Université de Genève
Faculté des sciences économiques et sociales
Département de systèmes d'information

KNOWLEDGE PATTERNS OF DISTRIBUTED INFORMATION SYSTEMS -

THE CASE OF DISTRIBUTION DESIGN AND IMPLEMENTATION BASED ON INTEGRITY CONSTRAINTS OPTIMISATION

THÈSE

Présenté à la Faculté des sciences économiques et sociales
de l'université de Genève
Par
SNENE Mehdi

Pour l'obtention du grade de
Docteur ès sciences économiques et sociales,
Mention systèmes d'information

Membres du jury de thèse:

Mr Dimitri Konstantas, Président du Jury, Professeur, CUI, Université de Genève,
Mme Giovanna Di Marzo Serugendo, M.A, CUI, Université de Genève.
Mme Moira C. Norrie, Professeur, Ecole Polytechnique Fédérale de Zurich,
Mr Michel Léonard, Directeur de thèse, Professeur, CUI, Université de Genève,
Mr Thibault Estier, Professeur Assistant, Ecole des HEC de Lausanne,

Thèse no 578
Genève, 2004

La faculté des sciences économiques et sociales, sur préavis du jury, a autorisé l'impression de la présente thèse, sans entendre, par là, émettre aucune opinion sur les propositions qui s'y trouvent énoncées et qui n'engagent que la responsabilité de leur auteur.

Genève, le 17 Décembre 2004.

Le doyen
Pierre ALLAN

Impression d'après le manuscrit de l'auteur

Abstract

The emergence of new technologies and the increasing decentralisation of information systems have given rise to new forms of distributed information systems. Different approaches and methodologies are proposed for the design of these new forms of information system. They propose different structures and ways to express this distribution throughout the notion of component but rarely address the issue of: *how to distribute the design schema between the existing sites ?* In this thesis, we propose an innovative approach to answer this question. This approach is based on the knowledge surrounding the distributed information system projects. Firstly, we define three knowledge patterns that encapsulate and underline the knowledge extracted from the design, the implementation and their overlap environments. Secondly, on the base of these patterns, and more specifically the overlap knowledge pattern that represents the area where the distribution decisions are taken, we define a set of formulas and algorithms that generate different distribution possibilities and their related costs. These costs are obtained by calculating the number of needed data calls for integrity constraints check and validation between sites. For this purpose, we proposed an enhancement for the integrity constraint specifications in order to adapt them to the context of distributed information systems. The integrity constraints represent an attractive distribution approach due to their multiple layers aspects. In fact, they regroup the data and the functional aspects. Thus, optimising their distribution is similar to optimising the data and the functions distribution. This optimisation covers the different knowledge overlap pattern components. Finally, we establish a list of required knowledge exchange at the distribution design step between the different project partners. This list underlines the most important knowledge that has to be considered by them in order to prevent the resulting distributed schema from being modified later.

Acknowledgements

I am profoundly grateful to my advisor Professor LONARD Michel. Throughout my time as a research assistant at Geneva University, he has provided logistics, assistance, guidance, ideas, editing and a healthy skeptical ear. I am grateful to the members of my committee for their individual influence on this research. I am indebted to Professor MOIRA C. Norrie for the early inspiration during the summer school in ETHZ and the honour she made to me by accepting to be member of the committee. I would like to thank Dr. Giovanna Di Marzo Serugendo for letting me always see the big picture and for extending his office hours so often to discuss both my progress and philosophical aspects of my PhD thesis. Special thanks to the Professor KONSTANTAS Dimitri for his support, enthusiasm, and advises. I will always be indebted to him for all his friendliness and help. Thanks to Professor Thibault Estier for accepting to be member of my committee.

Thanks to all my students and colleagues for their precious help and collaboration. Special thanks to Jorge for all what we have shared together. Special thanks also for Slim for all his patience, advises and time shared together. Many thanks to Nicolas, Abdelaziz, Toha, Lu, Dr. Than, Dr. Jolyta, Dr. Ciran, Dr Jean Henri, Evelyne.

I would like to express my deep and sincere gratitude to Hammouya-Tatai Family for their precious help, support and love. Special thanks to Manaï family for their kindness and help. I am also grateful to my friends, Kais and Ezzeddine for their precious presences and their friendliness. Tender thought to her, she will recognise her self cause she is the unique.

I would also like to thank my parents Hedi and Néjiba for their unwavering love, support, and encouragement for my academic pursuits. Special thanks to my brother, Mohamed Eines, for his understanding and patience. He has asked for so little but has given me so much. Great thanks to my Brother and friend, Wassim, for the fun, all the laugh and the love that we shared.

Table of contents

Chapter I

Introduction

This chapter presents the motivation of this thesis, its objectives, its relevance to the area of distributed information systems design, and the strategy adopted throughout its development. The concept of distributed information system is introduced in this chapter as a framework in which important issues in the area of distributed information systems modelling, design and implementation are recognized. We conclude that a better understanding of basic design concepts is fundamental in the development of solutions to support the correct design and implementation of distributed information systems. This better understanding and the evaluation of its consequences are some of the objectives of this work.

The chapter is further structured as follows: section 1 identifies some issues in distributed information system design, section 2 introduces and discusses the research

objective, section 3 discusses aspects related to the research contributions and section 4 presents the thesis's overview.

1.1- Problem statement

The dynamic behaviour of the organizations and their environment generates new informational and structural needs that should be satisfied for the survival of these same organizations. The unification of organisations and the effect of the globalisation imply an integration of the various data coming from various structures to coordinate them and thus ensure their integrity. Indeed the fusion of various organizations rather often implies the fusion of their data. This integration is done through their different correspondent information systems. The Information Systems have a major role intra and inter organisational by ensuring the collection, storage, share and the integrity of the data within the organisation. In addition, an operational and robust information system must provide the right information to the right person at the good moment and with the lowest cost. Information system has to be conceived as the principal actor in the decision-making process of any organization [Leonard92].

These information systems, therefore, must imperatively be able to accompany these structural and informational dynamism generated by the continuous evolution of the internal and external organisational environment (unification, decentralisation, delocalisation,…) and thus to evolve in order to satisfy the updated organisational and technical requirements. Because of their conceptual nature, the traditional information systems, generally centralized, are not easily maintainable and consequently can't be adapted to these changes in order to reflect adequately the new requirements [Deweger99]. Indeed these information systems permit only the evolution of the information contained in the databases, which represent informational set obeying to a fixed *conceptual diagram* with a fixed set of *constraints*. A conceptual diagram is a static and a graphical representation of data and organisational environment representing the internal data schema used in the DBMS (Data Base Management System). The constraints are the translation of business rules and of the data use restriction. They represent the integrity constraints of the information system. The evolution of the conceptual diagram and consequently the internal database diagram, the constraints and the specifications of the dynamic aspects (activities and behaviours of the specifications) are not supported by these traditional information systems. The schema structure of a database is not sufficiently robust to guarantee the relevance of information present in this base. To contribute to the maintenance of coherence in the database, it is necessary to specify constraints enacting of the organisational and informational properties, which the database must respect to preserve the data integrity while applying the different business rules that manage the data flow inside the organisation. They can appear in a various *types*, *contexts* and *ranges*. These constraints change and should be changed in order to be adapted to the new needs. In fact, the evolution of the organisational environment or/and the informational environment often implies a radical change in the thinking of these constraints, which control these environments. By information environment evolution we consider the

update of the existing constraints, the addition of new constraints or the integration of the constraints imported from a different system.

The integration of workstations in a distributed environment and the maturation of database management system (DBMS) technology have coincided with significant developments in distributed computing and parallel processing technologies. The result is the emergence of *distributed information systems* based on *distributed database management systems* and *distributed applications system*. A distributed information system is a *collection of data*, *transactions*, *integrity constraints* and sites [Leitzelman98]. Data is concretized by a set of multiple, logically interrelated databases distributed over a computer network. This set is managed by a distributed database management system, which makes the distribution transparent to the users.

Constraints of distributed Information system are distributed on various levels. They can be implemented on the different existing levels of the system. In architecture such n-tiers levels, constraints can be implemented in graphical interfaces, application level or the data level. It is important to make the evolution of these integrity constraints easier to ensure the survival of information system. Indeed, the consistency of the data is based on the rules and in a context of intra and inter organisational, an inconsistency in the data of the various sites involves system failure. The information system evolution based on its integrity constraints evolution is difficult to manage because this evolution is complex and still controlled neither by the distributed database management system nor by the design and development tools of information system. The control of this evolution is required since it is the guarantor of the data's quality and the information system's performances.

The evolution of integrity constraints in a distributed information system can cause three main problems localized at different levels:

- Transactions design: it has to meet a double aim which is to guarantee that each transaction is coherent with respect to the integrity constraints and to meet the requirements of the information system performance in terms of compromise flow. In fact an integrity constraint that uses a shared data between different sites causes an imperceptible network traffic that can considerably slow down the execution of the transaction. On the other hand the differed execution of the constraint or its removal can generate data transgressions, which make the database incoherent.

- The development of integrity constraints: They can be developed in an ad-hoc manner inside the information system (stored procedures). This solution is not very reliable and its maintenance is excessively difficult. The other possibility is to develop the constraints in the form of releases and assertions managed by the DBMS. It is the most accountable solution but the most difficult to implement because of the lack of adequate design methodology and the difficulty to forecast the behaviour of releases in realistic scenarios as well as its weak performances.

- The maintenance of integrity constraints: the general problem is to check that the integrity constraints based on distributed data among different sites, are verified after any performed transaction.

New architectural and technical forms of information systems add a more significant level of complexity due to the decentralization of the constraints, treatment and data. These architectures increase the deployment and the execution possibilities because of the number of existing sites. Indeed, the simple separation of the various functional levels as it is done in a classical architecture (Data, Treatment, Presentation) is not enough and the choice of the site of deployment or execution becomes significant for the optimisation of the production of the system. In these architectures, these decisions of distribution are generally made during the implementation phase. The conceptual structures offered to designers to allow them to express their needs for distribution (concepts of packages, business component,..) do not match with the rules used by developers for building their distributed components. In fact, software components represent a single and autonomous concept of real world. They encapsulate all the data concerning this concept including name, goal, behaviour and all other information with regard to them. In fact, a software component is a set of objects that can be physically deployed on two or several sites. It is usually made up of one or of several distributed components that offer together the various aspects of distribution necessary to the software component. The distributed components represent the physical modules used for application assembling. They encapsulate given data and treatments and provide their services through well-defined interfaces.

The first defect of these approaches is that these concepts are not based on a common knowledge shared between the developers who have their own constraints of distribution and deployment from one hand and the designers who have the functional and organisational system view from the other hand. So the model of distribution obtained at the design step is generally modified by the developers to improve its output and consequently the output of the system. The operated modification can change completely the main goal aimed by the designers. This discontinuity is harmful to the design consistency.

The second defect of these approaches is their lack of constraints. The distribution of an information system must include two aspects, which overlap: the data system aspect and the functional aspect. The functional aspect is largely covered by the rules. Indeed, the rules manage all the system transactions to ensure the validity of these transactions and the integrity of the data. Each functional transaction calls a set of constraints that manage the zone of responsibility where the data risks to be modified or transgressed. These constraints represent for us the internal operational aspect of the system. The constraints represent also a link between the developer and the designer. In fact, by their double aspect, functional and organizational the constraints represent an overlapping zone between the design and the development steps. This zone can be used for to establishment of a dialogue and an exchange that can help us to realize a logical continuity of the distribution design. The traditional approaches of distribution

do not consider the constraints as a fundamental base of the functional system aspect and are not well treated or not taken into account in these approaches.

1.2 - Research goal

The majority of distributed information system design methodologies do not take into account the necessary continuity that has to exist between the various steps of the systems lifecycle. Indeed, these methodologies are based on the data without taking into account the functions (distributed database [ÖZSU99]) or on a functional approach without taking into account the integrity data aspect (UML [Cheesman00], Catalysis [Winters03]). In the two approaches, the feedback from an implementation environment is missing at the design step. Our approach is an attempt to combine these various axes since the design step, which are the data aspect, the functional aspect by considering the functional interaction rules and finally the feed back of the projection towards the environment of implementation. This thesis proposes a *distributed information system design methodology*, which includes different interaction patterns that formalize the different data flows existing between the usual design steps.

We define three different levels of interactions. The first is the developer's need of information from the designer step. As an example, developers need to know the different organisational aspect for a better understanding of the designers choices of the distribution and allocation decisions. The second is the designer's need of information from the development step. For example, designers need to know the system security protocol used in the development platform to adapt to it their design schema. Finally, we consider the potential *knowledge overlap* between the design and development steps. For example, designers have to identify the diverse existing integrity constraints and to adapt them to the implementation by specifying the different locations of execution. For each level, we define a specific pattern that identifies the existing potential risk area and specifies the manner of behaviour to resolve them.

In this thesis, firstly we give our definition of distributed information system based on an evaluation grid, which regroups different criteria used to compare the different existing information systems architectures. We focus on the OKP pattern (Overlap Knowledge pattern) and we propose different algorithms that help designers and developers to combine their system and organisation knowledge to resolve the data and functional allocation problem. The distribution design based on integrity constraints is considered as the main goal of our research. The algorithms we propose attempt to minimize the transaction time of different verification and validation of each integrity constraints by getting the concerned set of data located on the same site.

1.3 - Research contribution

The main contribution of this thesis is the development of a framework and an associated suite of methods that enable designers to have assistance for the database schema fragmentation and allocation while considering the different parameters involved. Several algorithms are proposed ***FKDO****: Foreign Key Dependencies Optimisation,* ***ICDO****: Integrity Constraint Dependencies Optimisation,* ***GIDO****: Global Integrity Dependencies Optimisation* are the main component of the Overlap Knowledge Pattern. In our approach, the final proposed solution for the data fragmentation and allocation does not take into account the replication possibilities. In fact, we consider that such solution is not excluded from the development and can be added easily in the final generated combination of distribution. Our different algorithms are the principal components of our proposed methodology for the design of distributed information system based on integrity constraints dependencies optimisation. Our proposed framework makes abstraction of the functional implementation aspects. In fact, we consider that integrity constraints reflect the functional aspects and encapsulate them. We consider that any existing function has to establish a call to an integrity constraint that corresponds to the treated or exploited data. These calls are mandatory to preserve the data integrity into the database and to ensure the maximum of information system reliability.

The second contribution of this dissertation is the proposed integrity constraint specification framework. In fact, we consider that the simple mathematical or syntactical expression is not sufficiently described to use it as the principal support of system function's implementation. The proposed specification is enhanced by new definition criteria useful to express the different execution root and the propagation of the risk across the different existing table. We add the notion of the *risk class* that indicates the *root class* that the application of one of the primitives may violate the integrity constraint. This violation may occur on the data of the risk class or on the data that belongs to any other class within the integrity constraint context. This specific path of data needed to be verified after the execution of a risky primitive as defined in the range board is called the cross reference risk table. The *cross reference risk table* regroups the different risk classes defined in the range board. It emphasizes the relation between the root class and the classes that will be affected by the transgression of any corresponding primitive.

Finally, this thesis defines a set of interaction patterns between the various participants in the life cycle of a distributed information system. These patterns treat initially the exploited knowledge by each one of these participants. Then, these patterns will be the intermediary of communication to establish a form of formal communication between them to allow a better comprehension of the system and thus to ensure a better continuity between the various levels. These patterns are divided into three categories corresponding to three knowledge interaction levels. The first is the designer's knowledge needed by developers to capture the design environment and to well understand the purposed conceptual diagram. For example, designers need in some special case to fix some critical data on some specific sites to ensure their availabilities

in case of network shutdown. Developers usually adapt the purposed conceptual diagram to the technical environment to increase the distributed information system performance and if it is not noticed, they can move these data from the specified site to another one without informing designers. The second category is the developer's knowledge pattern that regroups the different implementation parameters and the technical environment specificities. This pattern is useful for designers. It assists them in adapting their purposed conceptual diagram to the technical environment. For example, some application servers used to develop distributed information system (Tomcat, J2EE, …) have their own security framework. Any security approach used to design the distributed information system must fit into this framework [Kassem00].

Finally we propose a knowledge overlap pattern that encapsulates the common knowledge between the designers and the developers. An example of this knowledge is the exception treatment. The technical behaviour of each used technical environment is different from others and its integration within the conceptual diagram must be discussed and approved by both developers and designers.

1.4 - Thesis overview

The remaining chapters of this thesis are arranged as follows:

- Chapter 2 presents different existing information system architecture under their different taxonomy. We intend to present the classification of the different presented architectures based on some common suggested criteria. We define the distributed information system architecture that will be used in the current thesis.

- Chapter 3 gives an overview of the distributed information system design methodologies. We present two different design approaches. The first is based on functional approach; the second rather focus on data.

- Chapter 4 focuses on the existing exchange between the different design steps. We formalize them under pattern structure and detail the different existing interactions.

- Chapter 5 introduces some database concepts. We introduce also the integrity constraint specification enhancement proposed to adapt them to the distributed information system design.

- Chapter 6 presents our different algorithms for the distributed information design, which compose the overlap knowledge pattern. First, we present the *FKDO* algorithm that is based on foreign key. Then we present the *ICDO* algorithm that consider the different other forms of integrity constraint. And finally, we present the *GIDO* algorithm, which present how to combine the different algorithm and how to allow them the design distribution.

