Database Cooperation: Classification and Middleware Tools[1]

PAOLO ATZENI, Università di Informatica e Automazione
LUCA CABIBBO, Università di Informatica e Automazione
GIANSALVATORE MECCA, Università della Basilicata

We propose new criteria for the classification of systems for database cooperation, based on the nature of the component databases. In fact, the traditional criteria — heterogeneity, distribution, and autonomy — are often constraints for the design process, rather than design parameters. In this case, other features of the component databases should be addressed. We claim that a more useful classification can be based on three new criteria: (a) degree of transparency, (b) complexity of operations, and (c) level of liveliness of data. This leads us to distinguish three main categories of systems: (i) multidatabases, (ii) data warehouses, (iii) local information systems with external data. For each of these categories we discuss implementations based on tools offered by currently available technology.

INTRODUCTION

The recent growth of computer networks, in terms of both technology, methodology and actual deployment, has created enormous expectations in many users. The achievement of a smooth interaction of the various components within the organization and the development of a simplified, uniform interface offered to the outside world can be considered major long-term goals of an enterprise-wide network (Kim, 1995). In this perspective, data should circulate easily, without need for replication nor for reentry.

In most cases, networks have appeared late in a picture that already included many application systems, developed independently from one another. As a natural consequence, it is expected that these applications cooperate, possibly within a larger effort to reengineer not only the information system but also the business processes of the enterprise. The need for establishing cooperation among these systems arises also for other motivations. First, it is common now to buy specific applications, for example to manage accounting or personnel data, from specialized vendors; these independent applications have then to be integrated in "the" enterprise information system. Second, companies now often merge or split, and

existing systems have to evolve accordingly. Finally, there are systems that have existed for years, such as airline reservation systems, that by their own nature need to interact with information systems from different companies. Specifically, the discussion in this paper is the synthesis of a contribution to the development of applications over a national network being deployed in Italy, to connect all the offices in the civil administration.

It is important to clarify that the interaction among different systems over a network may occur at various levels. The simplest is *connectivity*, which is realized when systems and networks are linked together in some way and so allowed to exchange information; this is for example the case of any system connected to the Internet using the TCP/IP protocol. A more complex form of interaction is *interoperability*, in which systems and networks interact by means of standard services. In Internet, we have standard protocols above TCP/IP, such as file transfer (with the file transfer protocol, ftp), virtual terminal (telnet), electronic mail (X.400 or ESMTP/MIME), directory service (X.500), the World Wide Web (http). In this case, the interaction between different systems is limited to standard services offered on top of the connectiv-

ity environment. Finally, the highest form of interaction is *cooperation*, the only one that gives actual benefit to the final user in a transparent way; in this case, applications over different systems interact with one another, and, at the extreme level, integrated, distributed applications coordinate existing local applications.

The cooperation of information systems requires that the participant systems offer services, and happens when systems make use of services offered by other systems. A major feature of most cooperative information systems is that their component systems have to serve two different sets of goals: those related to the specific task they have originally been designed for and those shared with the other participants in the cooperation. Component systems are subject to evolution for technical or organizational reasons, and the same could happen to the requirements of the overall cooperative systems. Therefore, it has been observed that a fundamental issue in the study of cooperative information systems is the "management of change" (De Michelis et al., 1998). It turns out that most of the issues related to this framework also arise when one considers migration of information systems: if a conservative, gradual approach is taken (and this is usually the only reasonable choice (Brodie and Stonebraker, 1995), except for a few extreme cases), then intermediate phases in the migration activities would require cooperation among components. Moreover, since migration is almost a neverending activity, as the steady state is seldom reached, it could often be the case that complex systems need to be cooperative at all times.

Cooperation can be studied in various ways. From a rather standard information-system point of view, we believe that it could be important to distinguish two main forms of cooperation:[2]

- *data-oriented* cooperation, in which data in a system is visible and/or accessible to other systems;
- *process-oriented* cooperation, in which systems offer services, exchange messages (or data, documents) and trigger activities.

Practical systems are usually based on both kinds of cooperation. However, it is very often the case that one of the two aspects — data sharing or process sharing — represents a prevalent requirement for the final system. Moreover, for methodological reasons, we tend to study data-oriented features and process-oriented features in a rather independent way. As a matter of fact, in our study for the Italian public administration, we found the major instances of potential cooperative systems to be placed in one of the two areas. For example, all the systems related to interoffice payments fall in the process-centered cooperation, whereas integrated civil service register or census systems fall in the data centered cooperation.

