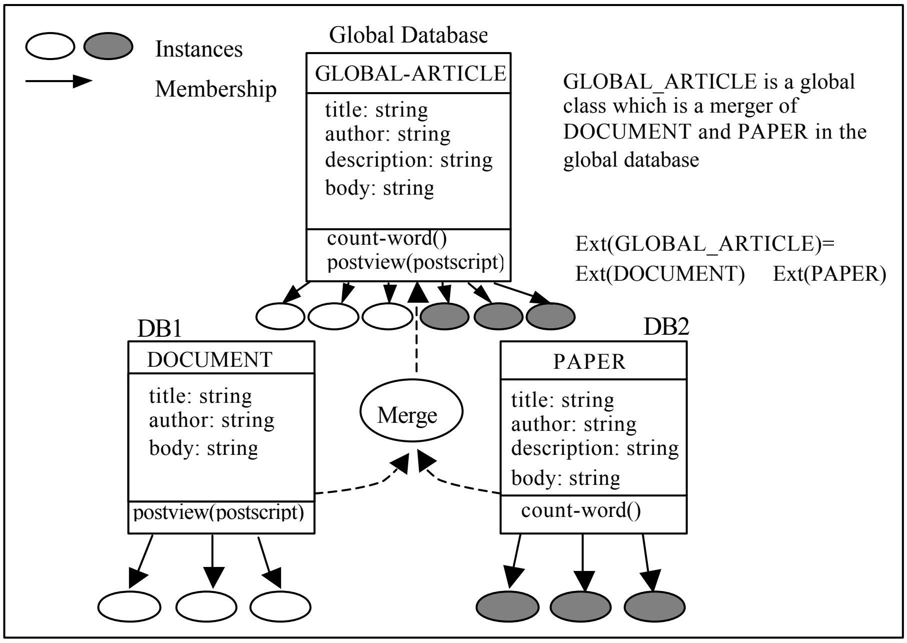

Figure 1: Class PAPER has a count-word method which may be reused at the global level

global schema level. In contrast, sharing behaviour is the ability to use the same behaviour (i.e. behaviour defined in a different component schema) but on instances belonging to other component database classes. For instance, reusing the count-word()

method in global class GLOBAL-ARTICLE occurs when we apply this method to instances derived from the PAPER class, while sharing occurs if we are able to apply this method to instances derived from the DOCUMENT class. The previous scenario is a simple example and we have chosen it to simplify the idea of reusing and sharing behaviour at the global level, where in order to reuse and share methods available in component databases classes, there are certain restrictions and requirements which must be satisfied. These requirements and restrictions are the topic of the next section.

4 Requirements for Reusing and Sharing Behaviour in an MDBS

At the top level we will distinguish between two types of requirements: *systematic* and *semantic* requirements. Systematic requirements deal with the execution of behaviour, while semantic requirements deal with meta-data and identifying the constraints that restrict a global class from reusing a behaviour.

4.1 Systematic Requirements for Reusing and Sharing Behaviour in an MDBS

A principal goal of focusing on behaviour in a federation of database systems is to provide a comprehensive mechanism for the transparent sharing of behaviour. Here we can distinguish between two aspects: first, the remote execution of behaviour; second, the location of the actual information units upon which behaviour operates.

Research in the area of distributed programming languages has addressed issues of the remote execution of behaviour (in the form of operations, methods, or functions) [li88]. The main concern of this area is the issues that relate to the programming of the methods themselves. It is basically tackled by means of suitable language constructs, communication primitives for sending and receiving data, etc.

On the other hand, the *Remote-Exchange* research project [Fa91], [FHM93] focused on the manipulation and location of information units (behaviours, and objects that use these behaviours). The major goal of this project is to provide a mechanism for transparent behaviour sharing. The component that imports a method is called the *local database*, while the component that exports a method (owns this method) is called the *remote database*. This distinction is made to recognise the location of data (object/arguments) provided to a method. At this level of abstraction there are four distinct cases:

local method - local object/argument, (i.e. both the method and its data reside on the same local component).
local method - remote object/argument, (i.e. local method is applied to remote data).
remote method - local object/argument, (i.e. the mirror image of the second case).
remote method - remote object/argument, (i.e. the mirror image of the first case).