- Chapter 7 presents the benefits of the use of the different defined algorithms on the distribution process. It introduces different examples of use of the overlap knowledge pattern components. It gives the interaction way between designers and developers at the distributed information system projects.

- Chapter 8 presents M7TOOL CASE. This tool is a distributed information system developed according to the different defined algorithms. It is also a CASE tool that integrates this algorithms and assist users at the design and implementation of a distributed information systems.

- Chapter 9 summarizes the contribution of this thesis and brings out important points for future research.

Chapter II
Information systems architectures

This chapter presents the different existing distributed information system architectures, identifies a list of comparison criteria, and establishes a comparison grid of these architectures based on these criteria. The concepts of information system architecture and comparison criteria are introduced in this chapter. The comparison criteria enable us to distinguish between the different existing architectures and provide a better understanding of the nature of the distributed information system. In fact, our main definition of what is a distributed information systems is based on these criteria and does not correspond to the different literature definitions. For example, we consider that the integration of different existing information systems cannot be assimilated to a distributed information system but to a dispersed one. Instead, the comparison criteria grid localizes the distributed information system architecture by reporting to other architectures. This better understanding is one of the objectives of this research. In fact, the different existing information system architectures generate distinction and comprehension confusion between them. This thesis addresses only the distributed information system architectures. Therefore the nature of this architecture and the existing difference between it and other multi-sites architectures is the first step for a better understanding of this research.

The chapter is further structured as follows: section 1 identifies some trends in information system architecture, section 2 introduces and discusses the different information system architectures, section 1.3 discusses and establishes different comparison criteria and section 4 discusses the strategy adopted in the development of this work and presents the result of this comparison synthesised in the comparison grid.

2.1- Introduction

The maturation of database management system (DBMS) technology coincides with significant developments in distributed computing and parallel processing technologies. The integration of workstations in a distributed environment enables a more efficient function distribution in which application programs run on applications servers, while database functions are handled by dedicated computers, called database servers. The final result is the emergence of new information system architectures based on data distributed across different sites. These data and functions can be built from scratch or can be integrated in different existing monolithic information systems.

Many problems emerge at the design step of information systems based on these new architectural possibilities. The first problem is how to guarantee the maximum functional continuity if a sub-system is disconnected from the global information system. The second is how to minimize the data flows among different sites and to maximize the integrity of the global system. Business components have been presented as a possible solution to resolve these problems. This concept is based on the notion of the business partitioning. However, it is an informal process and it does not take into account the real constraints of Distributed Information Systems (performance, Integrity, QoS,) [Snene04].

The coming out of multi-sites information system architecture has begin in the 80th with the need to integrate the of different existing information systems dispersed over different organizations. This need appeared following the new tendency of organizations integration. Out of this need emerged the first architecture which is the dispersed information system. Nevertheless, this architecture is not flexible enough to manage the huge amount of existing data flows over the different subsystem due to its technical limits. In fact, the different subsystems are coupled using message protocol. This protocol creates an overload of the network and does not ensure enough transparency to users. Due to these problems, the network performances have increased and new technologies of integration approaches have appeared. The first approach is the federation approach. It consists of the creation of a federated schema composed of the different exported schema representing the different data sources. This architecture supports the federation of different data sources (text, database, file,..) and helps end-users by providing them with a unique access system. The second approach appears with the democratisation on the World Wide Web. The agent based information system is an architecture, which is proposed to help different subsystem get information from each other. This technology is used to establish a controlled communication between the different subsystems. In fact the different agents are dedicated to query and to get precise information. This technology is still hard and complicated to implement. All these architectures are based on different existing information systems that are integrated. The different mentioned architectures are not build from scratch but from existing parts. They are not conceived as a distributed information system but as autonomous parts without taking into account the existing informational overlap between them. The distributed information system is usually an architecture designed to be distributed and is based on a distributed database schema.

The design for the distribution is done in two stages. Firstly the global schema is designed as a centralised one. Then, designers with the assistance of developers and final users fix some part of the global schema on some sites. Finally the global distribution is done.

The next section details four above-mentioned architectures.

2.2- Information system architectures

2.2.1- Dispersed Information System

The dispersed information systems (DispSI) were developed to resolve the various needs for intercommunication between various existing subsystems. In this context, subsystems are developed independently from the others [Mkrygiannis00]. At the organizational level, these subsystems represent the various branches of industry and their interactions are based on the real world, i.e. on the interactions between the branches of industry. A local database is used as support for the data. These latter are managed independently by each subsystem (see Figure1).
Conceived to resolve different problems better than other information systems, such as

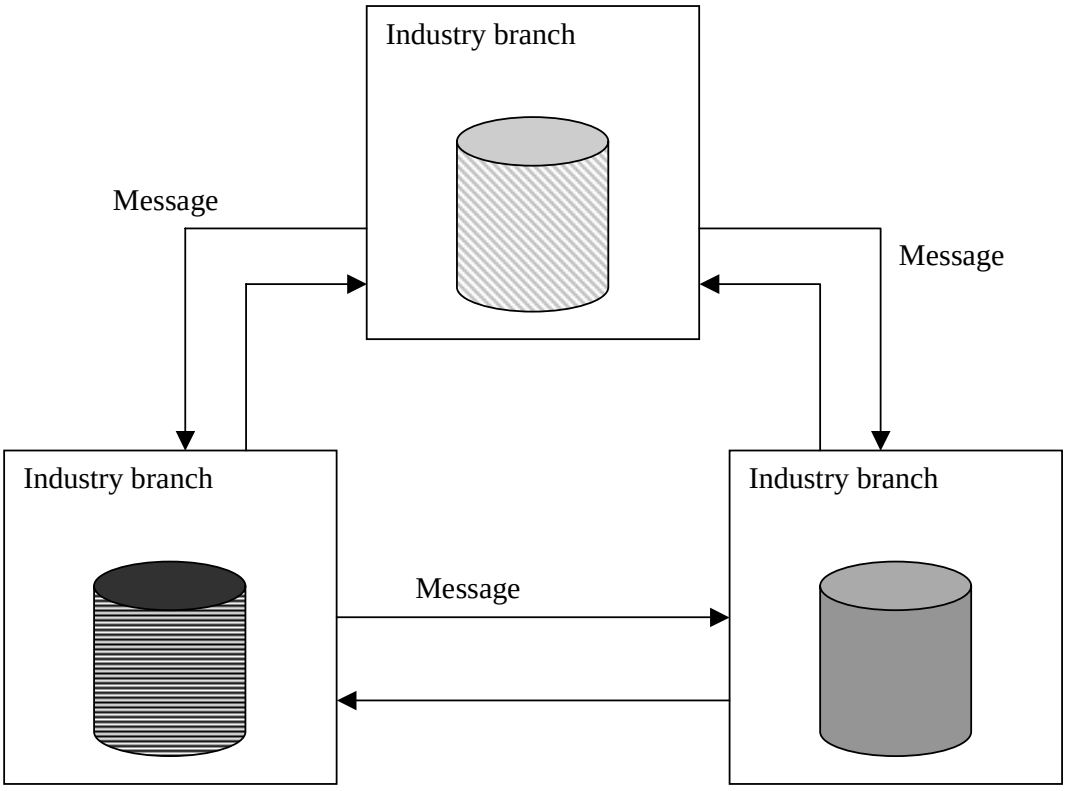


Figure1. Dispersed Information System [Mkrygiannis00]

the Federated Information Systems or Medians Information Systems, DispIS do not integrate a *global* data model to be used as a base for the subsystems integration. Indeed, the different subsystems communicate through a message protocol instead of directly carrying out the accesses on the data [Mkrygiannis00]. The aim of these messages is to transmit information between the subsystems, thus the operations on the local database are carried out only by the subsystem (the branch of industry) responsible of this database. The interactions must thus be correctly managed and based on the real relations of the branches of industry in order to maintain the separation between the various subsystems. This type of systems thus privileges the autonomy of the existing subsystems rather than their integration. Their purpose is not to support the exchanges of information or the distant accesses to the data, but to separate the roles from each subsystem and to limit their communication. This autonomy preserves the individuality of each subsystem [Mkrygiannis00].

The DispIS structure is relatively flexible for evolutions thanks to this autonomy of the various parts. However, this independence has a cost: the evolution of one of the subsystems can generate a greater complexity towards the remainder of the system, which interacts only in a limited way with it. The evolution of a subsystem must be controlled in order to prevent the subsystem from losing its autonomy to the profit of integration. This approach can also generate problems of information redundancy and incoherence between the subsystems. Indeed, the inexistence of direct relations does not allow a control, a comparison, and a continuous update of the data stored on different sites. This absence of constraints between the various subsystems, moreover, generates problems of synchronization between them. Indeed, modifications are realizable only by the system responsible of the database, which has the possibility of not immediately carrying out the update, for reasons of performances for example, thus possibly generating a temporary inconsistency of the database.

2.2.2- Federated Information System

Federated information systems (FIS) operate as a global layer over different existing data sources. The federation is ensured by a federation layer or a federator [Weher02]. The federator is a centralized system, with a model of control responsible for communication between various sites. The FIS are composed of a central common data model. This model guarantees the heterogeneity of the sources while respecting their integrity. It also brings a transparency for users who interact with one system instead of the various subsystems. There are two types of federated information systems: systems with dispersed configuration (loosely-coupled) and systems with grouped configuration (tightly-coupled). These two types of approaches are different mainly because the FIS with dispersed configuration does not integrate a total data diagram [Leser00] and tends to be regarded as a multi databases system, whereas the FIS with grouped configuration contains an implementation of a total diagram. The interest today tends to prefer this last approach based on a five-layer architecture [Leser00](see figure 2):

- The first layer represents the diagrams of the data sources: local schema,

- The second contains first layer's diagrams transformed mainly for a better interaction with the data model of the Federated Database System (FDBS): component schema,
- In the third, we have the subsets that will be taken into account by the FDBS: exported schema,
- The fourth layer represents the federated diagrams which are composed of the exported diagrams and which enable the system to establish the connection between the FDBS and the application: federated schema,
- The fifth layer defines the user view on the federated diagrams: exploited schema.

The problem of federation of the various information systems can be solved through two approaches of design for grouped configuration FIS as well as for other types of information systems:

- Top-down: we conceive a global and homogeneous diagram in order to meet the total needs. The diagrams of the sources are not considered in this process of design, but will be taken into account thereafter in an isolated way [Debenham02]. Through a vertical correspondence of diagram, i.e. to make correspond the total diagram with the different data diagrams, this solution supports the dynamism of the sources (creation, suppression). However, it involves a lack of semantics and of coherence between the data with respect to FIS [Busse99] [Leser00].
- Bottom-up: this approach is more restrictive than the first one as the total diagram is carried out from preset sources. The different existing data models are exploited for the design of the global model, but contrarily to the preceding method, these sources will be connected to make them correspond (horizontal correspondence) in order to obtain a semantic model. A great number of sources generate a multitude of diagrams to correspond that make the global model not easily maintainable [Leser00]. This solution is ideal to integrate pre-existent local conceptual diagrams in a total diagram but appears complex for a later evolution [Unger97].

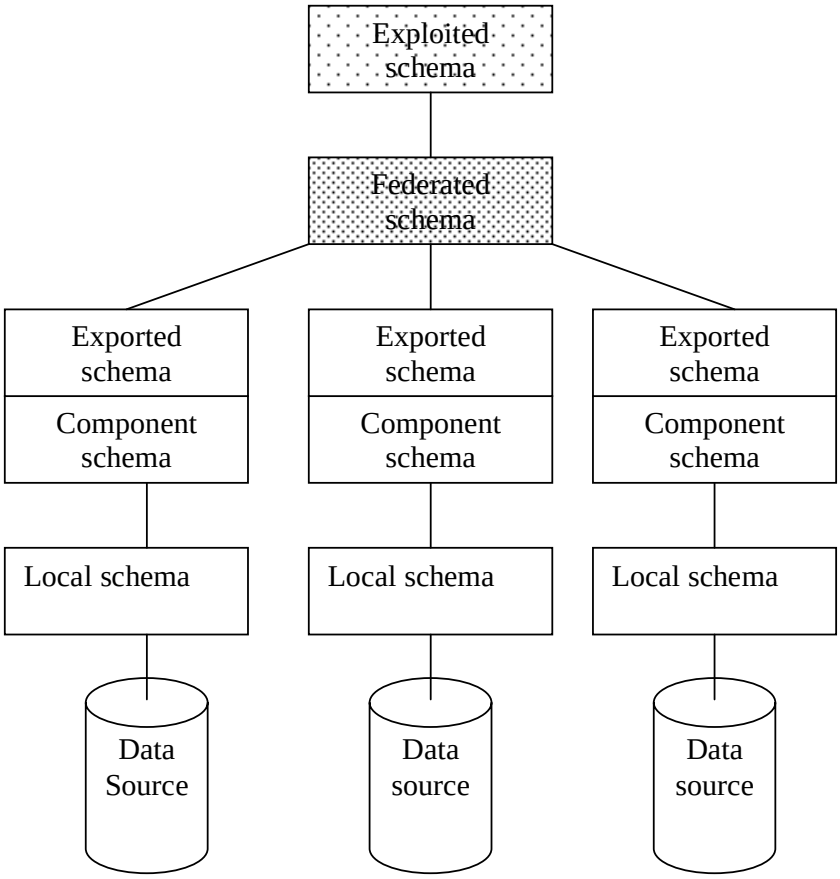


Figure2. Federated Information System [Leser00]

In the traditional environments composed of several databases, the distribution and the integrity are managed by a total supervising authority (requests, transactions, etc.) [Grefen96], however, taking into account the complexity of the various autonomous systems represented in the federated information systems, this global solution is not easily maintainable. The centralized index can also represent an additional difficulty because its structure can become extremely complex according to the heterogeneity of the sources and the numbers of users [Leitzelman98].

2.2.3- Agent based information systems

The Agent based information systems (AIS) are mainly used to afford the maximum of flexibility, interoperability and adaptation to information systems. Like the FIS, AIS support the inter-organizational connection in order to share the distributed data of various subsystems.

An AIS is composed of one or more intelligent agents allowing the data processing within an organization. An agent, or mediator, is a software module, which decodes

the informational contents included in a data set (databases or wrappers) in order to form exploitable information by the applications [Thiran99]. It has three functions: informational, integration and structural function. It must manage the contents of the data and their heterogeneity, integrate relevant information for the organization and simultaneously regulate the conflict problems between the various sites. An agent is characterized by its autonomy, its ability to anticipate and to react to an environmental variation and to have a social and cooperative behavior with the other agents [Murugesan02] [Chong00]. It is conceived to act in an independent way and with an aim of satisfying the specific users needs. However, it also can, to improve the quality of service, use the assistance of other mediators. An agent has a moreover capacity of adaptation to the environment in which it evolves. Indeed, it carries a certain "knowledge" related to its field of action making possible its adaptation to the needs of the various users.

Another important component of AIS is the wrapper; it is used as an intermediary between the data model and the mediators. Initially, the mediators' queries are transmitted to the wrappers, which communicate with the sources and release relevant information. Then, this information is exported to the mediators. The wrappers have a double role of interfacing between the sources on one side and the agents on the other. They include a diagram that is clean for them with an aim of having an informational structure in order to answer the requests of the mediators. These diagrams constitute a first total model of the data. The mediators are also required to translate the queries in an executable language by the wrappers.

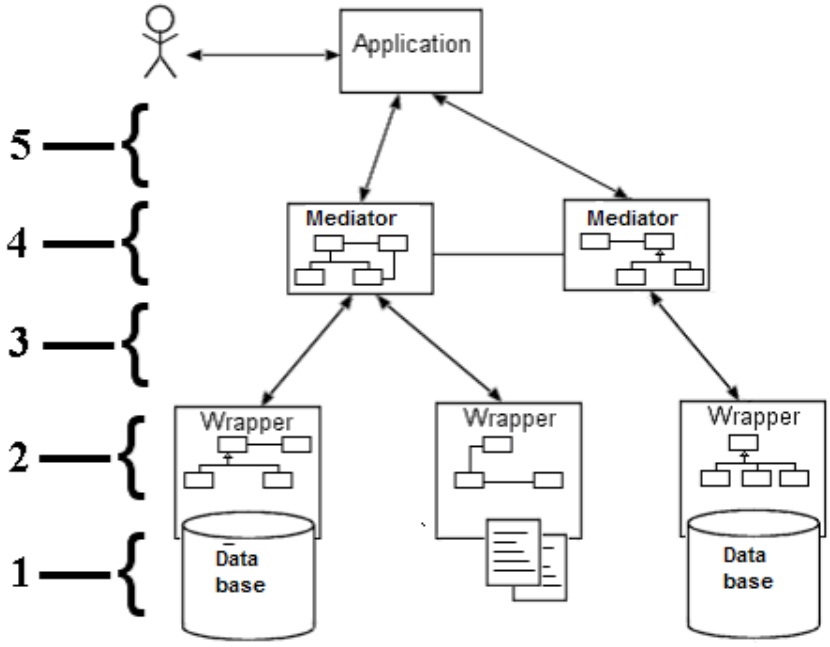


Figure3. Agent Based Information System [Chong00]

Generally, the AIS have a structure distributed on five successive layers, presenting some similarities with the FIS (see Figure 3). The five layers of the AIM structures are the following:

- The first layer constitutes the data model where various information are represented.
- The second represents the wrapper schema containing the diagrams of data integration.
- The third layer is the set of database local diagrams. The whole of these diagrams constitutes the first global data model.
- The fourth layer is composed of mediator's schemas being used as intermediary between the application layer and the wrapper to translate queries.
- The fifth layer contains mediator's diagrams, which constitute the global AIM diagram with which the user will interact in a transparent way.