In this paper we explicitly consider the issue of data-oriented cooperation, studying features and requirements of applications that need to share data. As an ideal goal,

one could think that database cooperation should aim at offering to the user a fully-fledged distributed database: the whole patrimony of relevant data is shown as if it were stored in a unique, possibly distributed database, with a complete transparency with respect to location, ownership, and with continuous access to non-replicated operational data allowed to all authorized users. However, current technology falls short from supporting such a situation or, at best, can support it only at very high costs, especially if performance, availability, and reliability are important. As a consequence, the design of a system for database cooperation requires an in-depth evaluation of costs and benefits.

The main contribution of the paper consists in a set of criteria for classifying data-oriented cooperative applications, which help to identify how close we should go to the ideal "distributed-database goal". The paper is organized as follows. First, we discuss the different needs from which cooperation may arise, and propose a set of classification criteria that go beyond the usual degrees of heterogeneity, autonomy, and distribution. Based on these criteria, we develop a new classification of data-oriented cooperative systems, which incorporates traditional architectures, such as federated databases or data warehouses. We then briefly discuss how existing middleware tools can be used as a basis for the implementation of the main categories of cooperative systems. Finally, we conclude with a discussion that relates cooperation with reengineering and migration.

CLASSIFICATION CRITERIA

Cooperation means that we have multiple systems; data cooperation means that we have multiple databases that handle data. Traditional criteria for classifying cooperative systems, in the context of distributed and federated databases (Sheth and Larson, 1990), refer to the level of: (i) *distribution*; (ii) *heterogeneity* ; and (iii) *autonomy* of the component databases. The level of distribution is a measure of how the various resources, and especially data, are spread over the system. Distribution may range from different databases on a same machine to databases spread over a geographic network. Note that, as opposed to usual distributed databases, here distribution is not a design process, but a fact, due to the preexistence of the cooperating databases.

Heterogeneity arises in many different aspects. There can be differences in the computing environments (hardware, operating system, network software). The database management systems involved may differ in the data model (e.g., relational, hierarchical, object-oriented, file-based), in details in the same data model (e.g., incomparable versions of the relational model, with different data types and available constraints), in the language (e.g., different dialects of SQL). There can be semantic heterogeneity, due to differences in the meaning of data. Autonomy is the absence of a common (and

coordinated) control over the various systems. The level of autonomy measures how much the component databases will preserve their own structure and will be able to satisfy local requests while they concur to the cooperative system. Autonomy arises in different aspects. Design autonomy occurs if the various systems are built independently, with different choices (thus inducing heterogeneity). Service autonomy corresponds to decisions on if and how cooperation is established (what services are offered). Execution autonomy occurs if cooperative operations are executed under local control, thus preventing cooperation to interfere with "private" operations.

These three criteria were mainly proposed (Sheth and Larson, 1990) with respect to contexts in which the primary goal is to establish an integrated system, and a central institution can influence directly the design of the peripheral information systems. However, they cannot be considered as classification criteria in our context, since in the design of cooperative information systems we should be ready to tackle highly distributed, heterogeneous, and autonomous systems. In the worst case, these are constraints on the design process. In other cases, the presence of some coordinating authority might take advantage of cooperation as a stimulus for reengineering, possibly reducing the degrees of distribution, heterogeneity, and autonomy, if they correspond to "deficiencies" to overcome.

These considerations suggest to adopt, with respect to the design of cooperative systems design, other criteria that allow to better characterize the nature of cooperation needs. The criteria we propose are:
- the degree of transparency of component data;
- the complexity of operations;
- the level of liveliness (or up-to-dateness, as opposed to obsolescence or latency) of data.

We discuss them in turn in the following sections.

Transparency

The *degree of transparency* in accessing distributed data refers to the need for hiding distribution and heterogeneity of component systems in a data-oriented cooperation. In other words, it tries to indicate how much the cooperative system must appear to global users as an integrated database.

Based on this criterion, we can recognize some interesting classes of systems:
- at the highest level of transparency, there are those systems that integrate the component databases; the cooperative system offers an integrated interface to the cooperative application, which sees one single (virtual) database, having an integrated schema (either a database schema or a set of functions);
- at the other extreme, each component database offers a set of services, and distribution and heterogeneity are not masked at all; in this case, each cooperative application is responsible for accessing, integrating, and transforming

the various pieces of data.

Clearly, the design and implementation of a single, integrated interface requires some complex and delicate activities, related to the *translation* and *integration* of the individual schemes (Batini et al., 1986) and instances. However, despite its complexity, this activity has the virtue of resolving, once and for all, the possible data conflicts due to different schemes or different semantics, whereas, in a low transparency system, these problems must be resolved each time a different cooperation need arises.