The important achievement of Remote-Exchange is that it provides a mechanism for behaviour sharing that makes the location of a method and its input argument transparent

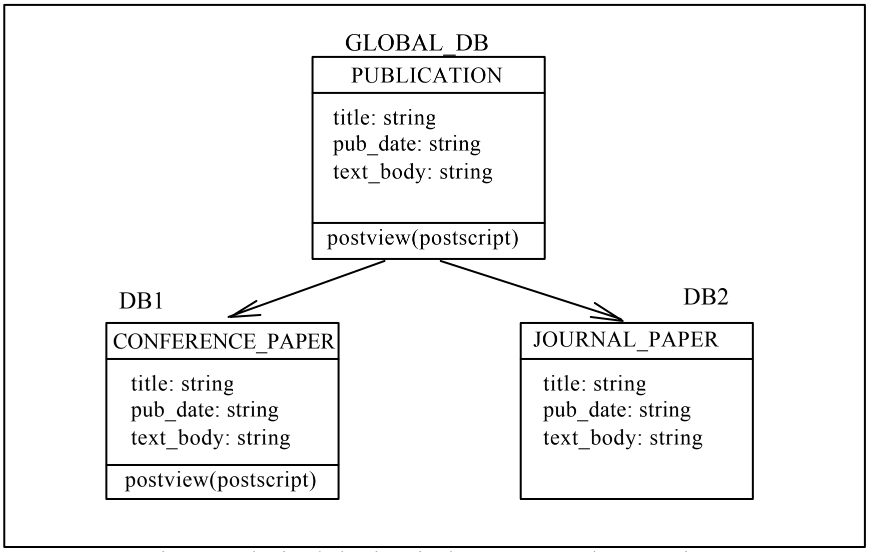

Figure 2: Sharing behaviour in the Remote-Exchange project

to the user. When an object is imported to a local database, a *surrogate* is created for it in the local component. The creation of a surrogate is necessary in order to refer to remote objects using local database system tools without modification. Since these surrogates are created locally, the local system is able to interpret and manipulate remote objects transparently, for example when using them as arguments in a local method call.

The limitation of Remote-Exchange is that no specification other than that of a method's input argument is considered. For example (Fig. 2), the method (postview(postscript)) which prints a postscript document on the screen requires only the postscript argument to decide whether a component database is able to share this method or not.

The previous elements are necessary but they are not sufficient to determine whether a method can be shared or not.

4.2 Semantic Requirements for Reusing and Sharing Behaviour in an MDBS

A component database that participates in a federated MDBS and wishes to share a behaviour available in another component database system requires a semantic relationship existence between the class that owns the behaviour and the class that wishes to share this behaviour (e.g. equivalence, overlap, inclusion). In other words, both classes must be integratable by establishing a global class in the global schema (i.e. this global class is a result of combination, union, etc of both local classes).

In this section, we demonstrate that reusing and sharing behaviour in an MDBS environment implies three basic semantic requirements: *property*, *property assertion validity* and *property value validity* requirements.

4.2.1 Property Requirements

The global class that wishes to reuse or share a method should ensure the properties required by the method in order to perform its functionality.

For example, let us consider the classes STUDENT and EMPLOYEE in DB1 and DB2, respectively (Fig. 3). These classes can be considered as semantically related and depending on the user's need they can be integrated in several ways (i.e. combination, generalisation, union, etc) [MB81], [Mo83], [Mo87], [DFG96b], [NS96]. The method (salary) in EMPLOYEE requires the data represented in its argument (pay_per_hour) and the attribute (hours_worked) in order to perform its functionality, which is calculating the salary of an employee based on how many hours he worked and the pay-rate per hour. The global class PERSON[3] (Fig. 4) must have the attribute (hours_worked) in order to share the salary method defined in the EMPLOYEE class.

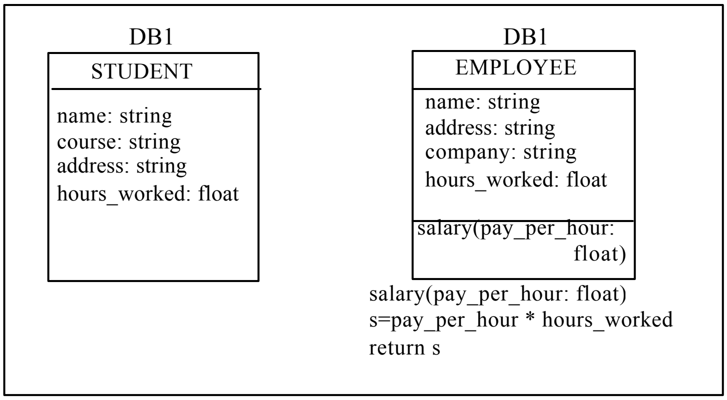

Figure 3: Method property requirements

Formally: for semantically related classes c_1 and c_2 in DB1 and DB2 respectively, with Properties(c_1)={A1,M1}, Properties(c_2)={A2,M2}
Let us have m   M1 a method available in the class c_1 and the global user would like to reuse this method on the global class G_c that is an integration of the classes c_1 and c_2 by applying one of the integration operators[4] (i.e. union, combine, intersect [DFG96a]).
We define the function MethodProperties(m,c)[5] which returns a set of properties (Properties(c*)) from class c that is required by the method m in order to perform its functionality:

MethodProperties(m,c) = Properties(c*) = {A*,M*}   Properties(c)

To use a local method m   M1 on instances of class c_2, it is required that:

MethodProperties(m,c_1)   Properties(c_2) and dom(m)   dom(c_2)

[3] Which is a generalisation of both classes (from Fig. 3) in the global schema.
[4] The extension of the class G_c could be either Ext(c_1)   Ext(c_2); Ext(c_1)   Ext(c_2); Ext(c_1) or Ext(c_2).
[5] This function is very similar to the atts(meth) function introduced by [TS95]

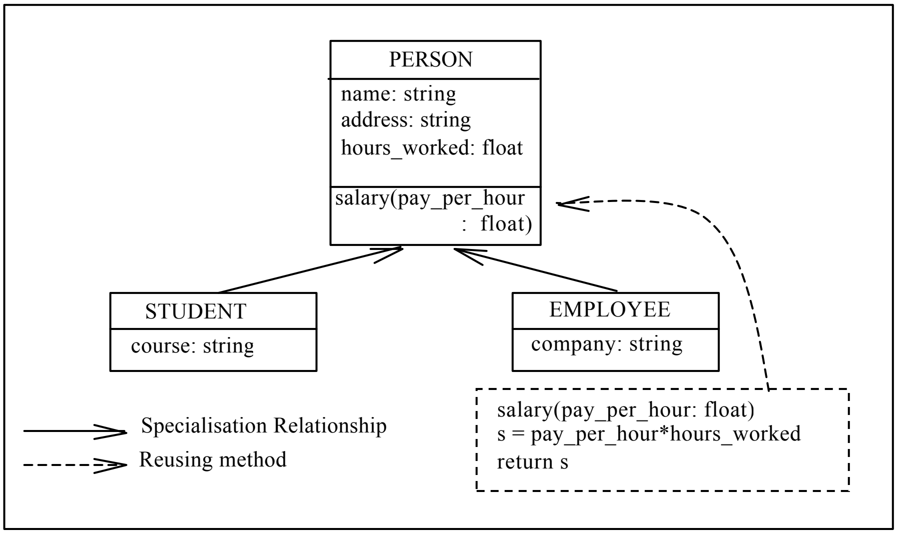

Figure 4: Reusing the salary method at the global level

The previous condition states that the method requires its properties and maintains the domain of the method as well. On the global class G_c which is the integration of both c_1 and c_2 the following condition is required:

MethodProperties(m,c_1)   Properties(G_c) and dom(m)   dom(G_c)

We will use the predicate (c_1.MethodProperties(m,c)?) where the result can be either false or true, to check whether a method m in class c is able to be shared by the class c_1 or not. According to this definition we can describe *Reusing* a method at the global level as reapplying it on instances retrieved from the class on which it was originally defined (class EMPLOYEE in the previous example (Fig. 4)), while *Sharing* a method at the global level means applying this method on instances retrieved from the related class in another database (class STUDENT in the same example).

From the MethodProperties point of view and by assuming that c is an integration of c_1 and c_2, we can formally recognise three cases:

If MethodProperties(m,c)   {Properties(c_1)   Properties(c_2)} and dom(m)   (dom(c_1)   dom(c_2)) this means the method m is applicable on instances retrieved from both class c_1 and class c_2 (whatever the extension of the global class is, Ext(c_1) Ext(c_2) or Ext(c_1) Ext(c_2)), and this corresponds to both Reusing and Sharing at the same time.

If ((MethodProperties(m,c)   Properties(c_1) and dom(m)   dom(c_1)) and (MethodProperties(m,c)   Properties(c_2) or dom(m)   dom(c_2))) this means the method m is applicable only on instances retrieved from class c_1 (Ext(c_1)), and corresponds to Reusing only.

If ((MethodProperties(m,c)   Properties(c_2) and dom(m)   dom(c_2)) and (MethodProperties(m,c)   Properties(c_1) or dom(m)   dom(c_1))) this means the method m is applicable only to instances retrieved from class c_2 (Ext(c_2)), and corresponds to Reusing only.