Complex information systems, like the FIS, the DIS… can use agents in order to facilitate the management and the automation of some of their tasks. However, the integration of these systems can be more difficult. Indeed, the AIS, are relatively poorly formalized and their design methodology still requires many improvements [Debenham02].

2.2.4 Distributed Information System

A distributed Information System (DIS) is a system made up of at least two subsystems that cooperate with each other. Unlike the various information systems described previously, the DIS is realized from a single data diagram with the aim of distributing it. The distribution is carried out mainly thanks to three levels [Snene04]:

- The data level: the data are distributed between the various sites and still reachable by the set of sites. The distribution is thus done in agreement with the total conceptual model.
- The application level: the different required functions are distributed between different sites which are managed by a common interface accessed by the various users in order to establish a restricted access according to their competences.
- The control level: the functions of controls are mainly represented by the integrity constraints. They can be distributed between the various sites containing the treated data or on a unique site dedicated to these constraints. The choice is made by considering the quantity of data flows that the distribution of the functions of controls generates.

In an evolutionary approach, the distribution of this kind of information system is extremely complex, since we need to take into account the various levels necessary to meet the users needs. These different levels have to be flexible enough to permit a system structure modification by modifying one or more subsystems. Indeed, it is essential to be able to add new subsystems in the global DIS and to modify those already existing without influencing the other subsystems [DeWeger99].

The DIS is mainly based on distributed databases (DDB) which are defined like a collection of distributed data bases through a network containing sets of logically dependent data according to the relational model [Ozsu94]. The approach commonly used in order to design a DIS is a Top-Down approach. Indeed, we are initially interested in the analysis of the needs and in the design of the global database diagram by including various semantic spaces that the subsystems represent. Then, we focus on the physical data distribution by integrating it in the various subsystems.

For the design of the DIS, several phases compose the Top-down approach:

- It begins by the system needs analysis and the extraction of the potentially useful metadata and processes by the system users.
- Then begins two continuous levels: the view design and the conceptual modeling. The first defines the user interfaces whereas the second examines the organization in order to discover the types of entities to be conceived, their fundamental relations and functions which compose it. According to these elements, it is possible to extract needed information to fix the functions based on entities. The development of these two parts is carried out then by regularly checking that they function in an optimal way [DeWeger99].
- The schema distribution design aims to model the local conceptual diagrams specifying the entities to be distributed on the various sites of the DIS. At this level, the entities are aggregated in the form of fragments, which are distributed thereafter. The fragmentation and the allocation phases are mainly executed at this level.
- Finally, the design process defines the physical locations of the various sites on which the conceptual diagrams obtained previously are projected.

In spite of the existing collaboration between the various sites, each site preserves a certain degree of autonomy. This autonomy is reinforced through the specification of the integrity constraints, which are used to manage the operations between different sites. [Ozsu94] explains that the problems on which it is necessary to concentrate are the development of distribution methodologies as well as the integrity constraints development as tools for the information system development inducing a clear separation between the different semantic spaces that represent certain forms of organization knowledge.

2.3- Comparison criteria

In this section, we present the different common criteria based on the different presented architectures. These criteria are the most important ones because of their effect on the possibility of the system evolution and the ease of system maintenance.

2.3.1- Heterogeneity

The heterogeneity of the Information systems can be observed on three various levels [Busse99]:

- Logical: on this level, we differentiate semantic and syntactic heterogeneity. In the first case, we focus on the concepts conveyed by the attributes; the most common problem encountered is the homonyms or synonyms that can generate mistakes in interpretations. The syntactic conflict arises when a given data-modelling concept is expressed differently using data modelling constructs provided by a data model. It appears also in the differences in structure or representation of semantically equivalent information (real world objects).

- Physical: Data can be of different natures and can take different forms. The diverse existing data support must be easily integrated in the global information system. The physical heterogeneity of the information system is the success factor of a successful evolution and integration of the different existing subsystems in the global IS.

- Technical: here is implied the multitude of average materials or software at disposal for the design and the implementation of the information system. This gathers the various platforms and operating systems. But it is also necessary to consider the interfaces, the languages (of requests), the restrictions of the requests as well as the access methods, for example the protocols (HTTP, JDBC, CORBA) and safety.

Table 1 (see Table 1), shows how the heterogeneity is handled in the different architectures presented on the previous section.

		Heterogeneity	
	Logical	Physical	Technical
DispIS	Messaging services enables the communication	It is ensured by the subsystem autonomy.	Implies the use of bridges
FIS	Ensured by the interactions between the 2nd and the 3rd level	The first and the 2nd levels manages the different data	The FDBMS control and federated the different databases
AIS	The wrappers manage the exported data.	Ensured by wrappers.	Wrappers and mediators manage the different data exchanges.
DIS	Ensured at the design step	Ensured at the design step	DDBMS control the distributed data

Table1: Heterogeneity comparison grid

2.3.2- Distribution

The distribution is considered as an important comparison criterion in our research. In fact, the decision of data allocation is generally taken under many constraints. These constraints are various and spread out from the needs expressed by the system users and the site dedicated technology to the Quality of Services (QoS) and the network answering time. The distribution criteria is mainly observed on three different levels [Snene04]:

- *Data distribution*: data is distributed over the different existing subsystems. The existing cooperation between them needs a high data harmonization level. It is clear that a certain degree of virtual homogeneity is needed to guarantee the treatment and the data quality.

- *Transaction distribution*: it can be classified in two main levels. The first is the transaction call. It represents the process starting. The localisation of the call must be well defined in order to prevent the concurrency problems and to maximize the execution answering time. Then the execution level starts. At this level the execution can be distributed or located on a unique site. It depends on the process-needed resources and on the available site resources.

- *Integrity Constraints:* the transaction propagation represents the last level. The data update propagation must be defined while respecting the data integrity constraints and the uniformity and homogeneity of the database contained information.

Table 2 (see Table2), shows the different manners and methods in which the distribution is handled in the different architectures presented in the previous section.

		Distribution	
	Data	Transaction	Integrity constraints
DispIS	Data is distributed before the integration. High redundancy risk	Every subsystem executes its transactions	Local IC. Transgression of global data can occurred
FIS	Is managed by the central control model	Subsystems are responsible of transactions	IC are designed separately but federated
AIS	Wrappers and agents manage the distribution and the data integration	Transaction distribution is related the specified agent	IC are distributed according to wrappers
DIS	Data are distributed according to users needs and system performance	Transactions are distributed according to system performance	IC are designed for the distributed schema

Table2: Distribution comparison grid

2.3.3- Autonomy

Autonomy takes place on several levels within the framework of multi-site information systems:

- Design autonomy: a subsystem is conceived independently from the others. The data model is independent and the subsystem remains autonomous for further evolution.
- Communication autonomy: a subsystem decides by its own with which subsystem it wishes to communicate. It also has its own mode of communication
- Runtime autonomy: the subsystem decides when and in which way it carries out a request.

Table 3 (see Table 3), shows how the autonomy is handled in the different architectures presented in the previous section.

		Autonomy	
	Design	Runtime	Communication
DispIS	Sub systems are designed independently	Subsystems decide independently	Subsystems decide how to communicate
FIS	The federation is done upon existing system	The execution is requested by the mediator	The federator is responsible for the communication
AIS	Design is partially autonomous	Agents preserve the runtime autonomy	Agents decided for its own communication
DIS	Subsystems can be designed independently or in a global schema	Runtime is autonomous	Partially autonomous

Table3: Autonomy comparison grid

2.3.4- Evolution

The system evolution is generally carried out through three different levels: Integration, Communication and Integrity. In fact, to facilitate the system evolution, the integration of the new subparts must be well structured and easily performed. The communication between different subparts must also be clear to pave the way for the new communication establishment between either the existing subparts or the new subparts. Finally, establishing the evolution of integrity constraints within the different subparts allows the evolution of the global system considering that integrity

constraints inhibit any operation or transaction modification until its adaptation to the new context.

Integration includes the concept of conceptualisation, i.e. the chosen approach to conceive the information system. It is necessary to find stability among a light conceptual model, i.e. not very complex and consequently easily realizable, and complete, i.e. answering all needs but with a higher level of complexity. At this stage, the concept of user transparency appears. That means that user uses only one system interface whereas the used architecture attends the totality of the systems, i.e. all the subsystems [Snene04].

The communication is a fundamental element between the different subsystems of an information system. Indeed, the communication between the various systems can cause problems which can harm the global system performances if the communication is not well managed or if the systems are not able to understand the different queries and answers formulated between them.

The integrity is ensured by integrity constraints. It represents a defined situation based on one or more fields, and their validation is carried out in an algorithmic way. The context of an integrity constraint indicates the fields concerned in its definition and its validation. The condition of the integrity constraint must be checked for any state of the information system or any modification of state of the information system. The range of an integrity constraint indicates the whole of the primitives of modification of categories, which must validate the integrity constraint, to preserve information coherence. The answer of an integrity constraint indicates the actions to be undertaken in the event of invalidation. The expression of an integrity constraint can be a mathematical expression, a comparison of sets, an automat, and a predicative expression on categories of the context of the integrity constraint or an algorithmic expression finished by a condition. Table 4 (see Table 4), shows how the evolution is handled in the different architectures presented in the previous section.

	Integration	**Communication**	**Integrity**
DispIS	Bottom up	Restricted by messaging protocol	No integrity constraint between subsystems
FIS	Bottom up	The federator is responsible for the communication between different subsystems	Integrity constraints are mainly dedicated to control the global schema and not the integrity of data between subsystems
AIS	Top down	Only agents can communicate between each other	Agents and wrappers manage the integrity constraint
DIS	Top down	Communication is based on subsystems needs	System design based on integrity constraint

Table4: evolution comparison grid

2.4- Comparison grid

With the vision of the diverse obtained results, we note certain similarities among the different architectures. A first similarity can be denoted between dispersed architectures and federated architecture. This resemblance is due to the nature of the needs that these architectures propose to fill and to the manner with which these architectures proceed. Indeed, the federated architectures represent an evolution and a continuation of the first architecture. It replaces the inter systems messages by federation diagrams which proceed in the same way that the preceding messages for the collection and the processing of decentralized data. These two architectures are centred on concepts of heterogeneity, distribution and autonomy. These criteria are the most important and needed ones to be satisfied among different subsystems that have to be integrated. In fact, these criteria ensure to each subsystem the maximum of autonomy for its transactions execution and for the queries answering. The others criteria give developers and designers the ability to integrate different kinds of subsystems realised in different technologies with the minimum of distribution constraints.

The second similarity appears between the agent based architecture and the distributed architecture. The evolution of technologies and of the networks performances, made it possible to have new modes of integration more intelligent and less constraining than the federated architecture and dispersed architecture. Indeed, agent based architectures are very flexible and evolutionary considering the facility to add new agents between the various existing systems without having to reorganize the totality of the existing architecture. The major problem of this architecture is conceptual. Indeed, from a high number of agents the control of the data flow circulating through the system becomes extremely hard to manage and especially to maintain. Moreover, the design of subsystems is generally made independently from each other; some problems of overlap and inconsistency are frequently detected. The last architecture answers the various suggested criteria and allows the design of the system distribution before implementing it. The advantage of this architecture is that it makes possible to draw systems from scratch while taking into account the final distribution aspect. These two systems prove, indeed, being most favourable for a later evolution of information system. In fact, the evolution of an information system depends on the chosen architecture and technologies. Architectures imply the manner of design and of integration between different subsystems. It is clear that in the case of existing subsystems, designers do not have enough range of choice to take into account the different presented criteria due to pre-existing system constraints. Throughout this work, we will use the term ***distributed information system*** to refer only to this architecture. Indeed, we consider the distributed information system as a system build from scratch and not from the integration of different existing subsystems. In fact, the subsystems integration step is not considered as a distribution design step due to the fact that these subsystems are already distributed.

The architecture comparison grid represents the global architecture quality by referring to the different criteria given above (see figure 4). Every criterion is divided into five

levels: unavailable, deficient, medium, suitable, excellent. These levels start in the middle of the grid with the unavailable mention and finish on the border with the excellent mention. The level assignment is based on the different synthesizing tables presented below.

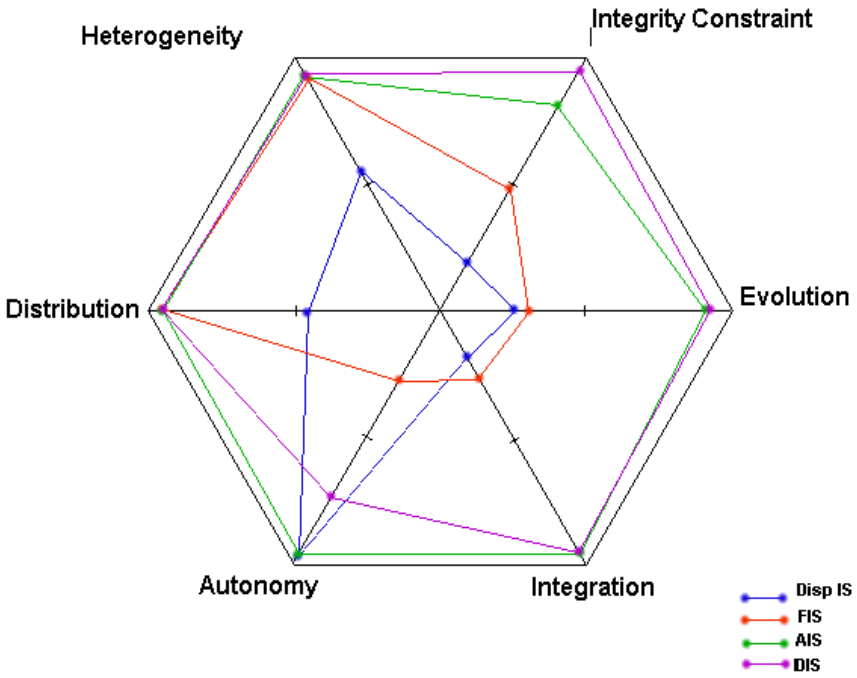


Figure 4. Architecture comparison grid [Snene04]

Next chapter presents and discussed the different existing approaches for distributed information systems design. These approaches have a definition of distributed information systems closer to the proposed definition in this chapter. Mainly, we distinguish two approach categories. The first is the component based approach. The second is a data based approach. These approaches contain different methods and have different level of formalism. We present the most used ones.

Chapter III
Distributed Information system design methodologies

This chapter presents the different existing approach for the distributed information system design. Basically two main categories can be distinguished. The first is the component design approach. It is based on the decomposition of the global conceptual diagram of an information system in a set of functional components that are extracted according to their business aspect. Indeed this decomposition takes into account the methods provided by each object of the component to assemble them and to build a component that provides a set of complementary and homogeneous functions. The second is the data oriented approach. This approach is based on the use of the distributed database. The distribution is done by a process of allocation and fragmentation of the information system database schema. This distribution is done according to predefined algorithms [Özsu97].

The chapter is further structured as follows: the first section introduces the concept of information system design and the main existing approach, then it defines different existing distributed environments. In the second section we establish a state of the art of the different methodologies of distributed information system design beginning with the component oriented approach and then with the data oriented approach. The last section includes a conclusion and a comparison between these different approaches.

3.1- Introduction

It is impossible to consider any organization without including, understanding and solving its information requirement. An operational and robust information system must provide the right information to the right person at the good time with the lowest possible cost. This information must be able to be provided at all management levels from the operational until the strategic.

An information system must include tools, techniques, and concepts of various disciplines such as data processing, the management science, and the behaviour of organization. These interdisciplinary tools, combined with a comprehension of the basic needs of an organization, make it possible for the professional of information system to apply data processing to the solutions of different commercial problems. The Information system cannot be considered as an isolated system but as a set of abstraction level interactions forming the entire environment of the information system (see. Figure1).

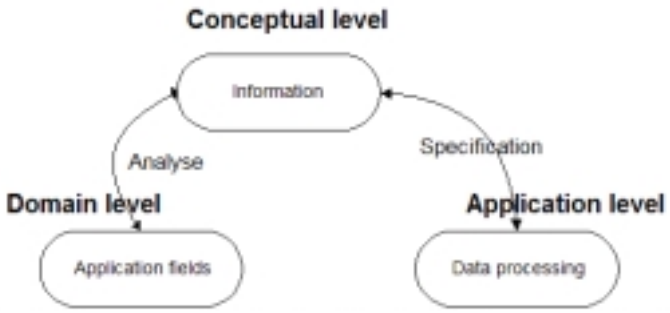


Figure1. Information system design levels

Analysis phase extracts domain related information from the global organisation environment. This information is structured into two levels: data level and domain process level. At the data level we focused on the information that has to be collected, treated, stored and maintained, resulting generally from the organisation environment or from its external environment [Bubenko94]. At the domain process level, we get domain functions and methods for data processing and domain specific consistency constraints. These constraints represent the interaction façade between data and functions. Dynamic aspects of information systems can be managed by changing, adding or disabling constraints unless the evolution affects data or methods. So, conceptual modelling plays the central role among the different steps in information system design. The result of the modelling process is a comprehensive and first formal description of the part of the domain to be modelled [Leonard92].