Complexity of Operations

The *complexity of operations* refers to the need for coordination in the execution of operations, that is, queries and transactions. In other words, it is the level of synchronization needed to perform queries and updates on the component databases. It is well known that the proper management of distributed transactions (Gray and Reuter, 1993) in a heterogeneous environment has important implications on data integrity and reliability (Garcia Molina and Hsu, 1995), so that the complexity of these transactions constitutes an important classification criterion. In fact:
- complex queries (involving join of large relations from different databases) or transactions (with multiple updates in different databases) require the development of sophisticated components to guarantee efficiency and correctness;
- on the other side, simple transactions (for example, involving read-only data, accessed separately) do not require specific support.
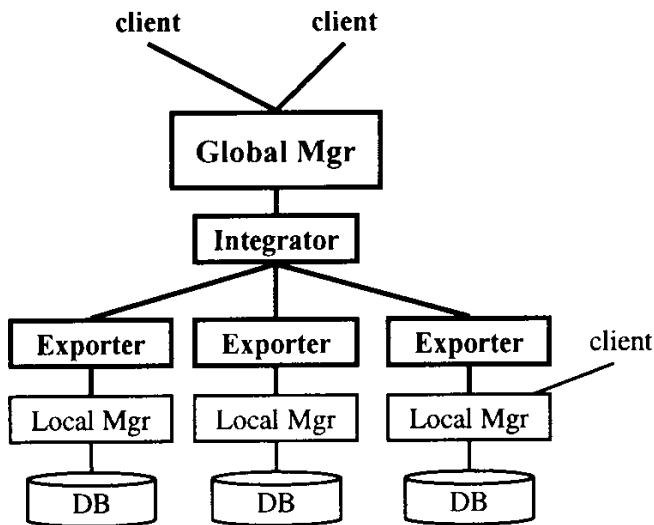
Liveliness of Data

The *liveliness of data* refers to the need for actual availability of current data. Symmetrically, it indicates how much the information associated with data can be obsolete. From this point of view, the extreme cases are the following:
- the cooperative system requires on-line access to the actual data in the various component databases;
- the system has access to copies, with a controlled degree of obsolescence with respect to the original data.

In the first case, the highest grade of coherence and liveliness is guaranteed; however, managing on-line connections and transferring large quantities of data over the network can sometimes be too expensive. In all the cases in which access to the most recent information is not mandatory, a reasonable alternative is to create, off-line, copies of data in the remote systems, and to allow access only to these local copies. Clearly, in this case, access to these secondary copies will mainly be read-only; otherwise, maintaining consistency with the primary copy may be a difficult task. On the other side, one has much more freedom in correlating and transforming heterogeneous data, in order to build a local integrated database.

It is important to note that this criterion can be applied

a real case is shown in the Discussion.

Figure 1: Multidatabase

independently to the various components of data: many applications require actual liveliness for some of their data and can tolerate obsolescence for others.

DATA-CENTERED COOPERATION: A CLASSIFICATION

The discussion in the previous section suggests that the design of a good cooperative system must be a compromise between efficiency, flexibility, and liveliness of the stored data. Based on the three criteria we proposed above for data-centered cooperation, we now present some important categories of cooperative systems. It is worth noting that there is no need to consider all combinations, since our criteria are not completely independent from one another. For example, usually a system that requires complex distributed transactions will be reasonably designed to provide a high degree of transparency, through a sophisticated integration architecture.

With this in mind, we discuss three different main categories of cooperative systems. A *multidatabase* is a system in which the highest degree of transparency, complexity, and up-to-dateness is provided. A *data warehouse* is a systems in which a high degree of transparency is still provided, but liveliness of data is sacrificed to achieve better integration and flexibility. Finally, a *local information system with external data* is a system offering support for simple transactions, with a low degree of transparency and varying degree of up-to-dateness. It is reasonable to expect that real needs of cooperation will often fall in between these categories, and that aspects of several of them must be incorporated. Anyway, we choose these systems because they allow an interesting discussion of the main features and because the intermediate solutions can often be seen as "linear combinations" of them. A discussion on how these architectures can be implemented using existing middleware tools can be found in the following section. Applications of our classification to

Multidatabases

A *multidatabase* is a cooperative system offering an integrated interface over the component databases, hiding distribution and heterogeneity, supporting complex operations and on-line access to remote data. Requests from local clients continue to be locally satisfied by the single component databases, thus preserving autonomy. This architecture corresponds to the extreme case in integration needs, with high degree of transparency, complexity, and up-to-dateness. This kind of cooperation is natural in all the cases in which it is necessary a strong integration among databases.

