A Business Process Driven Multidatabase Integration Methodology

Ralf M. Mühlberger and Maria E. Orlowska
Department of Computer Science
The University of Queensland
Qld 4072 Australia
ralf@cs.uq.oz.au, maria@cs.uq.oz.au

Abstract Development of multidatabase systems requires a careful understanding of the data requirements and of the data interaction between the participant sites. Total integration of the data is not feasible, due to such reasons as the dynamics of today’s information requirements. We therefore put forward that a multidatabase system must be built on a partial integration. In this paper we outline a methodology that draws on business re-engineering principles to guide the developer through the process of creating partially integrated schemata. This methodology also gives us added benefits by specifying integration requirements and cost factors, and thus limiting/weighing the options given to the developer during the integration process.

1. Introduction

As companies evolve to meet their growing competitive needs, so do the requirements they place on their information systems. Legacy systems need to be updated to newer, more efficient platforms; more functions need to be automated; data in different systems must be correlated and integrated to produce more desperately needed leverage; interoperability is the key word as new business structures evolve from the computing power now available.

Underlying all these changes is the information that the business processes work on and underlying the processes of migrating and creating integrated database systems is the evolution of the data objects. Inter database dependencies need to be identified, specified and enforced. Data scrubbing of existing information must bring it up-to-date with the company’s business rules.

Schema integration has a history of research spanning back more than 20 years. The aim has always been to produce a global schema, allowing ad hoc queries and complete re-implementation of the old systems. The complexity of this task has made any realistic results few and far between, however, mainly due to the labourious effort required in identifying and consolidating data objects. A partial integration is the only method by which feasible results may be gained, particularly when the integration/migration is done in a stepwise fashion.

The methodology outlined in this paper can be used for integration in multidatabase system design, enhancing existing applications, building global applications, migrating systems (including applications) to new platforms and architectures as well as creating view definition mappings for the partially integrated schemata, as used by most current multidatabase system implementations. Basic assumptions, such as the degree of autonomy that needs to be maintained by local systems and the various cost factors relevant to the target system, can be programmed into the methodology to influence the integration choices that may need to be made throughout. For the focus of the methodology we draw on the re-engineering principles of business process modelling, such as the
DEMO business process modelling method outlined in [Die94], where business process workflows are equated to applications in the information system terminology and these applications used to direct the integration, and subsequent growth, of the systems. We also introduce requirements needed for a partial integration to be acceptable, and show how these are met by our methodology.

2. A Methodology Framework

In this chapter we give an overview of the application integration methodology, followed by more in-depth descriptions of the individual components, illustrated by a simple example in the next chapter.

1. Modelling Phase

   (a) Data models are prepared for the participating sites.
   (b) The business process is specified as an application data demand model.

2. Integration Phase

   (a) Objects corresponding to those in the application data demand model are identified in the participant site models, based on the roles they play within the application model’s relationships.
   (b) Inconsistencies between the application data demand model and local systems, such as objects that can not be identified in the participant sites, are resolved.
   (c) A subschema expansion is calculated for each local, participant site schema.
   (d) The identified and correlated objects are integrated across the systems.

   The integration phase is repeated until no more expansion is calculated, and no more integration pairs need to be resolved.

3. Extraction Phase

   The objects involved in the previous steps are separated from the rest of the participant sites schemata, leaving an integrated application schema for the target application.

2.1 System Assumptions

When creating a theoretical approach that maps onto an engineering problem, too often basic limitations are not covered sufficiently and thus the theory promises more than it may be able to deliver. In integration and migration, many decisions need to be made regarding modification of and addition to existing database systems. Management and maintenance issues may make some of the possible modifications required not possible. The general cases of the modifications, as far as can be determined, can be filtered taking into account these basic assumptions, and thus further driving the integration process by requirements. Thus options such as degree of autonomy or emphasis on minimising modifications in favour of standardising the system definitions may be used to give cost values to different options faced by the designer working through the partial integration process.