3.2- Distributed environment

Mainly two different distributed environments can be distinguished. The first environment is data based and called *distributed database*, the second is based on

process and it is known as *distributed system.* Coupling these two different environments gives us the ability to build the *distributed information system.*

3.2.1- Distributed database

A distributed database is a set of several, logically interconnected databases distributed on a computer network. A distributed database can be defined as consisting of a compilation of data with varied parts under the control of separate distributed database management systems running on autonomous computer systems. All the systems are interconnected and every system has autonomous processing potential serving restricted applications [Özsu91]. Every system participates in the execution of one or more total function. Such applications necessitate information from more than a single location. The distributed environment of the database is unknown to users and this transparency is visible in different ways. Although there is a number of advantages in using a distributed database management system, there is also a number of problems and implementation issues [ÖZSU85]. Finally, data in a distributed database management system can be either partitioned or replicated or both.

The distributed database index has to identify the site(s) where information is being stored in addition to data in a system list in a centralized database management system. Because of data partitioning and replication, this additional information is required [Barbara82]. There is a number of approaches to realize a distributed database index (see Table1).

	Locality of reference	**Reliability and availability**	**Performance**	**Storage costs**	**Communication costs**
Centralized	Lowest	Lowest	Unsatisfactory	Lowest	Highest
Partitioned	High	Low for intern; high for extern	Satisfactory	Lowest	Low
Fully replicated	Highest	Highest	Best for read	Highest	High for update; low for read
Selective replication	High	Low for intern; high for system	Satisfactory	Average	Low

Table 1: Comparison of strategies for data allocation

- Centralized: data is distributed over the network, but the index still unique and localized on one site. Any data access query must be forwarded to this site to get an answer. The advantage of such architecture is that it prevents from data incoherence and that keeps an optimal database integrity. The disadvantage of this architecture is that it slows down the answering time and which can cause some bottlenecks [Kazerouni97].

- Fully replicated: an index copy is replicated on each database site. Each site, still independent from others to answer queries about its local data. In the case of query about distributed data, the query is first locally treated and then the site is responsible for asking other sites about their local data. The advantage of this architecture is that a faster answer in the case query about local data and an appreciably fast answer in the other cases are provided. The disadvantage of this architecture is the integrity violation risk and the redundancy cost [Karlapalem96].
- Partitioned: in this case, the index is partitioned and replicated according to usage patterns. Each index points the affected data site. The different site index have different pointer between them to ensure the function continuity. The advantage of this architecture is that every index can be used as a standalone index for its site. In fact, the data indexed is the local data. The disadvantage of this architecture is the risk of incoherence within the different database index. In fact, the replication of different data on different site can be a major risk if the update done on these different indexes is not done in a synchronous way. The synchronization of the update of these indexes is done by stopping all accesses to the different databases and which can generate a long access waiting time [Apers88].
- Selective replication: this architecture is a mixture of the different solutions presented above. It can be done in different manners. The global databases index is spitted in one or in many partitions. Some sub-indexes will be placed on their correspondent site. The others can be fully replicated or partitioned in different sub-indexes. This architecture is the most flexible one and can guarantee a good answering time. The disadvantage of this architecture is the low level of its data integrity. In fact, data index will be more complicated to manage and to guarantee a correct update over the different site [Ceri83b].

A distributed database management system should offer a number of features which make the distributed nature of the distributed database management system transparent to the user. These include the following [Wilson86]:

- Location transparency: command used to perform a task is independent from the location of the data and the location of the system where the command is issued.
- Replication transparency: copies of data are stored at multiple sites for better availability, performance, and reliability. Such transparency makes the user unaware of the existence of the copies.
- Performance transparency: users must not note performance degradation due to distributed architecture. The distributed database management system must determine the most cost-effective strategy to execute a request.
- Transaction transparency: ensures that all distributed transactions maintain distributed database's integrity and consistency. In fact, each distributed transaction accesses data stored at more than one location. It means that the distributed database management system must ensure the concurrency transparency. All transactions must execute independently and be logically consistent with results obtained if the transaction is executed in some arbitrary serial order.

- Catalog transparency: once a name is specified, the named objects can be accessed unambiguously without additional specifications.

3.2.2 - Distributed systems

Nearly all large software systems are by necessity distributed. For example, enterprise-wide business systems must support multiple users running common applications across different sites. A *distributed system* encompasses these applications, their underlying support software, the hardware they run on, and the communication links which connect the distributed hardware. The largest and best-known distributed system is the set of computers, software, and services comprising the World Wide Web, which is so pervasive that it coexists with and connected to most other existing distributed systems. The most common distributed systems are networked (client/server) systems [Vissers91].

Distributed systems have these defining properties [Wu98]:

- Multiple nodes: Software for the system and its applications executes on multiple independent computers. These nodes may range from information appliances to personal computers to high performance workstations to file servers to mainframes to supercomputers. Each may primarily take the role of a *client* that requests services by others, a *server* that provides computation or resource access to others, or a *peer* that does both. A minimal distributed system may be as small as two nodes provided that software connectivity is present.

- Resource sharing: The most common reason for connecting a set of computers to operate as a distributed system is to allow them to share physical and computational resources; for example files, databases, mail services, stock quotes, collaborative applications, and so on. Distributed system components that support resource sharing play a similar role as operating systems from which they have increasingly become indistinguishable.

- Concurrency: Each node in a distributed system provides independent functionality, and operates concurrently with all of the others. More than one *process* (executing program) per node, and more than one *thread* (concurrently executing task) per process may participate as components in a system. Most components are *reactive*, continuously responding to commands from users and messages from other components. Systems as a whole may concurrently execute a number of related applications, all relying upon common infrastructure software establishing system-wide policies, protocols, and services. Like operating systems, distributed systems are never designed to terminate, and so should always remain at least partially available.

Distributed systems also possess, to varying degrees, the following characteristic properties [Tanenbaum02]:

- Heterogeneity: The nodes participating in a system may consist of diverse computing and communication hardware. The software comprising the system may be written using diverse programming languages and development tools. Some heterogeneity issues are addressed by agreeing upon common message formats and low-level protocols that can be readily implemented across different platforms. Others may require construction of *bridges* that translate one set of formats and protocols to another. More thorough integration can be attained by requiring that all nodes support a common *virtual machine* that processes platform-independent program instructions.
- Persistence: At least some data and programs are maintained on persistent media that outlast the execution of any given application. Persistence may be arranged at the level of file systems, database systems, or programming language run-time support mechanisms.
- Isolation: Each component is logically or physically autonomous, and communicates with others only via structured message protocols. In addition, groups of components may be segregated for purposes of functionality, performance, or security.

3.2.3- Distributed information systems

A distributed Information platform consists of autonomous parts, independently functioning, but interacting parts. It is composed of two or more distinct Information subsystems deployed on different locations and which collaborate with each other. Each subsystem is able to process locally stored data. Data, stored for remote access or for centralized maintenance purposes has to be stored according to a global conceptual schema, on which a common database schema is designed [Dale98]. The subsystems of a distributed Information system generally share common resources. Cooperation is based on shared data. The consistency, integrity and availability of data are vital to the distributed Information system.

The design and the architecture of a distributed Information system are guided by some characteristics that make them different from a centralized Information system or distributed system. The most important characteristics of Distributed Information System are [Denker99]:

- Autonomous parts: the Distributed Information System is composed from different independent parts but which interact. The interactions between these parts represent the data flow inside the global system.

- Logical distribution: designers need to establish a clear logical separation in the design of the different parts of the Distributed Information System. This separation is based on functional aspect of the system. This distribution implies the packaging of each independent subsystem in an independent and fully functional subsystem. This level of distribution is usually done without taking in consideration the technical aspect or the needed deployment effort.

- Physical distribution: the components of Distributed Information System can be deployed either on common or different locations. This distribution is guided by some constraints such as performance, availability, answering time and the kind of performed transactions. This step is carried out by the system developer. The given design schema done by designers is often modified due to technical constraints.

- Functional integration: the different parts of Distributed Information System present different functionalities that are independent but in interaction between each other. Adding or deleting parts from the Distributed Information System must not disable or disturb the normal working of other parts.

- Interoperability: the different parts of the Distributed Information System can be developed with different technologies and deployed on different application servers and operating systems. Designers must take in account this kind of diversity and insure the interoperability of the parts of the Distributed Information System by avoiding specific services offered by specific platforms in design phase.

3.3- Design methodologies

Different methodologies of Information system design exist. These methodologies can be informal, semi formal or formal according to the used language. Methodologies' classification corresponds to the language classification according to their formalism level, their syntax and semantics [Fraser94].

3.3.1- Informal design

An informal design is built on natural language with or without structural rule. Its use introduces some ambiguity because its syntax and its semantic are not well defined. To reduce these risks, many approaches have been proposed. One of these approaches is to reduce the redaction to fill predefined forms. Another approach is to restrict the used natural language. This approach is commonly known as "controlled natural language" [Dupuy00].

The informal design methodologies have a main advantage which is the ease of use and of understanding between the different participants in the Information System development. It is often used as a complement of semi formal or formal design

methodologies. The difficulties of use of informal design methodologies are related to the scope of the natural language and specifically to the ambiguity that can be generated from its use. These ambiguities render imprecise every reasoning for analysis or verifying the specification [Fraser94].

3.3.2- Semi formal design

The semi informal design methodologies are generally based on graphical language, which has a well-specified syntax, but a weak semantics. The semi formal methodologies can be classified into three main categories:

- Cartesian methods [Jackson83],
- Systemic methods [Bodart83],
- Objects methods [Jacobson92].

3.3.3- Formal methodology

Formal design languages are based on mathematical notations that are precise and clear. They can be grouped under these different categories:

- Model-oriented languages: these languages are based on state notion. They express an explicit definition of the state of the system by building a model in mathematical terms of structures such as the sets, the functions or the predicates. Different languages exist such as Z [Spivey92], and B [Abrial96]. For example in Z language, a schema grouping variables with predicates represents the system state. A new trend of oriented model languages is objects formal languages. They introduce a structure, a class, that regroup the data structure and the operations set that are executable on the object. These language categories are well equipped with builders [Sommerville92], enabling them to write well-defined specifications. At the same time, this made them more difficult to learn and to use.

- Properties oriented languages: the system is described with a set of properties under an axiom set form that have to be satisfied by the system. Two main categories of these languages exist. The first one is axiomatic language. This category is inspired from Hoare research on the pre and post condition in the first order logic used for the operation specification [Hoare74] [Hoare73]. The second category is algebraic language. In an algebraic language, axioms are used to specify the properties of a system, but these axioms are reduced to equations [Place90]. An algebraic specification is made up of a signature and a whole of axioms that are logical formulas. The signature includes the names of the types and the operations defined by their profile [Backus78].

Thanks to the formal aspects, properties can be established by reasoning. Then it becomes possible to control the process leading of the specification to the program and to show that an establishment in a program is in conformity with its specification.

Nevertheless, formal methodologies are still difficult to learn and to use. Due to that, these methodologies are less used than others, especially to their lack in the relationship between designers and final users.

3.4- Distributed Information System design

The maturation of database management system (DBMS) technology has coincided with significant developments in distributed computing and parallel processing technologies. The final result is the emergence of distributed information system based on distributed database management system and distributed applications system. The integration of workstations in a distributed environment enables a more efficient function distribution in which application programs run on workstations, called applications servers, while database functions are handled by dedicated computers, called database servers. A distributed information system is a collection of data, transactions, integrity rules and sites. Data is concretised by a set of multiple, logically interrelated databases distributed over a computer network. This set is managed by a distributed database management system, which makes the distribution transparent to the users [Özsu99].

Nevertheless, methodologies for the design of information systems have usually paid modest consideration to distributional and communicative characteristics of information systems. However, recently this situation has improved: information systems have noticeably grown in dimension and range, organizations want many of their separate information systems to be integrated and consequently the number and the variety of geographically distributed users of these system have become large. The distributed systems community has produced methodologies for the design of distributed systems [DePaoli91] [CCITT88]. However, these methodologies pay little attention to the information aspects of distributed information systems; instead their strength is in the distribution and communication aspects of these systems [Bolognesi92].

New architectural and technical forms of information systems add a more significant level of complication due to the decentralization of the distribution, treatment and data. These architectures increase the deployment and the execution possibilities thanks to theirs offered under structures. Indeed, the simple separation of the various functional levels as we made it in a classical architecture (Data, Treatment, Presentation) is not enough and the choice of the site of deployment and/or running becomes capital for the optimisation of the output of the system [Denker99]. These decisions of distribution were generally made during the implementation phase. The conceptual structures offered to designers, to allow them express their needs for distribution (concepts of packages, business component,..) and for composition did not match with the rules used by developers for building their distributed components [Yilmaz02]. Indeed, business components represent a single and autonomous concept of real world. They encapsulate all the data concerning this concept including name, goal, behaviour and all other information with regard to them. A business component is a set of

objects, which can be physically deployed on two or several sites [Pawlan00]. It is usually made up of one or of several distributed components, which offer together the various aspects of distribution, necessary to the business component.

Many problems exist at the design step of the distributed information systems. The first is how to guarantee the maximum functional continuity if a sub-system is disconnected from the global information system. The second is how to minimize the data flows over different sites and to maximize the integrity of the global system.

Business components have been presented as a possible solution to resolve these problems. This concept is based on the notion of the business partitioning. It is still an informal process and do not take into account the real constraints of Distributed Information Systems (performance, Integrity, QoS,). In fact, information stored, maintained and used into the DIS has to be consistent to preserve the integrity and the correctness of the system. Integrity rules present enough robustness and usability to insure the data consistency. The main problem is that the evolution of this kind of system implies the operation of set of modification on integrity rules and their adaptation to the new information system. The set of integrity rules expressed separately in each system before coupling them, have to be verified, validated and optimised to avoid logic duplication, contradictory rules and database inconsistency [Bblakely89]. Also, building DIS offers a various set of deployment site for integrity rules. Deciding where to validate this kind of rules cannot be done at the development process but both in analysis and design steps. To achieve it, we need to enrich the conceptual model of a distributed information system by indicating the integrity rules, mode of validation and site of deployment.

3.4.1- Existing DIS design approach

Various approaches for the modelling of a distributed information system exist. These approaches are mainly different at the level on which the distribution is operated. Concretely, the design can be done at the organisational level applying a traditional approach such as object design approach, primarily based on concepts such as the objects' components or the packages in UML [Akehurst99]. The second approach is data oriented. Distributed databases approaches [Özsu99] are generally carried out with special interest to the data fragmentation data allocation over different existing sites. These two approaches remain in spite of their difference, very complementary. Indeed, the first approach collects the necessary knowledge to the good succession of different functionalities needed by the system. It also helps to set the functionalities on different distribution sites according to the specified needs of each site. The second approach is more dedicated to the optimisation of the data allocation over the different network sites. This allocation is carried on to minimize the system time answering while maximizing the Quality of Service (QoS).

3.4.1.1- Component oriented methods

In view of the importance of the component paradigm, most modern methods aim to accommodate them in one form or another, but, generally, components are viewed as just a convenient implementation tool rather than an integral part of the overall software development cycle. The different approaches, which focus on the use of interoperable components, are able to reduce the complexity and the costs involved in development. They also allow the reuse of these components in other information systems [Espindola04]. However, the analysis of the main component oriented methods reveals important limitations, particularly related to component distribution issues. These methods focus on the development of independent parts (components) as well as on the composition, the distribution and the interoperation of these parts [Waters99]. Many industrial and academic proposals of component oriented methods can be found, such as UML components [Cheesman00], Catalysis [D'Souza98], RUP [Kruchten99], Select Perspective [Allen98] and CADA [Boertien04].

We are going to present the three main existing approaches that regroup the main characteristics common to the different component oriented methods. These methods are Catalysis, UML and CADA.

3.4.1.1.1 - Catalysis

Catalysis [D'Souza98] is one of the initial methods developed to influence the UML in connection with component-based development, and embraces many reuse technologies including architectural styles [Colin02]. The method either introduced or popularised many ideas that today are considered natural ingredients of component based development, and many of them have been explicitly adopted in the UML [Waters99]. Catalysis is ordered in terms of three primary dimensions, called "modelling concepts", "levels of modelling" and "principles". Each of these dimensions, is, in its turn, composed of three further basic ideas. The three main modelling concepts are:

- Collaborations: a set of abstract interactions between objects which play certain roles with respect to one another.
- Types: an abstract definition of an object's externally visible behaviour.
- Refinement: a relationship between two descriptions of the same thing but at different levels of detail.
- To this list of modelling concept is added the notation of a framework. It is essentially composed of potentially reusable models captured in a generic way in the form of package.