In many cases, the burden of managing distributed transactions is too high, and a controlled level of obsolescence is tolerated to achieve better performance. Also, the high degree of transparency can be offered by a multidatabase in different ways. In some cases, the component databases will share all their data; in other cases, only a pre-defined set of functionalities or services. The higher is the degree of transparency, the higher is the flexibility, but also the implementation cost and the complexity of the system.

The development of multidatabases requires suitable methodologies and tools for integrating the various database features (schemes, data, and languages).

Figure 1 shows the logical architecture of a multidatabase. A particular class of multidatabases, in which the integration occurs along the data dimension, corresponds to the *(tightly coupled) federated database systems* (Sheth and Larson, 1990), which appear to external users as single databases, with a single schema, and allow to perform complex queries and updates.
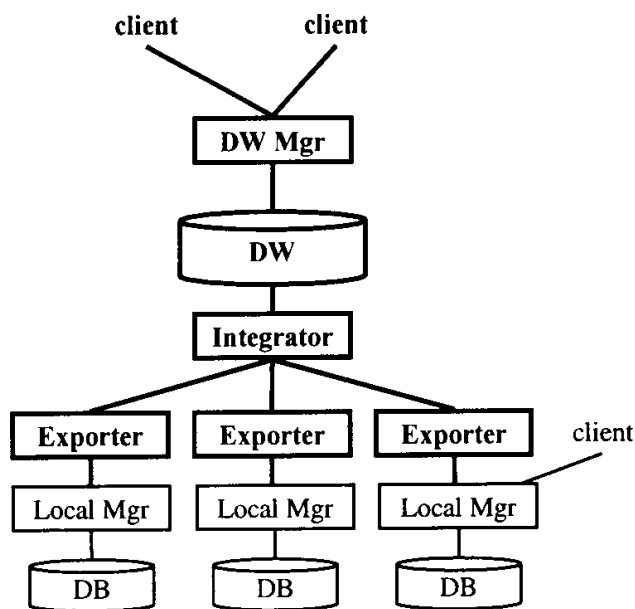
Data Warehouses

The use of secondary copies can be made extreme when planning or analysis needs impose to access large quantities of data; often, in these cases, it is not required to access the most recent version of the data, and a controlled level of obsolescence can be acceptable. In this case, querying the database is the main activity; queries can involve large quantities of data, and impose complex transformations, due to the need for correlating different data sources, in such a way that on-line processing may be very expensive. If so, the cooperative system is likely to be a *data warehouse*, that is, a collection of data with the following characteristics (Inmon, 1996):

- a high degree of transparency, since remote data are replicated and integrated in the warehouse to become homogeneous;
- mainly read-only access;
- limited, but indeed controlled, degree of up-to-dateness.

In the Data-Warehouse approach (shown in Figure 2) data are extracted from the component databases and integrated in the data warehouse in an off-line fashion. Of course,

Figure 2: Data-Warehouse approach

this makes updates a problematic task; however, read-only access is granted, with a great transparency and flexibility. The applications supported by a data-warehouse are typically oriented to decision support (for marketing, sales, financial analysis), investigation, and summarization. This architecture has attracted a great interest in the marketplace (OLAP, data cube, and multidimensional database technologies (Chaudhuri and Dayal, 1997)).

Multidatabases vs. Data-Warehouses

We mention advantages of each of the two proposed architectures with respect to the other. The advantages of the data-warehouse architecture are the following. First, it avoids conflicts between the operational activities (carried out on the primary data) and the decision support ones (carried out on the warehouse); in fact, data are replicated and accessed off-line. Also, the cooperative application can be available even when the primary source of data is unavailable (also because of execution autonomy). Finally, the warehouse approach allows to perform complex restructuring and aggregation over heterogeneous sources.

Correspondingly, the advantages of the multidatabase architecture are mainly due to the fact that a

multidatabase supports on-line access to current data, which is sometimes essential; in fact, up-to-dateness of secondary copies of data are difficult to maintain if primary data change rapidly; finally, a multidatabase can support unpredicted queries, whereas a data-warehouse might be tuned for specific aggregations.

Thus, there is no clear distinction between the two approaches. For instance, a complex system may require to manage: (i) data whose up-to-dateness is essential; (ii) data whose primary copy is expensive to access (with respect to the actual need for up-to-dateness); and (iii) stable data that are always aggregated in the same way. Therefore, we could need an integration of replicated and primary data. Figure 3 shows an intermediate solution satisfying such needs. Even in this case, our classification can be useful, since it can help in finding the portion to be implemented by means of a multidatabase (on-line) approach and the portion that can be replicated and accessed off line.