4.2.2 Property Assertion Validity Requirements

The properties of semantically related classes c_1, c_2 in DB1, DB2, respectively, may require naming conflict resolution or/and domain conflict resolution for integration to successfully take place. The global class G_c that is an integration of c_1 and c_2 can resolve such a naming conflict by renaming, and a domain conflict by creating a function that maps one domain to another [KS91], [SK93], [Ki93], [GSC96]. For example (Fig. 5), the attributes wage and salary in the classes EMPLOYEE in DB1, DB2 are semantically similar, but there are two types of conflict. The first is in name (wage versus salary) and can be resolved by renaming either wage to salary or salary to wage, depending on the global user preference; and the second is in domain, where the wage is represented in dollars and salary is represented in sterling, this can similarly be resolved by converting the value of wage from dollar to sterling or from sterling to dollar. So the global class GLOBAL_EMP which is the union of EMPLOYEE classes in DB1, DB2 uses salary as an attribute name to represent both wage and salary attributes and employs a function that multiplies the wage value by the dollar to sterling rate to represent the wage in sterling. This process is accomplished by the retrieval rules, which is the topic of a future paper.

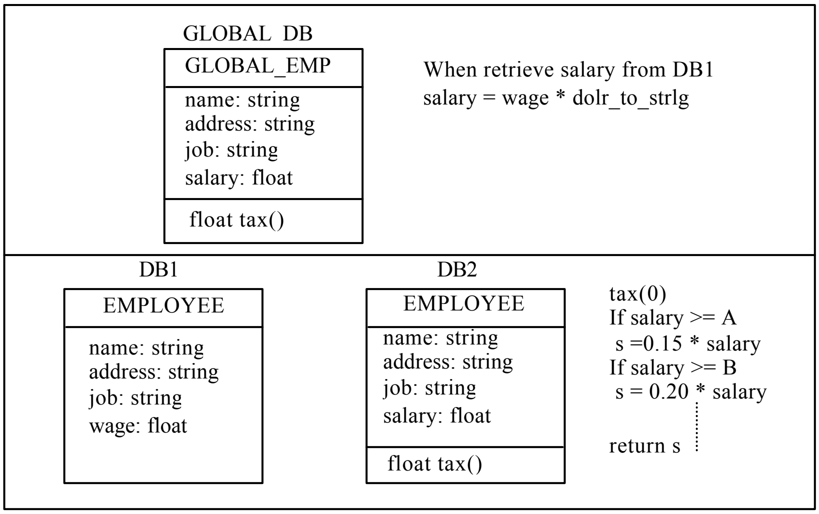

Figure 5: Example of reconciling method properties

For the previous scenario, if the global user decides to reuse a method at the global level, he must reconcile any property conflicts (naming or/and domain) to satisfy the method reuse requirements while he is at the integration confirmation step [BLN86]. As can be seen from the example, the method (tax()) depends on the salary attribute to calculate the tax, so in the confirmation step the user must choose the name salary (instead of wage) to represent both salary and wage properties. Similarly, if he wants to reuse the tax method by applying this method on instances derived from DB1, the salary attribute must be replaced with wage multiplied by the dollar to sterling rate.

Formally: for two related classes c_l and c_2 if p_i Properties(c_1), p_j Properties(c_2) with p_i detected as semantically equal to p_j after one/both of the following processes:

> Structure Conflict Resolution: $p_i = f(p_j)$, where f is the naming conflict resolution function which maps p_j to p_i.
>
> Domain Conflict Resolution: $p_i = g(p_j)$, where g is the domain conflict resolution function which maps the value of p_j to the value of p_i.

If the property p_i is required by a method m which is to be shared (p_i Methodproperties(m,c)), then whenever the method m is called, a reference to the property p_j is replaced by $f(p_j)$ to overcome the naming conflict and the value of p_j is replaced by $g(p_j)$ to overcome the domain conflict; in other words, a reference to the property p_j is replaced by $g(f(p_j))$. For the previous reason if p_i is equal to p_j after $p_i = g(f(p_j))$ or $h^{-1}(y^{-1}(p_i)) = p_j$[6] the user must choose transforming p_j to p_i to satisfy the method property requirement.