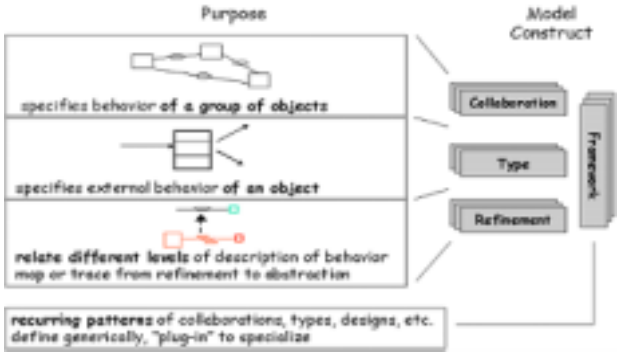


Figure2. Catalysis Business modelling level [Winter03]

In the secon

- Problem domain: the outside of a system which describes the concepts of relevance to the system users; this capture the environment or context in which the system is expected to operate.
- Component specification: the boundary of a system or component that captures externally visible behaviour.
- Component implementation: the interior of a component, which describes how it is constructed from smaller parts.
-

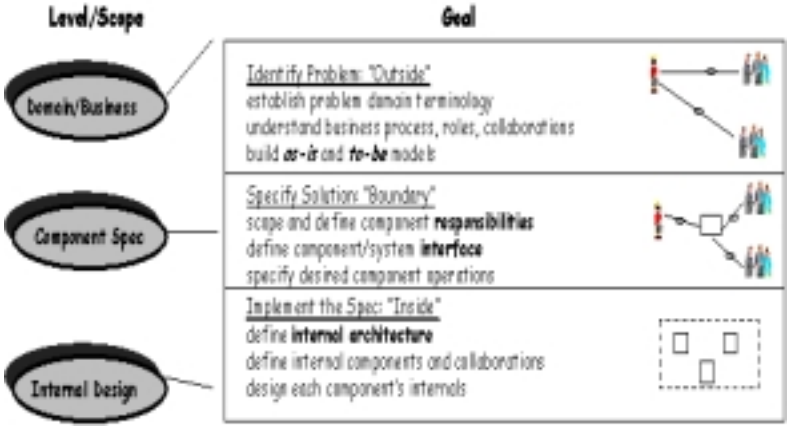


Figure3. Catalysis component modelling level [Winter03]

Finally, in the third dimension of concern, three principles are defined to support the development effort:

- Abstraction: the separation of the relevant aspects of a concept or artifact from irrelevant detail
- Precision: the development of abstract descriptive artefacts that make concrete, accurate, and testable statements.
- Pluggable parts: adaptable software building blocks that can be connected and used (together) with minimal effort.
-

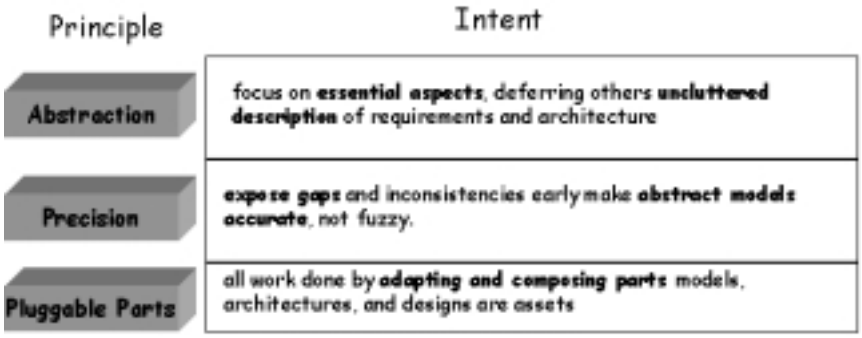


Figure4. Catalysis internal modelling level [Winter03]

3.4.1.1.2 - UML

The UML components method adopts an architecture that separates the interface, which includes presentation and dialog, from the application semantics, which involves business and system services. Such separation is referred to as client and server, but the method does not address distributed applications [Espindola04]. UML components are units of software structured according to some specific principles. The fundamental principles they adhere to are actually the fundamental principles that underpin object technology [Cheesman00]:

- **Unification of data and function:** A software object consists of data values (or state) and the functions that process those data. This natural collocation of dependencies between function and data improves cohesion.
- **Encapsulation:** The client of a software object is insulated from how that software object's data is stored or how its functions are implemented. We say that the client depends on the object specification, but not on its implementation. This is a very important separation of concerns conceived as a key to managing dependencies between software and reducing coupling.
- **Identity:** Each software object has a unique identity, regardless of state.

UML components address the specification of the architecture, the design of the existing dependencies between components, and the management of these dependencies. It emphasizes that the main objective, with regard to components, is its capacity to support changes. Thus, a component must present as main characteristic the ability of being substitutable, and the application architecture must pave the way for the management of the overall application [Waters99].

The method identifies two main phases: the requirements workflow which captures the basic needs that the system must fulfil in terms of use-cases and high-level business classes. The second phase is the specification workflow which documents the business types, interfaces, and components that have been chosen to satisfy these requirements.

3.4.1.1.3 - CADA

CADA is the COMPAS analysis and design approach developed at the Ordina institute for research and innovation. CADA is not developed from scratch, but incorporates best practice methods and techniques from Catalysis, UML and DSDM (Dynamic Systems Development Method). The CADA concepts largely correspond to those of Catalysis. However, CADA incorporates more concepts from the UML implementation diagrams and adds a number of concepts for business modelling [Boertien04].

One of the central ideas behind the CADA methodology is that a method should support analysts and designers. Therefore, CADA provides guidance for the various design stages, but without prescribing the precise order in which the steps are to be taken.

CADA distinguishes the following activities [Jonkers00]:

- Develop global business model;
- Develop detailed business model;
- Develop context model;
- Develop global system model;
- Develop detailed system model;
- Develop deployment model.

It defines deliverables for each of these activities. The order in which the various activities are performed is not prescribed. Instead, there are clearly defined dependencies between the deliverables. The context model, consisting of use case models and activity diagrams, provides the link between business modelling and system modelling. CADA provides guidelines and techniques for identifying components during the design, but does not provide precise guidance on how to include existing components into a new design.

3.4.1.2 - Data oriented methods

The design of distributed information system involves making decisions on the placement of data and applications across the different sites of the distributed information system. It has been suggested that the organisation of distributed systems can be investigated along three orthogonal dimensions [Levin75]:

1- Level of sharing
2- Behaviour of access patterns
3- Level of knowledge on access pattern behaviour

In terms of level of sharing, there are three possibilities. First, there is no sharing: each application and its data execute at one site, and there is no communication with any other program or access to any data file at other sites. We then find the level of data sharing; all the programs are replicated at all the sites, but data files are not.

Accordingly, user requests are handled at the site where they originate and the necessary data files are moved around the sites. Finally in data-plus-program sharing, both data and programs may be shared, that is to say that a program at a given site can request a service from another program at a second site, which in turn, may have to access data file located at a third site.
The distinction between data sharing and data-plus-program sharing illustrate the differences between homogenous and heterogeneous distributed systems. Along the second dimension of access pattern behavior, it is possible to identify two alternatives. The access patterns of user requests may be static, so that they do not change over time, or dynamic. It is obviously considerably easier to plan for and manage the static environments than would be the case for dynamic distributed systems [Özsu97].

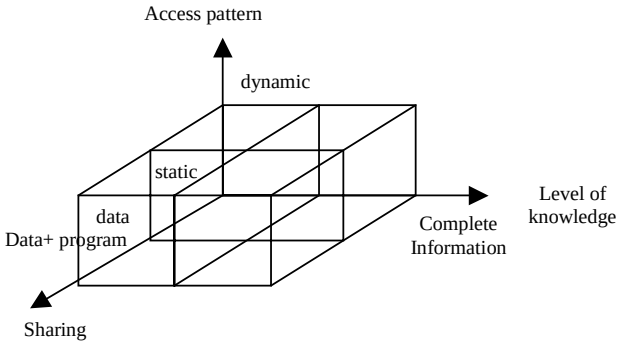


Figure 5. Framework of distribution [Özsu97]

The third dimension of classification is the level of knowledge about the access pattern behaviour. One possibility is that the designers do not have any information about how users will access the database. In this case, it is very difficult to design a distributed information system that can effectively cope with the situation. The more practical alternatives are that the designers have complete information, where the access patterns can reasonably be predicted and do not deviate significantly from these predictions, as well as partial information, where there are deviations from the predictions.

3.4.1.2.1- Alternative design strategies

Two major strategies that have been identified [Ceri87] for designing distributed databases are the top down approach and the bottom up approach.

3.4.1.2.1.1 - Top down design process

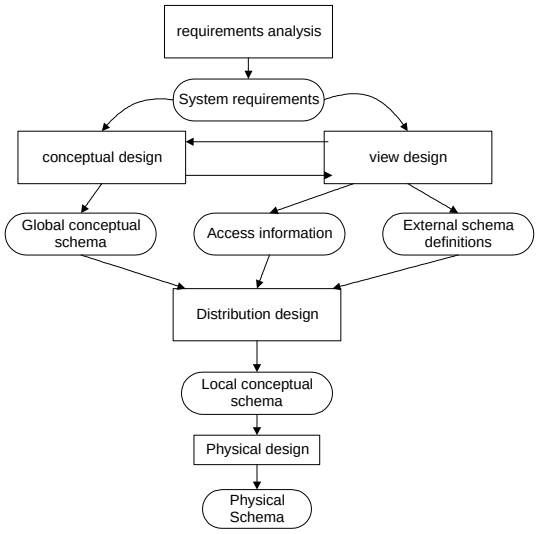


Figure 6. Top-down design process [Özsu99]

The design activity begins with requirements analysis that defines the environment of the system and the potential data and processes needed by different system users. The requirement document is an input to two parallel activities: view design and conceptual design. The view design activity deals with defining the interfaces for end-users. The conceptual design, on the other hand, is the process by which the enterprise is examined to determine entity types and relationships among these entities. We can divide this process into two related activity groups [Davenport81]: entity analysis and functional analysis. Entity analysis is concerned with determining the entities, their attributes, and the relationships among them. Functional analysis is concerned with determining the fundamental functions in which the modelled enterprise is involved. The result of these two steps needs to be cross-referenced to get a better understanding of which functions deal with which entities [Özsu99](see Figure.6).

In conceptual design and view design activities the user needs to specify the data entities and must determine the application that will run on the database. Then, the global conceptual schema and access pattern information collected as a result of view design are inputs to the distribution design step. The objectives of this stage are to design the local conceptual schemas by distributing the entities over the sites of the distributed system. At this level, entities are regrouped into fragments, then, these fragments are distributed. Thus the distribution design activity includes two steps: fragmentation and allocation. The last step in the design process is the physical design, which maps the local conceptual schemas to the physical storage devices available at the corresponding sites [Özsu99].

3.4.1.2.1.2- Bottom up design process

The bottom up approach is suitable for integrating exiting local conceptual schemas into a Global conceptual schema. The bottom-up approach is suitable for environments where a number of databases already exist, and the design task involves integrating them into one database. The beginning of the bottom-up approach is the individual local conceptual schemas (see Figure.7).

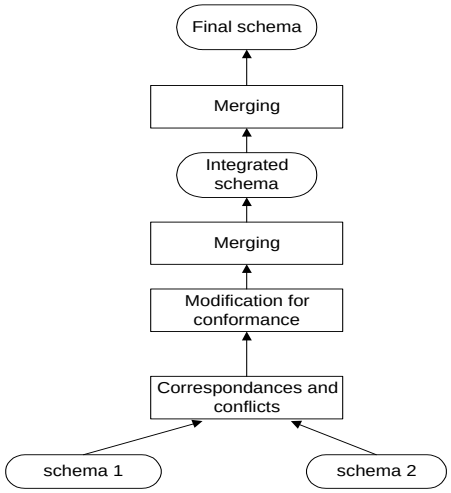


Figure 7: The bottom-up design process

3.4.1.2.2 - Fragmentation

Relation instances are essentially tables, so the issue is to find alternative ways of dividing a table into smaller ones. *A fragmentation schema is a division of the global schema of relations into smaller relations.* In fact, a relation is not a suitable unit for distribution. First, application views are usually subsets of relations. Therefore, the locality of accesses of applications is defined not on entire relations but on their subsets [Özsu99].

A '*horizontal*' fragment is a partial relation containing all the attributes of the original relation, but with fewer tuples, all of which match on a specific attribute. A '*vertical*' fragment is a partial relation containing a subset of attributes of a relation and a key to the original relation upon which vertical fragments can be re-joined to form the original relation. A vertical fragmentation may be followed by a horizontal one, or vice versa if the simple vertical or horizontal fragmentation isn't sufficient to satisfy requirements of user applications.

3.4.1.2.2.1- Horizontal fragmentation

Various horizontal fragmentation approaches were presented in the past. [Özsu99] Computes horizontal fragmentation of a class using predicates extracted from user

queries. Two versions of horizontal fragmentation are considered: primary and derived.

3.4.1.2.2.1.1 - Primary horizontal fragmentation

A primary horizontal fragmentation is defined by a selection operation on the owner relations of a database schema. A horizontal fragment R_i of a relation R consists of all the tuples of R that satisfy a minterm predicate m_i. A minterm predicate is a conjunction of simple predicates. A simple predicate p_j defined on a relation $R(A_1, A_2, ..., A_n)$, where A_i is an attribute defined over domain D_i is defined by:

$$p_j : A_i \ \theta \ Value$$

where $\theta \in \{=, <, >, \neq, \leq, \geq\}$ and Value is chosen from the domain of A_i ($Value \in D_i$). Given a set $Pr_i = \{p_{i1}, p_{i2}, ..., p_{in}\}$ of simple predicates for relation R_i, the set of minterm predicates $M_i = \{m_{i1}, m_{i2}, ..., m_{iz}\}$ is defined as

$$M_i = \left\{ m_{ij} \middle| m_{ij} = \bigwedge_{P_{ik} \in Pr_i} p^*_{ik} \right\},\ 1 \leq k \leq m,\ 1 \leq j \leq z$$

where $p^*_{ik} = p_{ik}$ or $p^*_{ik} = \neg p_{ik}$.

Hence, given a relation R, its horizontal fragments are given by:

$$R_i = \sigma_{F_i}(R),\ 1 \leq i \leq w$$

where F_i is the selection formula used to obtain fragment R_i. The selection formula is selected in adequacy with the project requirement. For example, the selection formula assigns to an attribute different ranges of value and according to these ranges the schema is fragmented.

The following algorithm generates a complete and minimal set of predicates *Pr'* given a set of simple predicates *Pr*. It begins with finding a predicate that is relevant and that partitions the input relation (see Figure.8). The iteration iteratively adds predicates to this set, ensuring minimality at each step. Therefore, at the end the set *Pr'* is both minimal and complete. The second step in the primary horizontal design process is to derive the set of minterm predicates that can be defined on the predicates in set Pr'. These minterm predicates determine the fragments that are used as candidates in the allocation step. The third step of the design process is the elimination of some of the minterm fragments that may be meaningless. This elimination is performed by identifying those minterms that might be contradictory to a set of implications *I* [Özsu99] (see Figure.8).

Algorithm COM_Min

Input: R: relation; Pr : set of simple predicates
Output: Pr':set of simple predicates
F: set of minterm fragments
Begin
find a $p_i \in P_r$ *such that* p_i *partitions R according to Rule 1*
$Pr' \leftarrow p_i$
$Pr \leftarrow Pr - p_i$
$F \leftarrow f_i \ \{ f_i$ *is the minterm fragment according to* $p_i \}$
Do
find a $p_j \in Pr$ *such that* p_j *partitions some* f_k *of* Pr' *according to Rule 1*
$Pr' \leftarrow Pr' \cup p_j$
$Pr \leftarrow Pr - p_j$
$F \leftarrow F - f_j$
If $\exists p_k \in Pr'$ *which is nonrelevant then*
$Pr' \leftarrow Pr' - p_k$
$F \leftarrow F - f_k$
end if
until Pr' *is complete*
End

Figure8. Algorithm of primary horizontal fragmentation [Özsu99]

3.4.1.2.2.1.2 - Derived Horizontal fragmentation

A derived horizontal fragmentation is defined on a member relation of a link according to a selective operation specified on its owner. Two points are important and have to be mentioned. First, the link between the owner and the member relations is defined as an equi-join. Second, an equi-join can be implemented by means of semi-join.
Accordingly, given a link L where *owner(L) = S* and *member (L)= R*, the derived horizontal fragments of *R* are defined as:

$$R_i = R \quad S_i, \ 1 \le i \le w$$

where w is the maximum number of fragments that will be defined on *R*, and $S_i = \sigma_{F_i}(S)$, where F_i is the formula according to the primary horizontal fragment S_i is defined.

3.4.1.2.2.2- Vertical fragmentation

The objective of vertical fragmentation is the partition of a relation into a set of smaller relations so that many of the applications will run on only one fragment. In this

situation, an optimal fragmentation is one that produces a fragmentation scheme, which minimizes the execution time of applications that run on these fragments. Relation *R* produces fragments $R_1, R_2, ..., R_n$ each of which contains a subset of attributes of *R* as well as the primary key of *R* [Navathe84]. The aim of vertical fragmentation is to put together those attributes that are accessed together. Two types of heuristic approach exist for the vertical fragmentation of global relations:

- Grouping: starts by assigning each attribute to one fragment, and with each stage, joins some of the fragments until some criteria is satisfied. Grouping was first suggested in [Hammer79] for centralised databases, and was used later in [Sacca85] for distributed databases.
- Splitting: starts with a relation and decides on beneficial partitioning based on the access behaviour of applications to the attributes. The technique was first discussed for centralised database design in [Hoffer75]. It was then extended to the distributed database environment in [Navathe84].