Local Information Systems With External Data

A *local information system with external data* is a cooperative system, localized on a single site corresponding to one of the component systems, with the need to access one or more remote databases, with the following characteristics:
- low degree of transparency in accessing distributed data; every database exports a set of pre-defined services (data and functions), based on appropriate cooperation agreements, and the task of the application is to integrate remote

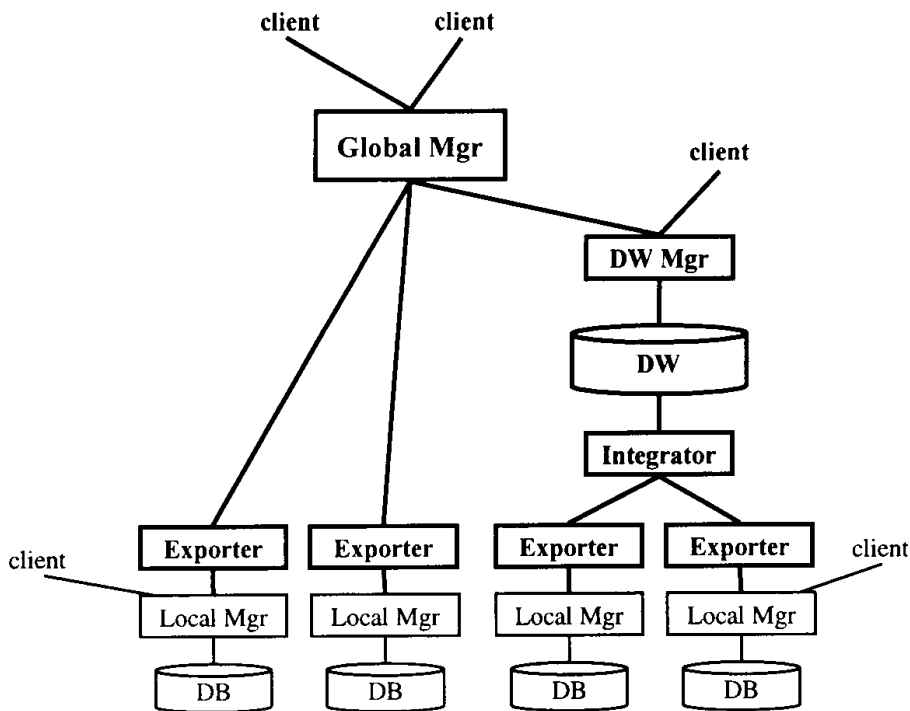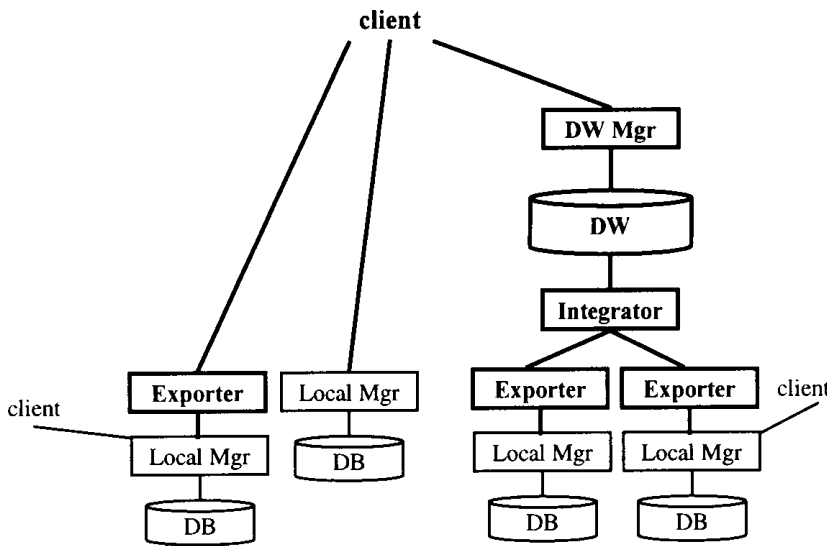*Figure 3: An intermediate solution between a multidatabase and a data warehouse*

Figure 4: Local information system with external data

data; thus, the level of integration of the external databases is very limited;

• small amount of distributed operations (queries and transactions), with low complexity; this is somehow the distinctive feature of these systems, that is, the need to access remote databases mainly one-at-a-time; the simplicity of transactions makes it possible for the application to guarantee correctness and consistency without dedicated mechanisms;