4.2.3 Property Value Validity Requirements

The value of a global property P_G in a global class C_G which represents two integrated local semantically related properties could be the result of a function (i.e. sum, average, etc) that is applied to both of them. The value of P_G may violate the semantics of reusing a method m if P_G MethodProperties(m,C_G) at the global level. For example, in Fig. 5 the salary attribute in GLOBAL_EMP at the global level could be determined as the value of the sum of wage and salary attributes from EMPLOYEE classes in DB1 and DB2 respectively. We leave the decision of reusing such a tax method to the user, as he is the only person who can determine whether the value of the salary attribute may violate the semantics of the tax method at the global level and therefore whether it is reusable or not.

5 Conclusions and Further Work

In this paper, we have presented requirements to support reusing and sharing of behaviours in a federation of object oriented database systems. We considered both systematic and semantic requirements. A formal definition of semantic requirements for such reusing and sharing was given. This has the overall benefit of allowing the global user to re-exploit at the global level the investment (effort and expense) made in developing the local component database behaviours. We also explained formally the difference between sharing and reusing behaviour at the global level.
This fundamental work is part of our current research [MGF01a], [MGF01b] into building an integration tool to assist users in creating multiple views supported by multiple behaviours in a heterogeneous object oriented multidatabase system. A *Multiple Views supported by Multiple Behaviours System* (MVMBS)[7] is under construction. MVMBS offers the potential for users to work in terms of integrated and customised

[6] Where y is a structure resolution function and h is a domain resolution function that maps p_i to p_j.
[7] The architecture and operation of MVMBS is the topic of a future paper.

global views supported by multiple behaviours where each global view is designed specifically to meet the needs of a particular user group or application. Our goals for MVMBS include flexibility and customisability, to this end we are developing a semantically-rich integration operator language at the heart of the system.

Bibliography

[AB91] Abiteboul, S.; Bonner, A.: Objects and Views. ACM SIGMOD, Denver, Colorado, May 1991; 20(22), pp. 238-247.

[BLN86] Batini, C.; Lenzerini, M.; Navathe, B.: A comparative analysis of methodologies for database schema integration. ACM Computing Surveys, December 1986, 18(4); pp. 323-364.

[CAG98] Castano, S.; Antonellis, V. De; Gugini, M. G.: Conceptual schema analysis: Techniques and applications. ACM Transactions on Database Systems, September 1998; 23, pp.286-333.

[Ca97] Cattell, R. G. G.; editors: Object Database Standard: ODMG 2.0. Morgan Kaufmann, 1997.

[CHS91] Collet, C.; Huhns, M. N.; Shen, W.: Resource integration using a large knowledge base in carnot. IEEE Computer, December 1991, pp. 55-62.

[DH84] Dayal, U.; Hwang, H.: View definition and generalization for database integration in multidatabase systems. IEEE Transactions on Software Engineering, November 1984; SE-10(6), pp. 628-645.

[Dr93] Drew, P.; King, R.; McLeod, D.; Rusinkiewicz, M.; Silberschatz, A.: Report of the workshop on semantic heterogeneity and interoperation in multidatabase systems. ACM SIGMOD RECORD, September 1993, 22(3).

[DFG96a] Duwairi, R. M.; Fiddian, N. J.; Gray, W. A.: A flexible integration framework for supporting user requirement changes in multidatabase environments. Proc. International Symposium on Cooperative Database Systems for Advanced Applications (CODAS), Kyoto, Japan, 1996.

[DFG96b] Duwairi, R. M.; Fiddian, N. J.; Gray, W. A.: A multiple view definition system for supporting interoperability among heterogeneous and autonomous databases. Proc. 10th ERCIM Workshop on Heterogeneous Information Management, Prague, Czech Republic, 1996.

[FHM93] Fang, D.; Hammer, J.; Mcleod, D.: An approach to behaviour sharing in a federated database system. In M. T. Ozsu, U. Dayal, and P. Valduriez, editors, Distributed Object Management. Morgan Kaufmann, 1993.

[Fa91] Fang, D.; Hammer, J.; Mcleod, D.; Si, A.: Remote-Exchange: An approach to controlled sharing among autonomous heterogeneous database systems. Proc. IEEE Spring Compcon, San Francisco, USA, February 1991.

[Fa88] Fankhauser, P.; Litwin, W.; Neuhold, E. J.; Schrefl, M.: Global definition and multidatabase language approaches to database integration. EUTECO Proceedings, Research into Networks and Distributed Applications, April 1998, pp. 1069-1082.

[GSC96] Garcia-Solaco, M.; Saltor, F.; Castellanos, M.: Semantic heterogeneity in multidatabase systems. In Omran A. Bukhres and Ahmed K. Elmagarmid, editors, Object-Oriented Multidatabase Systems, A Solution for Advanced Applications. Prentice-Hall, 1996.