2.2 Application Data Demand Model

The business process, with respect to which we focus our integration, are modelled in an application data demand modelling language. This is essentially an information model with some extra data not normally stored in data models. We don’t produce a purely behavioural model, such as CPM, but rather concentrate primarily on the data requirements of the application. Temporal constraints
governing the order in which operations are performed in an application can be illustrated in a workflow diagram, with the application broken down into a set of transactions \(^1\), which are used as the nodes of the workflow diagram.

However, for determining the extent of integration we must consider all possible cases of execution, and hence are not concerned with order. We are interested in refining our model to include both location and operation information about the roles played by the objects. The operation details can be used in limiting the degree of integration with respect to the definition of inter database dependencies and thus reducing the cost both of the integration process and of the transaction processing as less inter database dependencies need to be checked. We consider the following operations,

**read** Read a data item playing a role.

**insert** Insert a new, or existing, data item into a role. Where the role is mandatory to the object, any insert operation always refers to a new instance.

**delete** Delete an existing data item from a role played by it.

**modify** Modify an existing data item playing a role.

**replace** Replace a data object. (insert; delete) Whereas the **modify** refers to the same object, **replace** refers to a new object that supercedes the previous object in a role. The old object is thus deleted and a new object inserted.

accepting that both modify and replace can be implemented using delete and insert. In practice however, it becomes simpler to abstract these functions and deal with them as separate operations.

The locations of data objects are primarily used in the identification phase of the methodology, given the assumption that during the business process modelling the possible locations of data is often self-evident, while it may not be as intuitive when searching the schemata for their existance.

These extensions can then be mapped onto any existing information modelling language (e.g. NIAM, EER, etc), and thus be used in the conceptual integration process. For an example of an application data demand model, see figure 3 illustrating the travel agent business process.

### 2.3 Subschema Expansion

An operation is performed on a role, that is played by an object and that may have some constraints enforced on it. For each operation, we consult the type of role that is involved, and see what expansion is required, expressed as operations on other roles within the schema. The types of roles are defined by the constraints on the roles, such as a mandatory role, an optional role, a role spanned by an uniqueness constraint or a role that is a part of an uniqueness constraint. Some rules may also affect all roles, regardless of the constraints involved, and are thus specified as acting on all roles. This expansion may itself cause further expansion, so the algorithm must be repeated until no more operations are marked. Note that as this algorithm is used in application integration, the result of an increase in the subschema necessarily implies a possible increase of the application. For a description of the operations see section 2.2. A sample list of expansion rules for an ORM constraints specification is given in table 1.

\(^1\)We define an application as a set of transactions, not all of which need to commit, nor even be executed, for the application to be run successfully. A transaction is taken to be a set of operations, all of which must complete successfully for a transaction to be able to commit, else a complete rollback must be performed.
<table>
<thead>
<tr>
<th>Operations</th>
<th>Roles</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete</td>
<td>Mandatory</td>
<td>Delete all other roles played by object.</td>
</tr>
<tr>
<td>Delete</td>
<td>Optional</td>
<td>No expansion required.</td>
</tr>
<tr>
<td>Insert</td>
<td>Mandatory</td>
<td>Insert into all other mandatory roles played by object.</td>
</tr>
<tr>
<td>Insert Existing</td>
<td>Optional</td>
<td>No expansion required.</td>
</tr>
<tr>
<td>Insert New</td>
<td>Optional</td>
<td>Insert into all mandatory roles played by object.</td>
</tr>
<tr>
<td>Modify</td>
<td>All</td>
<td>Modify all other roles played by object.</td>
</tr>
<tr>
<td>Read</td>
<td>All</td>
<td>No expansion required.</td>
</tr>
<tr>
<td>Replace</td>
<td>Mandatory</td>
<td>Insert into all other mandatory roles played by object.</td>
</tr>
<tr>
<td>Replace with</td>
<td>Optional</td>
<td>No expansion required.</td>
</tr>
<tr>
<td>Replace with New</td>
<td>Optional</td>
<td>Insert into all mandatory roles played by object.</td>
</tr>
</tbody>
</table>

Table 1: Some of the expansion rules based on an ORM constraints specification.