Figure 5: Main client/server classes of techniques

On-line data transfer

Off-line data transfer

On-line message exchange

Off-line message exchange
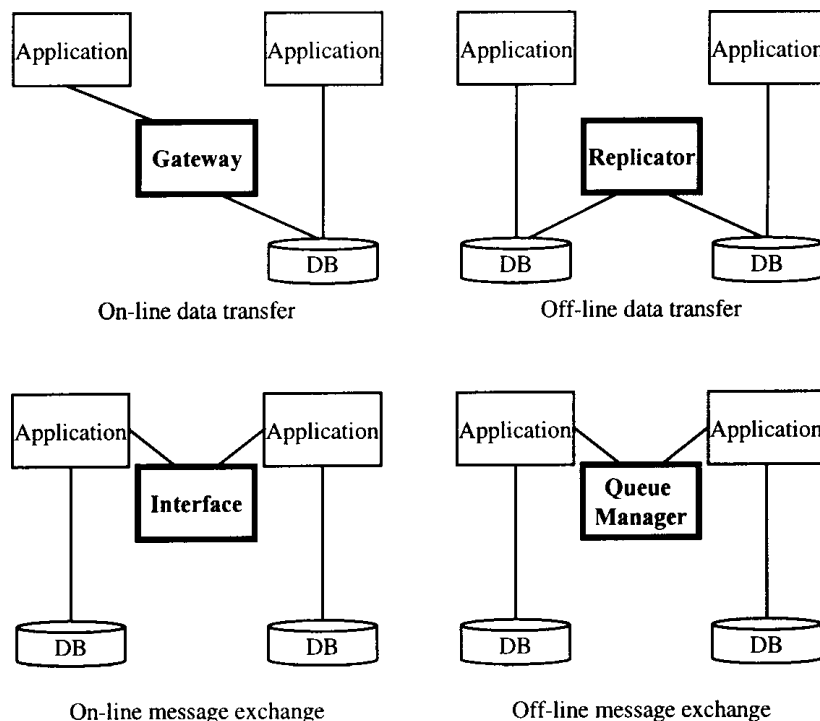
• variable degree of up-to-dateness; usually, access to on-line data can be provided, but also intermediate solutions based on replication can be adopted.

Figure 4 shows the architecture of a local information system with external data. The application of the local system is responsible for the integration, and includes also the management of data conversion and access control. The approach is meaningful only in presence of simple operations.

ARCHITECTURAL TECHNIQUES

Cooperation between components in a complex environment is usually performed by means of the client-server paradigm. Because of the complexity of the applications, the overall architecture is often a multi-tier client-server one, where a server is in turn a client of another service, and so forth.

The various techniques can be classified into two main classes: (i) techniques based on *data transfer*, and (ii) techniques based on *message exchange*. For both of them, we further distinguish between *on-line* techniques and *off-line* ones. Thus, we obtain four main classes of client/server techniques, shown in Figure 5. Each technique has possibly several implementations as a *middleware service*, that is, a software service that resides in an intermediate layer between a computing platform (above the operating system and the networking software) and an application (Bernstein, 1996). A *middleware tool* is a software package offering a (possibly integrated) set of middleware services. Usually, each middleware tool is based upon one such technique, providing one class of functionalities, useful for loosely cooperative systems. However, new commercial tools are arising, that allow to use functionalities corresponding to several of these techniques, in a more integrated fashion.

It is also worth mentioning that the explosion of the World Wide Web represents an extraordinary opportunity for establishing cooperative initiatives (Atzeni et al., 1997). In fact, it is based on a standard connectivity environment (the Internet), and offers a uniform inter- face for sharing data, at the same time guaranteeing complete transparency with respect to the distribution of data.

On-Line Data Transfer: Gateways

A *gateway* is a middleware tool enabling

applications for one database to access data over another database, usually by means of data manipulation language (DML) commands, such as SQL ones. The goal of a gateway is that of mapping client commands into server commands, and data returned from the server in the client format.

Gateways exist in the relational world to access, from a relational application, both legacy data and relational data living in a heterogeneous DBMS (possibly using different SQL dialects, data types, and constraints). This technique is flexible since the client, if authorized, can access the whole server database; we note, however, that some tools allow read-only access, especially in accessing legacy data. On the other hand, this technique can be rather inefficient, since there are no predefined queries, that is, the server has to execute essentially casual queries. For these reasons, it is usually acceptable to use a database gateway only for accessing legacy data from a relational client, if accesses have relatively low frequency and the answer time is not critical.