The major information required for vertical fragmentation is related to applications. The major data requirement related to applications is their access frequencies. Let $Q = \{q_1, q_2, ..., q_n\}$ be the set of user queries (applications) that will run on relation $R(A_1, A_2, ..., A_m)$. Then, for each query q_i and each attribute A_j, we associate an attribute usage value, denoted as $use(q_i, A_i)$, and defined as follows [Özsu99]:

$$use(q_i, A_j) = \begin{cases} 1 \text{ if attribute } A_j \text{ is referenced by query } q_i \\ 0 \text{ otherwise} \end{cases}$$

3.4.1.2.3- Allocation

Fragment allocation is a major issue in distributed database design since it concerns the overall performance of distributed database systems. Assume that there is a set of fragments $F = \{F_1, F_2, ..., F_n\}$ and a set of sites consisting of $S = \{S_1, S_2, ..., S_m\}$ on which a set of applications $Q = \{q_1, q_2, ..., q_p\}$ is running. The allocation problem involves finding the optimal distribution of F to S [Özsu99].

One of the important issues that need to be discussed is the definition of optimality. The optimality can be defined with respect to two measures as defined by [Dowdy82]:

- Minimal cost: the cost functions consist of the cost of storing each F_i at a site S_j, the cost of querying F_i at a site S_j, the cost of updating F_i at all sites where it is stored, and the cost of data communication. The allocation problem then attempts to find an allocation scheme that minimizes a combined cost function.
- Performance: the allocation strategy is designed to maintain a performance metric. Two well-known ones are to minimize the response time and to maximize the system throughput at each site.

A simple formulation of the problem is : let F and S be defined as before. For the time being, we consider only a single fragment F_k. We make a number of assumptions and definitions that will enable us to model the allocation problem [Özsu99].

1- Assume that Q can be modified that it is possible to identify the update and the retrieval only queries, and to define the following for a single fragment F_k: $T = \{t_1, t_2, ..., t_n\}$

where t_i is the read-only traffic generated at site S_i for F_k and

$$U = \{u_1, u_2, ..., u_m\}$$

where u_i is the update traffic generated at site S_i *for* F_k.

2- Assume that the communication cost between any two pairs of sites S_i and S_j is fixed for a unit of transmission. Furthermore, assume that it is different for updates and retrievals in order that the following can be defined:

$$C(T) = \{c_{12}, c_{13}, ..., c_{1m}, c_{m-1,m}\}$$
$$C(U) = \{c'_{12}, c'_{13}, ..., c'_{1m}, c'_{m-1,m}\}$$

where c_{ij} is the unit communication cost for retrieval requests between sites S_i *and* S_j and c'_{ij} is the unit communication cost for update requests between sites S_i *and* S_j.

3- Let the cost of storing the fragment at site S_i be d_i. Thus we can define $D = \{d_1, d_2, ..., d_m\}$ for the storage cost of fragment F_K at all the sites.

Then the allocation problem can be specified as a cost-minimization problem where we are trying to find the set $I \subseteq S$ that specifies where the copies of the fragment will be stored. In the following, x_j denotes the decision variable for the placement such that

$$x_j = \begin{cases} 1 \text{ if the fragment } F_k \text{ is assigned to site } S_j \\ 0 \text{ otherwise} \end{cases}$$

The precise specification is as follows :

$$min\left[\sum_{i=1}^{m}\left(\sum_{j|S_j \in I} x_j u_j c'_{ij} + t_j \min_{j|S_j \in I} c_{ij}\right) + \sum_{j|S_j \in I} x_j d_j\right]$$

and $x_{j=0}$ *or* 1.

The second term of objective function calculates the total cost of storing all duplicate copies of the fragment. The first term, corresponds to the cost of transmitting the updates to all sites that hold the replicas of the fragment, and to the cost of executing the retrieval only requests at the site, which will result in minimal data transmission cost [Özsu99].

3.5- Summary and conclusion

We have presented the two main existing approaches for the design of distributed information system. The first approach is a component oriented approach. As explained, the different presented methods are based on semi-formal design methodologies, and often do not define a real distribution design step but offer to designers different concepts as components or packages to be used at the design step. The three presented methods are all inspired from existing design methodologies. They presented high complexities of use and practise. In fact, they have different notations which make harder their training. Moreover, they are not equipped with any sub-system identification process. This step is still completely depending on the designer experiences and skill. This implicates that there is no optimal obtained solution but a huge set of solutions with a distribution cost which can be varied from the optimal to the worst. Finally, there is no consideration of integrity constraints in any of the presented methods, which can generate a high risk of system inconsistency (see table 2).

	Catalysis	**UML**	**CADA**
Notation	UML+ Extensions	UML	UML+ Extensions
Complexity	Very complex	Complex	Very complex
Sub-system identification	No	No	No
Decomposition level	No	No	No
Formalisation level	OCL	No	No
Sub-system site relation	No	No	No

Table 2: Distributed information system design methodologies comparison

The second approach presented is the data oriented method. It is entirely based on different allocation and fragmentation process formalised by a set of different algorithms. This approach is better formalised and more precise than the first one. Nevertheless, it presents a very high level of complexity but no flexibility. Indeed, there is no defined and clear process of algorithms use that implicates a hazardous use. The designer has no ability to indicate the different project constraints such as the location of some desired tables on predefined sites. This approach does not include the system integrity constraints. In fact, the distribution is chiefly based on data and does not take into account any functional system aspect.

These two approaches are based on different system aspects, which are the functional aspect concretised by component concept, and the data aspect. The dissociation of

these aspects is harmful to the design continuity and does not reflect the real operating mode of the system. Indeed, each one of these systems' facets is essential to its correct running and must be regarded as the base of any distribution approach.

Chapter IV
Distributed Information system Knowledge Pattern

The Distributed information System is a collection of distributed data and process over multiple sites that are connected with some kind of distributed system architecture commonly known as *middleware* [Shmidt00]. It exposes a common set of services across platforms and provides a homogenous computing environment in which distributed information applications can be built. Today several middleware are available. The most used ones are *CORBA* (Common Object Request Broker Architecture) from OMG and *EJB* (Enterprise Java Beans) from Sun [Ekberg99].

The existing gap between current information system design methodologies and the distributed system design models, forced developers to operate modifications on the design models to adapt them to the technology specification. Every middleware specification has its own implementation constraints and generally, designers do not have enough knowledge about the implementation process to realize a well-designed system that corresponds to the implementation model without major modifications [Hirschfeld96].

In fact, methodologies for the design of information systems have usually paid modest consideration to distribution and communication characteristic of systems [Deweger99]. However, this situation has improved: information systems have noticeably grown in dimension and range, organizations want many of their separate systems to be integrated and consequently the number and the variety of geographically distributed users of these systems have become wide [Ferreira91]. The distributed systems community has produced methodologies for the design of distributed systems [Casal02]. However, these methodologies pay little attention to the

information aspects of distributed information system; instead their strength is in the distribution and communication aspects of these systems [Andrade96]. The information aspect still not well specified in these methodologies despite its importance for the distributed information system survival [Sol92].

For this purpose, we define a set of interaction pattern between the various participants in the life cycle of a distributed information system. These patterns initially treat the exploited knowledge by each one of these participants. Then, these patterns will be the intermediary of communication to establish a form of formal communication between them to allow a better comprehension of the system and thus to ensure a better continuity between the various levels [Rivera-Vega90]. These patterns are divided into three categories corresponding to three knowledge interaction levels [Hui00]. The first is the designers' knowledge needed by developers to capture the design environment and to well understand the purposed conceptual diagram. For example, in some special case designers need to fix some critical data on some specific sites to ensure its availabilities in case of network shutdown. Developers usually adapt the purposed conceptual diagram to the technical environment to increase the distributed information system performance and can, if not noticed, move these data from the specified site to another one without informing designers [Varadarajan89]. The second category is the developers' knowledge pattern that regroups the different implementation parameters and the technical environment specificities. This pattern is useful for designers and assists them in adapting their purposed conceptual diagram to the technical environment. For example, some application servers used to develop distributed information system (Tomcat, J2EE, …) have their own security framework [Rahul01]. Any security approach used to design the distributed information system must fit into this framework. Finally we purpose a knowledge overlap pattern that encapsulates the common knowledge between the designers and the developers. An example of this knowledge is to exception treatment. The technical behaviour of each used technical environment is different from others and its integration in the conceptual diagram must be discussed and approved by both developers and designers.

This chapter would be further structured as follows: the first section introduces the concept of distributed information system and its main characteristics. The second section presents the design knowledge pattern. The section three presents the general forces of these patterns. The last section presents the pattern that will be developed in our study and introduces some use cases of our patterns.

4.1- Distributed information System design requirements

Distributed information systems are strongly inspired from the newest system architecture especially the distributed system, distributed databases and middleware infrastructure. But methodologies for the design of information systems have usually paid modest consideration to the distribution and the communication characteristic of information systems [Deweger99]. The distributed systems community has produced methodologies for the design of distributed systems. However, these methodologies pay little attention to the information aspects of a distributed information system;

instead, their strength is in the distribution and communication aspects of these systems. To develop distributed information system, we have to adapt a global conceptual model to the offered distributed infrastructure. Generally, the decisions of deployment and distribution are taken in the development phase. This obliges developers to take decisions completely different from their competence field [Espindola04].

The gap between these complementary environments is harmful to the continuity and to the robustness of the built distributed information system [Waters99]. To establish a strong communication mode between the two major actors working on this kind of system, we have to afford for them a common language and a unified mode of interaction. Knowledge pattern extracted from each environment represents the best option to bring them closer.

We define the most important goals that the design methodology adopted for the design of the distributed information system must consider. These goals are: transition to distributed information system, conformity of implementation and evolution and conformity. Next sections present these different goals.

4.1.1- Transition to Distributed information System

Distributed information system design methodologies and distributed system design have different process models. The implementation of a distributed information system upon a distributed architecture required the definition of clear steps and a unified process. The transition from the unified information system design schema to the distributed ones must mach with the main goals of this system. The aim of the distribution of the system is to share data between the suitable locations to reduce data cost, and to improve the system performance and Quality of Service [Booth81]. However, developers need to have clear indications about the data treatment inside the organization to avoid redundancy and data inconsistency [Snene03b].

The first step of this transition is performed at the end of the design phase. The usual result of the design step is a centralized information system schema which is not yet splitted in different subsystems. Designer must proceed to data fragmentation by exploiting the different data resulting from the analysis step [Ceri83a]. In fact, according to the different needs expressed by the end users they can fix the number of the different needed sites and decide which data must be located on the corresponding site. The matching is done by evaluating the data nature, its importance for the site and level of imposed security and integrity imposed to these data. Designers have also to provide enough explanations to developers to help them well understand the different choices that have been done at this step. Finally the different data that is still not fixed on any site must be clearly indicated to developers.

The major problems encountered by designers at this step are to adapt the unified design schema to the used distributed system technology. Without correct and complete information about the targeted technology, designers can produce a schema

that is not realisable because of the non respect of the different imposed constraints. The most common example of this case is the security framework needed by the different distributed system architectures. In fact, the design of data related to users, sessions, data protection must be done while conforming the strict design given by the system framework otherwise the security part cannot be implemented or run on the system. These information must be then given at the beginning of the design step to designers by developers to prevent them from making such incoherency.

4.1.2- Conformity of implementation

The architecture choices and the implementation models implicate to operate modification on the conceptual model obtained after the design process. Performance problems in distributed systems oblige developers to adapt the conceptual model to their needs. This may causes some differences between the initial model and the implemented one. More than this, security level definition in distributed environment has to be resolved sometimes by adding new components in the design model [Boogaards90].

The design model must be used only for its power to define the information level. The final implementation architecture must conform to its results. To ensure the design model from semantic modification, new control rules have to be added. These rules must be based on the different transition levels and they must be included in different parts of the common transition process.

The major modifications carried on the design schema result from the need of increasing the system performance, security and Quality of Service [Josifovski99]. In fact, it is still impossible to produce a design schema which presents the best indicators. An example of these modifications is the integrity constraints adaptation. Under their basic form delivered in the design schema, integrity constraints are always expressed under a database constraint rule. The implementation of such integrity can be done in three different forms in a distributed environment. The first form is the trigger form which does not imply to carry out any modification. This form is the simplest way to keep the design schema conform to the developed schema. The second form is done by user choices restriction. This form is used in the case where users have to select data from the database to carry out their queries. The presented list of data choices will exclude any data that can transgress the database integrity. The modification done on the design schema is more focused on the functional aspect than the data aspect. Such modifications can be dangerous to the performance of the information system if the restrictions prevent users from running some needed queries. The last form is to implement the integrity constraint as a command executed on the server side. In this case, the check control is done at real time on data stored in the server. The major modification carried out to the design schema is of the integrity constraint runtime location.

4.1.3- Evolution and conformity

Any information system needs to be adapted to the organizational needs and to the technological evolution. Evolution of distributed information system implies modifications and adaptations of implemented methods, classes, relations and integrity rules. This point means that the system must support evolution and maintenance activities. The evolution management must include integrity rules validation and transaction validation. They constitute the system dashboard [Leonard92].

The adaptation of the system must be done with respect to two conditions. The first is to respect the main aim of the system by keeping functional all the primary user needs. Updating the system without taking care about them can result in a system discontinuity which is harmful for its usability. An example of such update is adding new integrity constraints or simply their modification. If any of the new or modified constraint is contradictory with an oldest one, this will result in an incoherence of the database and so in stopping all the system functionalities related to these data

The second condition is to readapt the information system distribution while taking into account the new added data or functionalities. In fact, system performance and quality of service can decrease if the distribution logic that has been used to carry out the data and functions fragmentation is not respected. For example adding new integrity constraint can dramatically decrease the answering time. If this integrity constraint requires a set of data that is located on a site different from the runtime site of the constraint, or if the data set is dispersed over the different sites, running any transaction whose result needs to be verified and validated by the integrity constraint. System time needs to complete such transaction compared to needed time for any optimized distributed transaction is considerable.

4.2- Distributed Information System Knowledge Pattern

As explained above, three knowledge categories are distinguishable in a distributed information system project. These zones are called *Knowledge zone*. In fact, every zone is characterized by its own knowledge that is needed by other partner to successfully achieve the project. To put this knowledge under a formalized form comprehensible by different project partners who do not share the same technical and knowledge language, we define it as *knowledge pattern* form. These patterns will contain and express the different needed knowledge contributing to ensure a normal continuity for the project. Three different pattern categories are identified (see Figure 1).

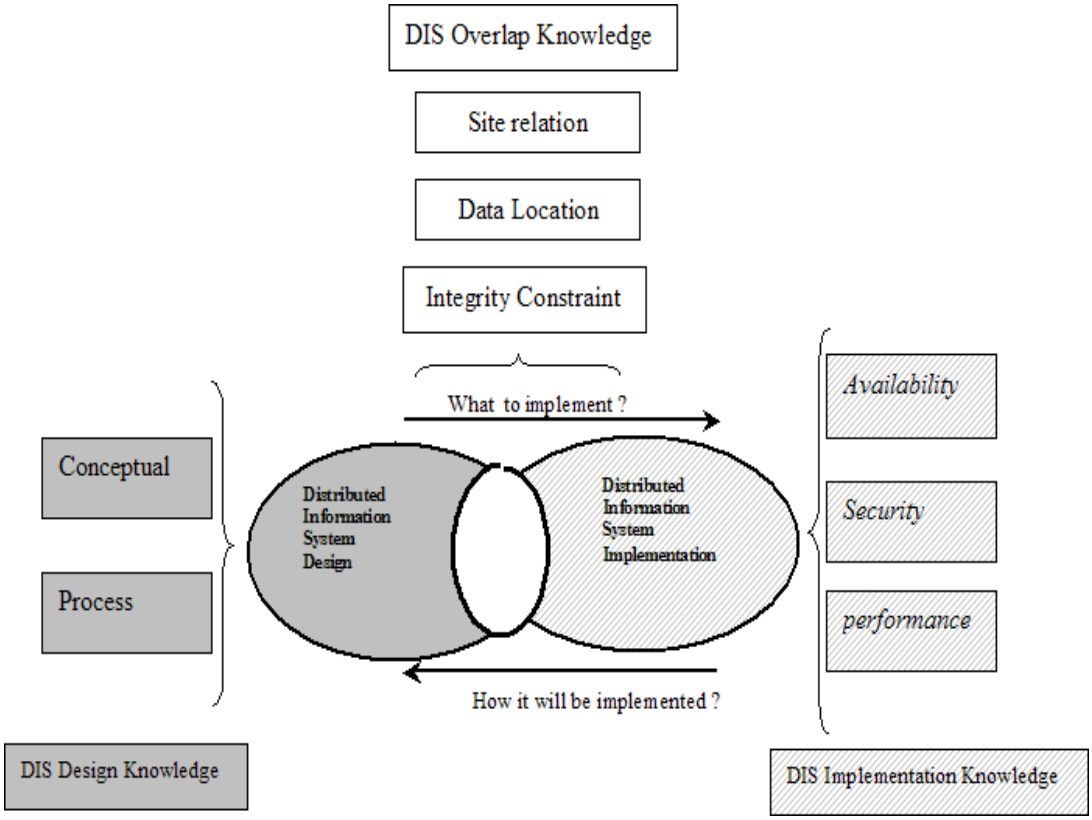


Figure 1: Distributed information system knowledge pattern [Snene03b]

The first pattern category is the *distributed information system design knowledge pattern*. This pattern conceptualise the diverse knowledge used to design the system and to produce it under the form of schema. As this schema is the result of a transformation of different information collected from end users, this transformation, as any other transformation, causes some data loss. Two domains are concerned by this data loss: *the conceptual domain* and the *process domain*. The first domain represents the different data that make up the system. They are represented under the form of a design schema. But, some of the collected data cannot be represented under this form due to their particularity. Nevertheless, these data can be important for developers for a better understanding of the given schema. For example, on a database table, designers can specify a user-address attribute under many fields, which are street-number, street, and town. This specification can result from a user need, such statistical use or for a further evolution. At the development step, as these needs are not clearly expressed in the design schema, developer for better performance and to decrease the answering time can unify the different fields in a unique one that represents the same data. In this case, the obtained information system will provide users with the same data then the wanted information system. But at the time when the end users will try to get the statistical information which concern only a part of the global address such as the street, this information will be unavailable and the information system will be unable to satisfy the users' requirements. The second

domain is the process domain that represents the different information system functionalities. These functionalities are represented under the form of functions and transactions. At the design step, designers usually do not specify the runtime manner and site needed for each transaction. However, this information can be vital for some critical functions. The term critical does not indicate their system aspect but the organisational one. In fact, if we suppose that the statistical functionalities mentioned in the example above will be used for each new data entry, and if we take the case of an international interim worker company, the location of this functionality runtime is crucial to determine the system answering time and its quality of services.