2.4 Integration

As this is a methodology primarily for defining the extent of integration, with integration in the classical sense, i.e. identifying and resolving conflicts between objects, incorporated as one of the steps, the representation of the problems faced in integration may seem naive. We do recognize the extent of these problems, but we also realize that most of the work done in schema integration has concerned itself with these problems, and can be applied within our framework.

The integration will be required to deal with previously identified objects, that are tabulated in the previous steps. For any particular integration case required, we can then draw on any of the multitude of methods developed previously, such as [Urb91, Gel91, Spa91, Kim91].

For the different choices of modifications that may be required for implementation of an application, lists can be given throughout to ease the integration. The choices may depend on assumptions made before the integration regarding the target systems (i.e. platform/architecture in the case of migration, existing systems in the case of multibase systems), as described in section 2.1. This may mean that a modification, required to provide an application with its requested functionality, may not be possible due to some basic assumption, such as the degree of autonomy for the participant sites. Although still in the theoretical list of options, this modification then may not be implemented. It may also, however, be given as a choice, with the possibility of changing an underlying assumption, given that the benefits outweigh the requirements for which this assumption was made in the first place. This then becomes a management issue, affected by various cost factors, which the methodology supports through its options mechanism.

2.5 Integrated Application Schema

After the integration process, we end up with an integrated application schema, which we can then use for further work during the design of a multidatabase system. This includes:

- detecting possible inter-application conflict by comparing their integrated application schemata
- generating some of the application code
- producing view definitions of the export schemata, as used in most current multidatabase system implementation platforms
Figure 1: Part of the conceptual schema for the airline system.

The integrated application schema also has the application specific annotations (i.e. location and operations). If different information models were used for the participant site schemata, the integrated application schema can either be left heterogeneous or mapped onto a canonical data model during the integration process.

3. Example of the Methodology

To illustrate the various phases of the methodology, we give a simple example integration in this section.

3.1 Modelling Phase

We map the modelling requirements onto the conceptual information modelling languages ORM (Object Role Modelling) and ER (Entity Relationship model). The application data demand model used is simply an adaptation of ORM. For a more indepth explanation of NIAM (the dialect of ORM used in this paper), the reader is referred to [Hal89].

Participant Site Schemata

The airline uses its database systems to track flights, destinations, reservations, planes, pilots and other related air travel information. Figure 1 shows the conceptual schema of the airline system.

Figure 2 shows the conceptual schema of the travel agent database. This covers the data required for the travel agent, such as customers, holidays planned for the customers, including flights and hotel rooms booked, destinations to be visited and information about these destinations.

Application Data Demand Model

This is a sample application for which we wish to produce the partial integration. Here we have an application in which the travel agent can plan a customers holiday, including the actual booking of flights with an airline with which the travel agent is affiliated. Figure 3 shows the application data demand model for this application.
3.2 Integration Phase

Object Identification

In the object identification, we make a list of the roles from the application data demand model, and find the corresponding roles in the participant site schemata. We then match the objects playing these roles for the integration. Figure 4 lists the roles for integration, and figure 5 lists some of the objects required for the integration phase.

Inconsistency Resolution

In this example, there is no object inconsistency, i.e. there are no objects required by the application that are not stored at some participant site.

Subschema Expansion

We run the subschema expansion algorithm on the objects marked by the previous steps, resulting in a larger list of objects to be integrated. Figure 6 shows the expansion on one of the schemata, and figure 7 shows the increased object list for integration.

Object Integration

For the integration, we define some rule for each object pair from the identification phase, resolving any conflicts. The implementation of this integration may involve data scrubbing, modifications to the participant site or implementation of a translation system. Figure 8 gives some of the objects and their integration rules.
Figure 4: Roles Identification.
3.3 Extraction Phase

Finally, we extract the integrated application schema from the working diagram, leaving the heterogeneous data modelling constructs as they are (figure 9), or alternately mapping it to a cannonical data model.