A multidatabase can be implemented over a *database gateway server*, a software tool offering in an integrated fashion several functionalities, among which SQL on-line data transfer and distributed transaction management. The database gateway server plays the role of database integrator, becoming the data server for the cooperative application. The architecture corresponds to that of a distributed DBMS, implemented with gateways, and accomplishes the goals of a federated database system, although only in a limited way. It offers high degree of transparency (especially in defining a global database scheme), supports the complexity of the operations (with a number of limitations: efficiency of complex queries is not guaranteed, and it is not always possible to update non-relational data), and a varying level of liveliness, according to the needs. (Refer to (Rezende and Hergula, 1998) and (Carey et al., 1998) for an in-depth discussion on gateways and database gateway servers.)

Off-Line Data Transfer

Off-line data interchange means that data are extracted from the server database, transformed, and then stored in the client database; this technique is thus based on replications. Ad-hoc solutions to replications have been used for decades; current tools offer a systematic approach to replication, usually to support data-warehousing environments. In particular, there are possibly separate tools for the various activities: extraction of data (incremental, with change detection), translation (between heterogeneous environments), integration, cleaning, and aggregation. It is possible to implement replicated data collections by means of specific commercial tools for extracting and replicating data. In particular, some tools capable of accessing heterogeneous and distributed sources have been recently proposed. The architecture is similar to the one based on a database gateway server, except for the fact that there is no on-line databases integrator, but the integration is performed off-line, and there is a global inte-

grated database (the Data Warehouse). Such an architecture has to be supported by a number of additional tools, for administration (to manage the global scheme) and data access and analysis, with the goal of achieving a decision support system and supporting on-line analytical processing.

On-Line Message Exchange

There are several techniques for the synchronous exchange of messages among distributed applications, implementing a function-oriented interface between the client and the server. The simplest tool is represented by *Remote Procedure Calls* (RPC): a client sends a message to a remote server, containing the name of a procedure to be invoked and the necessary parameters; the server executes the procedure and returns the results in a reply message. More sophisticated implementations exist in database management systems, ranging from open system APIs to (SQL) stored procedures in distributed environments.

On-line messages can also be managed by *distributed transaction processing monitors*. A (traditional) TP monitor offers efficient and reliable access from remote terminals, based on queue management. A distributed TP monitor enhances the functionalities of a traditional one offering remote services in a distributed environment. These software products support the distributed execution of transactions, guaranteeing also for their "logical" correctness, in particular ensuring their atomicity and durability. Object-oriented services offered by modern distributed object technologies (e.g., Object Request Brokers) are the natural direction toward which distributed TP monitors are evolving.

These techniques can be used to implement forms of cooperation in which the component ISs share set of specific functionalities operating on the data they control, rather than the scheme of that data. The global cooperative IS is then built with the goal of defining complex services based on those "elementary" functionalities. In this case, the use of a distributed transaction processing monitor as the main middleware tool, because of specific functionalities to achieve a strong level of coordination in distributed transactions, allows to implement effectively atomic and persistent transactions.

A similar architecture is defined based on an Object Request Broker, a distributed middle- ware tool enabling the interoperation of objects, which can easily encapsulate (legacy) functions and applications.

A discussion on the relationship between gateways and brokers (and other tools for message exchange) can be found in (Stonebraker et al., 1998).

Off-Line Message Exchange: Queue Managers

This technique is the asynchronous counterpart of on-line message interchange. It is again function-oriented, but the message flow is handled by a queue manager (the corresponding tools are classified as *Message-Oriented Middleware*, MOM). The asynchronism solves a main drawback of on-line cooperation, since it allows to tolerate un-

availability of the server connection. This may happen because of a network malfunctioning, or because the server is overloaded, and decides to interrupt remote on-line services because of execution autonomy. In some cases, the ultimate goal of the application allows to de-couple the synchronicity of the cooperation, for instance, when a client sends a requests to the server, without the need for an immediate answer.

DISCUSSION

In this paper, we have seen that data-centered cooperation may correspond to very diverse needs, and so different solutions can be offered. Specifically, it turns out that there is not only one coordinate for complexity (and, as a consequence, for cost and risk of failure). As with any complex engineering task (and database cooperation is in general a complex engineering task), each decision has to be based on a careful evaluation of costs and benefits. We have offered some lines along which such an evaluation could proceed.

Let us now briefly discuss how our classification can be applied to the data-centered co- operative applications of the Department for Higher Education in the Italian administration. We have found major applications in each of the three main categories of cooperative systems introduced so far:

- an envisioned application for the on-line management of the Department's budget and expenses will fall in the category of multidatabases, since it requires on-line coordination of information managed by different sources;
- an application for the a-posteriori evaluation of the performances of the universities falls in the data warehouse category, since it can work with off-line data coming from the various universities in the country;
- an application for the support to the submission and evaluation of research proposals falls in the category of local information systems with external data, since it handles its own data, with some access to other sources (for example the databases that lists all the professors and all the universities in the country, with their schools and departments).