[GL98] Gingras, F.; Lakshmanan, L. V. S.: nd-sel: A multi-dimensional language for interoperability and olap. Proc. 24th International Conference on Very Large Databases, 1998, pp. 134-145.

[HM93] Hammer, J.; Mcleod, D.: An approach to resolving semantic heterogeneity in a federation of autonomous, heterogeneous database systems. International Journal of Intelligent & Cooperative Information Systems, World Scientific; 1993; 2(1), pp.51-83.

[KS91] Kim, W.; Seo, J.: Classifying schematic and data heterogeneity in multidatabase systems. IEEE Computer, December 1991, 24(12), pp. 12-18.

[Ki93] Kim, W.; Choi, I.; Gala, S.; Scheevel, M.: On resolving schematic heterogeneity in multidatabase systems. Distributed and Parallel Databases, 1993; 1, pp. 251-279.

[Li88] Liskov, B.: Distributed programming in argus. Communications of the ACM, 1988, 31(3), pp. 300-312.

[Li93] Litwin, W.: O*sql: A language for object oriented multidatabase interoperability. In D. K. Hsiao, E. J. Neuhold, and R. Sacks-Davis, editors, Interoperable Database Systems (DS-5) (A-25). North Holland, 1993.

[MHP99] McFadden, F. R.; Hoffer, J. A.; Prescott, M. B.: Modern Database Management. Addison Wesley, 1999.

[MB81] Motro, A.; Buneman, P.: Constructing superviews. Proc. ACM SIGMOD International Conference on Management of Data, 1981, pp. 56-64.

[MGF01a] Mourad, M. B.; Gray, W. A.; Fiddian, N. J.: Detecting Object Semantic Similarity by Using Structural and Behavioural Semantics. To Appear in Proc. 5th World Multi-Conference on Systemics, Cybernetics and Informatics, Orlando, USA, July 2001.

[MGF01b] Mourad, M. B.; Gray, W. A.; Fiddian, N. J.: Guiding Object-Oriented Schema Integration by Using Structural and Behavioural Specification. To Appear in Proc. 13th International Conference on Systems Research, Informatics & Cybernetics, Baden-Baden, Germany, July 2001.

[Mo83] Motro, A.: Interrogating superviews. Proc. 2nd International Conference on Databases (ICOD-2), Cambridge, England, 1983, pp. 107-126.

[Mo87] Motro, A.: Superviews: Virtual integration of multiple databases. IEEE Transactions on Software Engineering, 1987, 13(7), pp. 785-798.

[NS96] Navathe, S.; Savasere, A.: A schema integration facility using the object-oriented data model. In O. A. Bukhres and A. K. Elmagarmid, editors, Object-Oriented Multidatabase Systems: A Solution for Advanced Application. Prentice Hall, 1996.

[PTR96] Papazoglou, M. P.; Tari, Z.; Rusell, N.: Object-oriented technology for inter-schema and language mapping. In O. A. Bukhres and A. K. Elmagarmid, editors, Object-Oriented Multidatabase Systems: A Solution for Advanced Applications. Prentice Hall, 1996.

[QFG92] Qutaishat, M. A.; Fiddian, N. J.; Gray, W. A.: A schema meta-integration system for a heterogeneous object-oriented database environment – objectives and overview. Proc. NordData'92 Conference, Tampere, Finland, 1992, pp. 74-92.

[SK93] Sheth, A.; Kashyap, V.: So far (schematically) yet so near (semantically). In D. K. Hsiao, E. J. Neuhold, and R. S. Davis, editors, Interoperable Database Systems (DS-5). North-Holland, 1993.

[SL90] Sheth, A.; Larson, J. A.: Federated database systems for managing distributed heterogeneous and autonomous databases. ACM Computing Surveys, September 1990, pp. 183-236.

[SSU90] Silberschatz, A.; Stonebraker, M.; Ullman, J. D.: Database systems: Achievements and opportunities. ACM SIGMOD RECORD, December 1990, 19(4), pp. 23-31.

[TS95] Thieme, C.; Siebes, A: Guiding schema integration by behavioural information. Information Systems, 1995, 20(4), pp. 305-316.

[VA97] Vermeer, M. W. W.; Apers, P. M. G.: Behaviour specification in database integration. Proc. Conference on Advanced Information System Engineering (CAISE97), Barcelona, Spain, June 1997.