4. Completeness of Methodology

As outlined in the introduction, we do not accept the idea of a global integration. Partial integration is the only method by which we can feasibly integrate any set of realistic systems. For us to accept a partial integration methodology, however, it must satisfy two constraints:

1. Guarantee of Functionality
2. Integrity Maintenance

To prove that an integration will provide the functionality required for an application, we first need to propose a measure of functionality, namely the availability of the data required for the operations. So if an application needs to allow the user to write a value into an object, then for the integrated schema to supply the application with sufficient functionality it must include that object and, if such a distinction can be made in the data model used, allow the object to be written to.

More importantly however, by basing the integration on the application specification, and by not proceeding with the integration until all objects required for the application have been identified at the participating sites, we ensure that when the integration is completed, the systems have been integrated sufficiently for the requirements of the application. Hence we can say that the integration complies with our requirement that the functionality of the application be provided. A partial integration based on export schemata given to the builders of the multibase database system may also not be adequate if it does not take into account what function the system is to provide. In [Sem94], the mistake of not using the business structure of the processes to determine implementation issues is outlined with respect to the growing field of client/server application development.

To ensure sufficient integrity begin maintained by the application relies on enforcing the constraints defined for the local sites. Through the subschema expansion step we include all data objects in the integration that need to be included in the application, based on the constraints specified at the local sites and the operations that the application will perform during it’s transaction flow. Due to the complex and dynamic nature of constraints we can never claim completeness with respect to our subschema expansion rule table. We can however list fundamental constraints, for example internal and interpedicate uniqueness, exclusion, involvement, ring, subset, equality and frequency constraints, and develop the rules for these. We can cover even more cases and thus
Figure 9: The Integrated Application Schema, using heterogeneous data modelling constructs and showing the interdatabase dependencies.
producing a usable, if not complete, rule set using general classifications such as for example those given in [Kim91].

Thus we can ensure the integrity of data (across the multidatabase system) is not compromised by an application. This approach will not ensure global consistency of all data, as would be required for ad-hoc queries of the system, but is usable for systems where the data manipulation is driven by applications, such as can be found in most large commercial systems today, and these applications have been processed using the methodology.

5. Evolving a Multidatabase System

Theoretically, a multidatabase system can evolve in two directions, application and site. In both cases, the evolution may occur in one of three ways:

1. adding of new application/site
2. deleting existing application/site
3. modifying existing application/site

The way in which we deal with these growth factors lies mainly in a methodical application of the earlier work, and is outlined in the following sections.

5.1 Evolution of Applications

Growth of applications is the most common factor in multidatabase system evolution. This is also the method by which we, incrementally, build the original multidatabase system design. We create the integrated application schema for the first application, and then add more applications using the approach listed below.

Adding an Application

For subsequent applications the modelling phase only requires the application data demand model for the next application, i.e. business process, to be made. We must use the previously designed local, participant, site schema. Next we go through the partial integration method as before, but we now include an interapplication conflict detection step, in which we ensure that the new application will not

- require modifications in the participant schemata that would prevent execution of previous applications, nor
- implement inter database dependencies that will conflict with constraints from the other applications.

The conflict detection is a process by which we compare the common elements of the integrated application schemata. The elements that need to be compared can be automatically listed due to the integration and some of the comparisons can be automated (such as in the case of well known constraints).

Deleting an Application

When deleting an application, several constraints may be freed up, thus lowering the cost of the multidatabase system on the local sites. The constraints that may be deleted can be derived from the difference between the sets of constraints required for the application and all other applications. Let $C_i$ be the set of constraints for application 1, and let $C_k$ be the set of constraints for all applications except application 1. To find the removable interdatabase dependencies, calculate the set difference between $C_i$ and $C_k$. 
Modifying an Application

There are two methods to deal with modification of an application:

1. Deal with all modifications as they are to be made, revising the application data demand model, re-checking the partial integration and comparing the modifications with the other applications for conflict.