The second pattern category is the *distributed information system development knowledge pattern*. This pattern regroups structural and technical information related to the development languages and the runtime platforms. Such information is generally extracted from the different specifications of the used technologies. This information is expressed through the developed system but still hard to get at the design step. In fact, designers usually ignore the technical constraint that can be transgressed by their produced design schema. This obliges the developers to adapt the design schema to their used technologies and by the way to carry out some modifications that can be harmful to the system objectives. The different knowledge that must be considered by these patterns concern: security, performance and availability.

The security knowledge is of two kinds. The first is the *data security framework* defined and requested by the technical platform. Indeed, according to the used platform different protocols exist and must be respected at the design step. This special requirement must be expressed at the beginning of the design step by a clear and formal specification. Different forms can be used to express this specification, but the most comprehensible one is to provide the designer with a conceptual schema that must be integrated in the system schema. For example, the Tomcat [Rahul01] server obliges the designer to use a unique security framework defined by the Tomcat specification. The security data which are: User, Login, Password and Session must be defined as fields of a unique table called security. Otherwise, even the data are protected by other control checks and constraints; the transition from the login step to the session step cannot be done. In fact, if the requested data are expressed under another form than the one which the platform specifies, they cannot be treated.

The second security knowledge form is the *system security Knowledge*. In fact, any distributed information system needs to be protected from external or internal attacks. For this purpose, developers use different technologies and define different lockers on the system. This set of technical information must be transmitted to designers by indicating the security constraints that will be applied to the design schema at the development step. For example, using distributed databases must be accompanied by a data flow restrictions. This means that the critical data will be used just on a defined site and cannot be called by non-authorized sites. If designers distribute the different process without taking in account this fact, the security can be transgressed and the critical data are doomed to be in danger.

The third pattern category is the *distributed information system knowledge overlap pattern*. By overlap we designate the overlapping zone between the design and the development step. This knowledge results from the mixture of different kinds of information kinds related to both domains. It is composed mainly of three different components which are: *data location, site relation* and *integrity constraint*.

The *data location* concerns the distribution manner of the data. Indeed, a part of the data cannot be distributed in a classical way (fragmentation and allocation [ÖZSU99]). Some data is needed on specific site because of its importance to the good running of this site, or due to the end-user request. This information results both, from designers and developers. It implies that the design and implementation constraints have to be resolved jointly. The first pattern component's aim is to regroup these different locations' information in a formal way to facilitate the resolution of their related different restrictions.

The *site relation* component is the set of different information concerning the different existing relations between the sites. These relations can be of different kinds. The first relation category is the *organisational* relation. In fact, some needs require special connection between the different organisation sites. For example, some distributed information systems with critical sites are completely replicated due to their information importance. In such case, each site must inform the others of its correct management. It has also to maintain its backup site correctly and continuously. The second relation category is *technical*. Indeed, some technical constraints need some specific relations between the different existing sites. For example, using the J2EE platform requires the establishment of a special permanent connection between the different application containers in order to guarantee their coherences [Pawlan00].

The *integrity constraint* component represents the set of the different sorts of information needed to design and implement these constraints correctly between the different sites of the distributed information system. In this thesis, we have a special focus on this component due to its major importance and impact on the system good running. The different data existing around the integrity constraints are necessary for a good implementation of these constraints. In fact, as these different constraints regroup and result from different system environments such as the organisational, the security and the technical, it is necessary to classify these data in order to obtain the best possible level of system integrity.

4.3- General forces

The three kinds of knowledge pattern extracted from a distributed information system project are extracted from the different steps. Each pattern represents the main knowledge of the participant which is useful for others to ensure the project conformity and continuity. This knowledge represents the generic characteristics of their domain and the recurrent information concluded from the encountered experiences. In fact, depending on the nature of the project, the used infrastructure and the end-user requests, we can obtain various information that can enhance the

knowledge pattern, especially if these conditions are frequent. In this section, we present the main goals of each pattern in detail.

4.3.1. Distributed Information System Design Knowledge Pattern

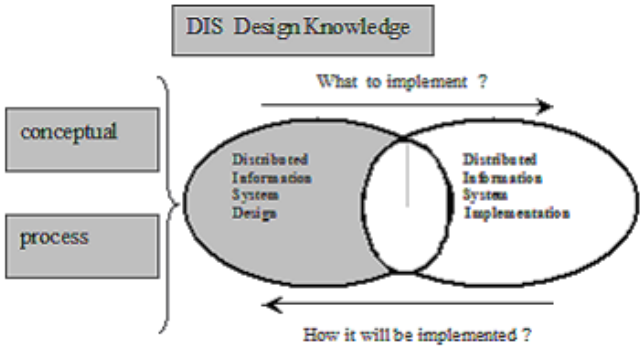
This pattern regroups the different kinds of information extracted from the design step. This information is mainly obtained from the different end-users queries which summarize generally the specific functionalities or needs and which are considered as constraints by the designers. Developers that are rarely in a direct touch with the end-users have to clearly understand these constraints in order to respect them at the development step. These patterns also summarize the following information (see figure2):

DIS Design Knowledge

Context : DIS design pattern is a knowledge pattern that is used while developing distributed information system. It concerns more specifically the transition from the design step to the development step.

Problem: establishment of the different knowledge used to achieve the design step of the distributed information system. As this knowledge can not be expressed throughout the schema, developers faced some difficulty to discover the aim of the schema.

Structure:



Consequences:

- The different processes are underlined and showed throughout the different schemas structures
- The designed organisational structure is shown thanks to a better process separation
- The different information components together with their preferred location are better explained.

Figure 2: Distributed information system design knowledge pattern [Snene03b]

- Static design: designers underline under this pattern form the different specificity of the obtained static design. They have to explain the users requests about some data location if it exists. In fact, if users request some data on some specific site, designers have to point it out in a clear way. They also present to the developers the general directories of the physical distribution. The logical distribution is done jointly between the designers and the developers and still strongly dependent on the logical distribution due to performance and cost constraints. Helping designers choose the best solution for decoupling the business logic will have a strong feed back on the physical distribution.

- Dynamic design: this information presents to developers the different possibilities to manage pertinent and impertinent data across the information system life cycle. Designers are solicited to clarify the kind of data used during the life cycle. In fact, for each kind of data a different development form can be deployed. This information will help developers choose the correct and the most suitable data form while respecting the design and development constraints.

- Organizational design: the functional aspect of the organization generally guides the organizational design. In fact, the organization is represented under two forms. The first is the data form which is represented under its two structure: the static and the dynamic design. The second is the transactional form which is represented under the set of system functionalities added to the different integrity constraints. This information will provide developers with the different transactional system aspect and help them realize a distribution according to the different business subsystem.

4.3.2. Distributed Information System Implementation knowledge Pattern

This pattern regroups the different data related to the development step. It identifies the different constraints resulting from the use of a specific platform. It also underlines the set of requirements imposed by the system performance and the quality of service. All this information is useful at the design step to ensure the production of a design schema that respects the technical constraints. In fact, designers have to be informed about this set of constraints in order to reflect them in the produced schema. Some specific information must be explained in this pattern. This information is:

- Resource identification: developers will have a clear design schema of the needed system. Designers have to produce it while taking into account the common system resources.
- Mode of communication: by defining a clear mode of definition, designers have to produce design schemas (sequential diagram, data flow,) according to the choice of the communication mode. Developers will not have to adapt or to modify them since designers respect technical constraints.

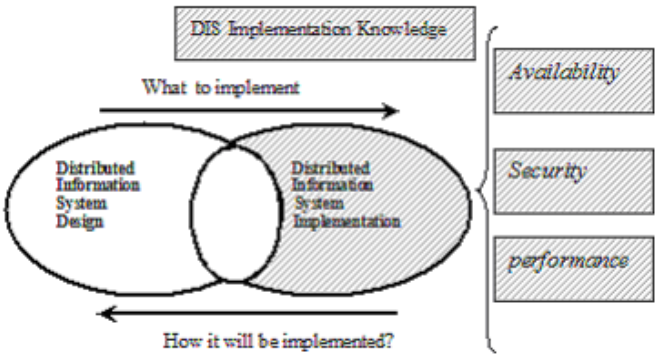
- Quality of service: by classifying the importance of different system functionalities, developers have enough knowledge about the system use manner and so how to distribute the different data and process locations.
- Architecture: giving to developers the global organizational design, will help them choose the right DSI architecture taking into account the geographical sites and the manner of their collaboration.

DIS Implementation Knowledge

Context : DIS implementation knowledge is a knowledge pattern that is used while designing distributed information system. It helps designers by giving them the main particularities of the target technological platform and development language, to adapt their design product to these particularities.

Problem: every technical platform has its own requirements and standards. Designing an information system that doesn't take into account these constraints oblige developers to change the design schema to adapt it to the used platform. That means making them involved in a realisation step that is completely different from their competences field.

Structure:



Consequences:

- Technical constraint and problem are examined before beginning the design step
- Less problem in the development level since the integration problems are examined at the beginning of the project
- The designed project is done under some technical constraints such as QoS, security and performance. They can provide a better result.

Figure3: Distributed information system implementation knowledge pattern [Snene03b]

4.3.3. Distributed Information System Overlap Knowledge Pattern

This pattern is the most critical one. In fact, it regroups two knowledge categories resulting from the two project domains. This overlap is very harmful for the continuity of the project if it is not well controlled and exploited. In fact, contradictory decisions can be taken in the case the communication between the different partners is not well established. In such case, this pattern will help the different project participants by building a formal way of communication that will precise the different decisions about the shared information (see figure4.).

The most important information that must be transmitted by this pattern is about:

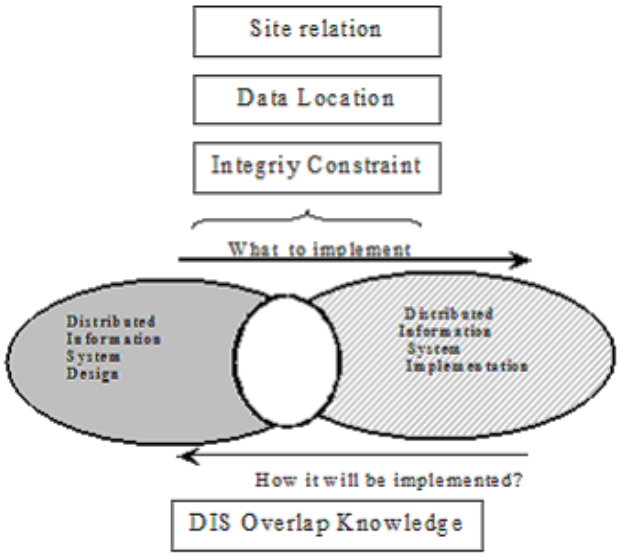
- Conformity of implementation: the verification and the validation of the distributed information system that resulted from the implementation step, is done by comparing the initial design and the proposed distributed information system. These patterns define verification methods understandable by both designers and developers.
- Data location: as explained above, the data distribution is decided in two different steps. The first is the design step, where designers or end-users can affect some data on some predefined sites. This step is called the logical distribution. The second step is at development step. Developers affect the set of data on the different sites according to the system performance and to the different constraints. To protect the distribution phase from being incoherent, every decision must be explained and argued according to the constraints required at decision domain.
- Evolution and conformity: the evolution of the distributed information system is essential for its survival. Generally this kind of system presents more attractive solution to replace, integrate or add new sub-systems. Patterns will be used as a knowledge repository that stored the manner in which the initial system is built. Using this kind of repository is helpful for the system maintenance.
- Integrity design: stored procedures, triggers and views generally represent integrity rules at the technical layer. These kinds of implementation reduce distributed system performance and interoperability. The overlap knowledge patterns present to designers the different implementation levels that can support the integrity rules implementation. The designers will decide for the mode of execution, validation and the priority order of these rules to avoid data inconsistency.

DIS Overlap Knowledge

Context : DIS overlap knowledge is a knowledge pattern that is used to find the shared and common needed knowledge between designers and developers in the aim to achieve in the best way a distributed information system. This pattern represents the common bridge between the two categories involved in such project.

Problem: Some DIS step require different kinds of knowledge categories. For example, the integrity constraint must be done while taking into consideration the network performance in order to minimize the data flow over the network. The two knowledge categories are expressed in different ways. Designer knowledge is usually informal and implicit while developer knowledge is formal and explicit

Structure:



Consequences:

- The different knowledge are expressed under a unique format and can easily be understandable by the different participants involved into the project.
- Communication between different participants is formalized and saved so it can be used later to understand the way in which the system is designed and implemented.

Figure 4: Distributed information system overlap knowledge pattern [Snene03b]

4.5. Conclusion

The set of Knowledge Patterns extracted from a distributed information system project are a group of proven reusable assets that can be used to increase the speed of developing and deploying distributed applications. These patterns have to help and to identify the interaction and processes of selecting and runtime topology. They will provide enterprise developers with a set of guidelines for building information

application, including performance, technology options, application design, development and security. These patterns will aim to reduce the existing gap between information designers and the developers by providing them with a unified interaction language. This language will enrich the design step by new concepts, which help developers manage the distribution step while respecting the project goals. They will also provide designers with different information summarizing the technical environment with its constraints. Such information is important due to the modification that has to be done on the design schema to be adapted it to the technical platform. Finally these patterns will define a formal way of communication between the different participants of the project. It will be useful specifically in the overlap domain case. In fact, the overlap knowledge pattern represents the most risky zone that can generate the system incoherence. In this thesis, we focus mainly on this pattern and more specifically on the integrity constraint formalisation in the distributed information system project. Indeed, integrity constraints represent enough characteristics to enable the different participants to define their common goals while respecting the major project goals. Integrity constraints are deduced from the system business logic. In fact, every constraint represents a set of one or more functional transactions or a set of security or integrity transactions. In all cases, these transactions influence the system performance due to the data flow generated by these transactions validation. In the same time, optimising these constraints implementation provide the system with a better integrity, coherence and security.

In next chapters, we present the needed enhancement of integrity constraint specification to adapt them to a distributed information system project. Then we give in detail the different algorithms that composed the overlap knowledge pattern. The goals of these algorithms are: optimising the system performance, guaranteeing the system integrity and security by defining a single way of data and function distribution and finally increasing the system availability by reducing the number of needed functional call between them.

Chapter V
Integrity Constraints specifications Enhancement for distributed information system design

This chapter firstly introduces the basic concepts integrity constraint in a relational database. Then it presents, the integrity constraints specifications enhancement necessary for their use in a basic context of distributed data. Indeed, the simple definition of the integrity constraints is not practical enough to be used in an optimal way which can ensure the total integrity of different sub systems.

This chapter is structured as follows: the first section presents the notion of integrity constraints in a classical relational database context. The second section introduces the proposed enhancement of integrity constraints specifications to adapt them to a distributed relational database context.

5.1- Database integrity concepts

This section introduces the basic *integrity concepts* for relational database systems. First, the general concept of integrity in database systems is introduced. Next, *integrity constraints* are discussed as a means to explicitly describe the integrity of a database. The definition of integrity constraints is then used to describe the properties that the execution of transactions should satisfy to maintain the integrity of a database.

5.1.1- The integrity concept

In a database system, the correctness of data accuracy is of great importance. The following disciplines in database technology try to keep certain classes out of error.

Security control deals with preventing users from accessing and modifying data in a database in unauthorized ways. The security control subsystem of a DBMS keeps record of the authorization of users to perform certain operations on certain data and checks this authorization upon database access [Date83].

Concurrency control deals with the prevention of inconsistencies caused by concurrent access of multiple users or applications to a database. The concurrency control subsystem orchestrates the access to the database in a way that the serializability property of transactions is guaranteed. In most cases, a locking or time-stamping technique is used [Korth86].

Reliability Control deals with the prevention of errors due to erroneous data that transgress the main logical coherence of the database. These errors can occur if the database is not well equipped with different business rules or integrity constraints that check the syntactic and the semantic values of the different entered data [Barbara92].