An important issue to be discussed here is that cooperation is often associated with migration (Brodie and Stonebraker, 1995) or business process reengineering (Hammer and Champy, 1993). It is important to stress that the three problems are different, although closely related. Business process reengineering is aimed at improving the way an enterprise (or organization) operates. As such, it may require information system (and database) cooperation, as a mean to support its goals. Migration is an activity that aims at replacing old, expensive, inflexible hardware and software systems with more modern ones, with a number of goals that go from reducing maintenance costs to offering new services. Therefore, migration can be seen as a prerequisite to information technology support to business process reengineering, because of the increased flexibility that modern systems offer, compared to old "legacy" ones. Cooperation, as we argued in the introduction, is essentially motivated by the need to respond to new business requirements, both in terms of new services to be offered because of the development of networks and because of reorganization of companies.

Now, it has been argued (Brodie and Stonebraker, 1995) that migration, except for a few extreme cases, should be pursued with a conservative strategy, composed of a number of small steps, rather than with a revolutionary one-shot replacement. This would require the coexistence of new and old components, which should therefore cooperate. Our view of cooperation does not require migration of components, but at the same time does not prevent it. In a sense, it can stimulate reflections about migration needs (for example in terms of devising enterprise standards in languages, database software, operating system, or hardware), without imposing them as prerequisites: in fact, migration should be gradual, but long term goals should be set. Similarly, cooperation and migration could help in a business process reengineering initiative, offering both a stimulus in reconsidering the way the activities in the organization are conducted, and a better framework to implement new operating procedures, because of more modern and flexible systems.

REFERENCES

P. Atzeni, G. Mecca, and P. Merialdo. To Weave the Web. *International Conference on Very Large Databases (VLDB'97)*, pages 206—215, 1997.

C. Batini, M. Lenzerini, and S.B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323—364, December 1986.

P.A. Bernstein. Middleware: a model for distributed system services. *Comm. of the ACM*, 39(2):86—98, February 1996.

M.L. Brodie and M. Stonebraker. *Migrating Legacy Systems: Gateways, Interfaces & the Incremental Approach*. Morgan Kaufmann, San Mateo, California, 1995.

M.J. Carey, L.M. Haas, J. Kleewein, and B. Reinwald. Data access interoperability in the IBM database family. *Data Engineering* 21(3):4—11, 1998.

S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26(1):65—74, March 1997.

G. De Michelis, E. Dubois, M. Jarke, F. Matthes, J. Mylopoulos, M.P. Papazoglou, K. Pohl, J. Schmidt, C. Woo, and E. Yu. Cooperative information systems: A manifesto. In M. Papazoglu and G. Schlageter, eds., *Cooperative Information Systems: Trends & Directions*, pages 315—363. Academic Press, New York, 1998.

H. Garcia-Molina and M. Hsu. Distributed databases. In W. Kim, editor, *Modern Database Systems*, pages 477—493. ACM Press, 1995.

J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, San Mateo, California, 1993.

M. Hammer and J. Champy. *Reengineering the Corporation: A Manifesto for Business Revolution*. HarperCollins Publ., New York, 1993.

W. Kim, editor. *Modern Database Systems: the Object Model, Interoperability, and Beyond*. ACM Press and Addison Wesley, 1995.

W.H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, second edition, 1996.

F.F. Rezende and K. Hergula. The heterogeneity problem and middleware technology: experiences with and performance of database gateway. *International Conference on Very Large Databases (VLDB'98)*, pages 146—157, 1998.

A.P. Sheth and J.A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183—236, September 1990.

M. Stonebreaker, P. Brown, and M. Herbach. Interoperability, distributed applications and distributed databases: the virtual table interface. *Data Engineering* 21(3):25—33, 1998.

Endnotes

1 This paper is part of a joint work between Italian Autorità per l'Informatica nella Pubblica Amministrazione and Dipartimento di Informatica e Automazione at Università di Roma Tre. The authors were also partially supported by MURST and Università di Roma Tre. A tutorial based on the same content was presented at the E_R Conference in Cottbus, Germany, October 1996 and the CAiSE Conference, in Barcelona, June 1997.

2 It is worth noting that, in the area of Computer Supported Cooperative Work (CSCW), similar concepts are often referred to as *passive* and *active*, respectively.