2. Treat the modification as a deletion of the old application, followed by an addition of the new application. This may be more effective for large modifications, particularly as we can still learn from the previous applications integration steps, thus making the new integration simpler.

5.2 Evolution of Sites

Although site growth will be much less frequent than application growth, for the sake of completeness we include the outline of the algorithm by which this is handled. Adding new sites into an multidatabase system is now a well defined process, that involves the processing of new applications as they apply to the interaction between the existing multidatabase system and the newly added site, as well as the revision of any existing applications that may be affected by the new site (a process semi-automatable, where the only user-affected decision that needs to be made is whether the new site may need to be included in an existing application, hence making that application a ‘new’ application in some ways.) Application conflict, as already shown, can then be detected automatically for the known constraints and where unknown constraints are involved, the system can highlight them as well as learn from them.

Thus the method allows for easy integration of new sites and applications into existing multidatabase systems, requiring much less modification to either the multidatabase system or the new site than in the case of either a global integration approach, or a traditional security centered, export schema approach. This of course cuts development/integration time, and thus greatly increases the cost efficiency of multidatabase system development.

Adding a Site

The first step when adding a new site, is to check all existing applications for any enhancements that can be gained from it. This is simply a matter of going through each application with a domain expert. The enhancements can then be noted on the integrated application schema, which is then used as the application data demand model in the methodology. The existing sites have already been identified, resolved and integrated, so the integration process is only required to deal with the integration between the existing systems and the new site.

For any applications created after the inclusion of the new site in the multidatabase system, we simply follow the methodology as before.

Deleting a Site

When deleting a site, we must check which applications are affected (automated due to the object locations stored with the integrated application schema), and then determine whether the application is to be adapted to a more restricted environment or whether it is to be disabled. Restriction may be possible if some of the transactions can be removed from the application or if the storage of data objects still required is moved to one of the remaining sites.

Modifying a Site

Modification requires once again a list of applications involved with the site, followed by a return to the methodology for each of these to ensure the subschema expansion does not now include
more objects. If less objects are included (using a system where the directly affected objects are
differentiated from the objects that were marked in the subschema expansion during the original
processing), then we may also have the choice of disabling some interdatabase dependencies and
thus reducing the transaction cost of the application.

6. CASE Tools

As can be seen from the example, the various components of the methodology can be mapped
onto existing information models, with some extensions for the application data demand model.
The main difficulty of any methodology dealing with high complexity of data is to impose a
thorough, planned approach. However, this lends itself perfectly to CASE tools, where the rule
checking, subschema expansion, exhaustive development and system assumptions filtering can be
easily incorporated into existing technology.

We are currently looking at the initial design of a case tool to work with this methodology,
as well as investigating the possibility of working with different, existing case tools through an
integration interface. Together with existing data scrubbing tools and classical integration tools
we can thus produce an integration suite that can be used for multidatabase design, legacy system
migration and standardization of previously integrated systems.

7. Conclusion

We hold that for most multidatabase system design, total integration is not adequate due to the
time requirement and the dynamic nature of database systems, and also that it is often simply not
needed. We also believe that partial integration needs to fulfill certain requirements with respect to
providing functionality and ensuring consistency, else it may also involve unnecessary work, provide
unsatisfactory for the requirements of the system and cause degradation of the information.

By learning from business re-engineering we give direction to partial integration, and have
thus developed a methodology that meets the requirements for partial integration while minimising
the complexity of the integration by only concentrating on integrating those objects that are
absolutely required.

We show the potential of the method not only as a fast integration tool, particularly when
implemented as a CASE tool, but also in the further stages of the multidatabase system lifecycle,
such as the addition of applications and the expansion of the system to include new participant
sites, as well as in migration and in the implementation through SQL view mappings.

References


[Gel91] James Geller, Yehoshua Perl, Erich Neuhold, and Amit Sheth. Structural schema in-
tegration with full and partial correspondence using the dual model. Technical Report
Submitted to Information Systems.