5.1.2 – Definition of integrity constraints

An integrity constraint is a non-variable of the field of application which concerns any of its states or any of its changes of states. Translated into database terms and applied only to data (excluding processes on the database), this concept of an integrity concept becomes a property that is possible to express from relations in the model, and which must be always verified by means of an algorithmic process at each state of the database or at each modification undergoes it [Leonard92].

During this difficult phase in the modeling of a field of application, the work of analyzing and formalizing these non-variables is of a remarkable efficiency and relevance for at least two reasons. The first of these lies in the fact that every non-variable relating to the data is such a significant property of the field of application that it must be able to be described in the model. Every final data model is complete with regard to the non-variables if it contains a description of all the non-variables relating to the data [Leonard92].

The second reason derives from the every position of the designers who analyze a field of application. Their main task is to observe and summarize their observations into a model that obeys, for example, the formalism of the relational data model. They are thus in a position to listen attentively to all information concerning the field of application [Bernstein80]. They are faced with the problem of having to remain rather passive; we could say that information is brought to them and they make a model from it.

Alternatively, they might adopt a more active position and gather information that they require to establish the data model. Designers, while adopting the active role immediately encounter a major problem; how to find the pertinent information and how to better understand the field of application. A data model must contain not only the relations, attributes and domains as we have described in the previous section, but also the integrity constraints. These have a very important role to play in the use of database, namely that of ensuring its coherence. A database is said to be coherent if all its relation's instances verify all the integrity constraints retained to ensure this coherence. Each integrity constraint must have its assigned range, that is, the set of update primitives that require its verification in order for the database to remain coherent [Leonard92].

Integrity constraints defined on a database schema *D* must be verified for each database state. Two major cases are underlined: the static integrity constraints, which concern the individually database state and dynamic integrity constraints, which concern a transition between two different database states. Integrity constraints can be defined as based on the following characteristics [Leonard92]:

- *The space scope* of a constraint is the part of the database on which the constraint is defined. Usually, a distinction is made between attribute constraints, tuple constraints, relation's constraints, and database constraints.
- *The time scope* of a constraint is the number of database states on which the constraint is defined. An important distinction can be made between static constraints, single step transition constraints, and general transition constraints.
- *The nature* with respect to the data model specifies whether the constraint is considered as an integral part of the data model or not. The first type of constraint is called an inherent or structural constraint; the second type is named an explicit constraint.

Different integrity constraints can be distinguished using the above properties. Mainly three main categories of integrity constraints are used.

5.1.2.1- structural constraint

According to [Codd79], the two minimal structural constraints are the entity rule and the referential integrity rule. The entity rule dictates that each attribute of the key is no null. This constraint is necessary to enforce the identification of entities. As these kinds of constraints can be involved in the distribution process, our research will focus uniquely on referential integrity.

Referential integrity is useful for capturing relationships between objects that the relational database cannot represent. Referential integrity involves two relations and imposes the constraint that a group of attributes in one relation is the key of another

relation. In the example 2 described below, there is a referential integrity constraint between domain and project. This rule prescribes that each project belongs to at least one existing domain.

5.1.2.2- Static constraints

These constraints are a set of rules that detect instantaneous databases, which cannot correspond to any probable state of the application's world in the past, present, or future, regardless to the database's update history. A static integrity law can be regarded as a Boolean function from the set of all the instantaneous databases that are well formed according to the schema or the database model. This function assigns the value false to those instantaneous databases, which cannot correspond to any probable state of the application's world.

In the example 2 described below, different static constraints must be defined such as the identification of employee, project and qualification.

5.1.2.3- Dynamic constraints

The dynamic constraint has the same definition as the static constraint, but the domain of the function is the set of transactions between instantaneous databases and false is assigned to highly improbable transitions between states of the application's world.

5.2- Integrity constraint specification enhancement for information system design

An *integrity constraint* is a condition defined in relation to either a class (intra-class integrity constraint) or to several classes (inter-class integrity constraint), and whose verification test can be carried out in an algorithmic manner on the objects of the classes concerned. An integrity constraint is defined by a Context, a Condition, a Range, a Response and an Expression. The *context* of an integrity constraint designates the classes on which the constraint is defined. The *condition* of an integrity constraint must be verified for every state of the database (*static integrity constraint*) or for every alteration of the state of the database (*behaviour integrity constraint*) by objects of the context classes. The *range* of an integrity constraint designates the set of primitives that alter the database tables and which must contain an algorithm for the verification of the constraint so that the coherent state of the database will be transformed by the primitive into another coherent state.

The *response* of an integrity constraint indicates the actions to be undertaken in the event of non-verification of the constraint. This involves specifying the action to be taken when, once a within range primitive has been triggered, the verification algorithm detects that complete action by the primitive would cause an incoherence relating to the integrity constraint. The normal response is to refuse to execute the primitive. However, the response may be more complicated and involves a cascade of deletions.

The *expression* of an integrity constraint may take the form of a mathematical expression or an algorithmic expression ending by a condition (for example functional dependencies, join dependencies, comparison of sets of entities, automata, predicative expression whose variables are classes or attributes…).

As an example we will use a database that models an engineering company. The entities to be modelled are the Domain, Project, qualification and the employees. Each project *proj*, requires specific qualifications *qua*. Each employee *emp* has at least one qualification *qua* and is assigned at least to one project *proj*. For each project *proj* we define a global domain that regroups similar projects. As shown in Figure 1, the example database is composed of four tables connected by four different relations. Some basic integrity constraints are expressed by using the cardinality proprieties. For example, the relation assigned to have a 1..n cardinality from employee to project. This cardinality indicates that every employee is assigned at least to one project.

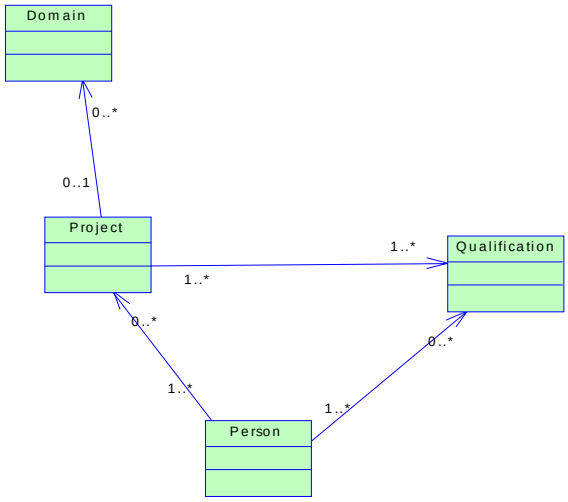


Figure 1: ER example model

IC $_{Expression}$ **:**

$\forall$ *pers* $\in$ *Person*, $\exists$ *proj* $\in$ *Project* $\wedge$ *qua* $\in$ *Qualification* / *pers.qua* $\in$ *pers.proj.qua*

Context *(Project, Person, Qualification)*

Different degrees of importance may be considered on the same integrity constraint according to their data context. Indeed, with the same integrity constraint and the same database model, an integrity constraint may be more or less important due to the quality of stored and imported data. To define the data context we have to specify the level of data propagation according to the integrity constraint.

In our example, the creation of a new person can invalidate the constraint since the new created entity *pers* of Person must be matched with an instance *proj* of project. The *proj.Qualification* must also match with the *pers.qualification*. The creation of a new project *proj* or a new qualification *qua* does not transgress the constraint. We suppose that an instance *pers1* of person is assigned to one project *proj1* of project and

has only one qualification *qua1*. The update of *proj1* can transgress the constraint in the case of the delete of *qua1* from the required qualification of *proj1*, *pers1* will have no more project to which he is affected. The update of *pers1* can also transgress the constraint in the case of the delete of *qua1* from *pers1*.

The delete of *proj1* from project represent the same risk as the updating of *proj1* if an instance *pers1* of person is matched only to *proj1*. The delete of a qualification *qua1* can transgress the constraint if an instance *pers1* of person is matched only which *qua1*.

A *range board* represents the set of primitives likely to transgress the integrity constraint when applied to a specific class called risk class. By *set of primitives* we designate the three basic database functionalities, i.e. new, update and delete. By *risk class* we indicate the class that by applying one of the primitives may violate the integrity constraint.

	New( )	*Update ( )*	*Delete ( )*
Project	**-**	**Yes**	**Yes**
Person	**Yes**	**Yes**	**-**
Qualification	**-**	**-**	**Yes**

Table1: The integrity constraint range board

5.2.1-Integrity constraint specifications enhancement for distributed information system design

To implement integrity constraints in a distributed information system context, we need to add the distribution aspect to the given definition of the classical integrity constraint. In fact, the different element of an integrity constraint context can belong to different sites of the distributed information system. In the case of the unavailability of one or more sites the integrity constraint can't be verified which can be the cause of the creation of an incoherent database state.

We have also to notice that the major data flow between different sites is materialized by the integrity constraint call and verification. In fact, every operated function except the single select in some special cases (data do not need special authorization check to be available to the end users) is based on different integrity constraints calls. For example, the simple update functions have to execute different integrity constraints before being performed. The first constraint is to check that the user has enough rights to run an update upon the context. Secondly, other constraints have to check that data is not used by other users. Finally, other constraints have to check if the data is concerned or not by some restriction constraints and if the new data will keep the data state coherent.

This violation may occur on the data of the risk class or on the data that belongs to any other class within the integrity constraint context. This specific path of data needs to be verified after the execution of a risky primitive as defined in the range board called the *cross-reference risk table*. The cross-reference risk table regroups the different risk classes defined in the range board. It emphasizes the relation between the root class and the classes that will be affected by the transgression of any corresponding primitive.

Risk class	***Primitive***	***Cross reference***
Project	New ( )	-
	Update ( )	Person, Qualification
	Delete ( )	-
Person	New ( )	Project,
	Update ( )	Project, Qualification
	Delete ( )	-
Qualification	New ( )	-
	Update ( )	Person, Project
	Delete ( )	Person, Project

Table2: Cross reference risk table

As a result of the cross reference risk table, we obtain the propagation schema of an integrity constraint across the different distributed information system sites. Knowing the different table locations, we get the data flow schema across the different distributed information system sites related to every primitive. The first goal of this integrity specification enhancement is to provide the designers and the developers with information on real system performance needed by each transaction carried out over the system and which make different integrity constraint call. The second goal is to help designers in the distribution step by giving them a classification of the most related entities. Reducing the distribution over these entities, the system performances are increased due to the reduction of the different primitives calls.

The next chapter presents the different optimisation algorithms based on the set of integrity constraints specification enhancements. As presented above, our algorithms are based on two kinds of constraints. The first algorithm is based on the different referential constraints presented under foreign key form. The second algorithm is a continuation of the first one. In fact, it exploits the generated set of distribution combinations from the running of the first one and optimise the set of the others integrity constraint calls.

Chapter VI
Distributed Information System design based on integrity constraints optimisation

The design of distributed information system is mainly executed by two different approaches. The first approach is based on the data distribution without taking considerations of the functional, transactional or the integrity aspects. This approach states that the data distribution optimisation enabling to obtain the best system performance. The major defect of this approach is its difficulty of use, its increasing complexity according for the whole data to be distributed and its non-consideration of different other system aspects such as integrity and respect of the technical constraints. The second approach is based on the transactional aspect. This approach is based of the design of the distributed information system while constituting different business or logic components that can be distributed over different sites. This approach is still easier to use than the first one but generates a set of different distribution solutions that cannot be compared according to their performance or correctness. These two approaches remain primarily approaches, which get involved in the level of the design without according importance to the development level. They present also major gaps of integrity and possibilities of giving the choice to the end-users to specify their constraints. The design of distributed information system based on integrity constraints presents new alternative compared to the existing methodologies. Indeed, considering the integrity constraints as the main meeting point of the different domains and participants involved in the project is based on the different layers of use of the integrity constraints. By optimising the validation of integrity constraints existing

along the different sites, we attempt to settle these different aspects and produce a clear distribution and essentially more powerful and coherent aspects.

The first algorithm is based on the foreign key distribution optimisation (*FKDO*). Indeed, the foreign keys are considered as the basic and more used form of integrity constraints. The algorithm goal is to reduce the number of site relations resulting from the foreign key connections between the different existing data tables. The second algorithm is based on the global integrity constraints optimisation (*ICDO*). The algorithm goal is to redistribute the first distribution combination obtained from the first algorithm while considering the whole set of integrity constraints. Finally, the third algorithm is based on the tuning of the obtained distribution combination in accordance with the weighting of the different integrity constraints. The weighting of each integrity constraints is calculated from their frequencies of use. In fact, some integrity constraints are less used than others according to their call by the different system transactions. This algorithm is called the global integrity dependencies optimisation algorithm (*GIDO*).

The chapter is further structured as follows: section 1 presents the FKDO algorithm; section 2 introduces the ICDO algorithm. The third section presents the GIDO algorithm and the different weighting criteria. The last section presents the performances of the different algorithms and their way of use to guarantee the optimal distribution set.

6.1- Foreign Key Dependencies Optimisation

A foreign key or the referential integrity involves two relations and imposes the constraint in a way that a group of attributes in one relation is the key of another relation. This means that between different data structures or tables, we have a basic relation based on the importation of the set of data from a table to figures out as a foreign key of a second table. If we consider that for every transaction calling the second table data, we have to establish a network communication between this table and the first from which the foreign key is imported, we easily deduce that the more the existing foreign keys are dispersed of the importation data tables, the more it will generate a data flow inside the distributed information system. This data flow reduces systematically the system performance and increases the risk of incoherence between the different data sites. In fact, the answering time of the system is partially depending on the amount of the data flow. The coherence of the different sites can be violated in the case of the validation of a transaction that modifies data, which is the foreign key of another table on another site distinct from the validation one. The incoherence can happen if the transaction is validated while the second site is down. By down, we mean that the site is disconnected or that connection is impossible to be established between them.

6.1.1- The FKDO constraint

The purpose of the FKDO algorithm is to help the designers choose the distribution combination with the minimum possible cost while guaranteeing the flexibility of choice by giving them the possibility to fix some tables on some dedicated sites. We suppose that a foreign key *FK1* exists as a reference in a Table *T1* on the site *S1* and references a table *T2* on the site *S2* will generate a transaction cost *C1*. This cost will be adjusted after the final distribution set generation. The cost is arbitrary equal to 1 if the two tables are located on two different sites. The adjustment of the weighting of the various rules is done firstly at the final generation step by determining the importance of each relation, and secondly and at the runtime by calculating the frequency of call of every relation. A site S_i represents a physical part of the distributed information system that contains a logical subsystem.

Starting with the number of the tables and of the locations, the FKDO algorithm generates the different primary distribution possibilities. The result will be a matrix with the different possible solutions and their related costs. Users and designers have the possibility to fix a set of tables on some sites. In this case, these tables will be assigned to their related sites and the matrix will contain the value of the matching of the other tables with the fixed one. We consider that every table depending on another table generates a transaction cost while the tables without any relation between them or which are located in the same site do not involve any cost in distribution. Basically we have to resolve the optimisation constraint that is represented by the following formula (see Figure.1):

$$\text{Min} \sum X_{IJ} \quad \text{I, J} \in \{Table_n\} \text{ and } \text{X} = \text{FK (I, J)}$$
$$/ \, X_{IJ} = 0 \text{ if } (\text{I, J}) \in \{ S_N \} \text{ and}$$
$$X_{IJ} = 1 \text{ if } \text{I} \in \{ S_N \} \text{ and } \text{J} \in \{ S_M \} \text{ and } N \neq M$$

Figure 1: the FKDO constraints [Snene03a]

where:

- $S = \{S_1, S_N\}$: represents the set of different existing sites. This set deduced from the number of physical existing sites belongs to the set of deployment sites. We consider that the geographical distribution has no consequence on the answering time of the site. Thus, we consider that this time is the same without any dependence of the physical site location.

- $T = \{T_1, T_j\}$: the set of tables. These tables are extracted from the design schema. As our approach does not take into account the partial or the full replication, this set still constant along the different distribution step. This set is extracted only from the logical design and not from the physical design, which implies the addition of multiple tables corresponding to the technical constraints of the used Distributed Database Management System.

- X = $\{X_{ij}\}$: the set of foreign keys dependencies between T_i and T_j. This set represents the number of primary relation between the different table and which are based on a foreign key importation. As explained above, this set is composed of a one-way primary relation. This statement implies that each relation will be counted only for the table that contains the imported keys.

- Matrix = double $[n^{j'}]$ $[j+1]$: the matrix containing the distribution possibilities where $n^{j'}$ represents the number of different design possibilities. n represents the number of the different existing sites. j' represents the number of existing tables. If designers match the localisation of y tables on preferred sites the number of the tables that will take into account is $(j-y)$. $j+1$ represents the number of the matrix column. This set is composed of the number of the existing tables added to the cost column. The value of the cost column is generated form each distribution solution according to the number of existing key relations.

The algorithm has the following structure:

```
indC=C-1;
indS=1;
indC1=0;
indL1=0;
 while (indC>=0){
        indL=0;
        bloc =  S ^ (C-indC-1);
        while (indL<=L-1{
                     indL2=indL;
                     while(indL<=indL2+bloc-1){
                           assign indS to m[indL][indC];
                           indL=indL+1; }
          if (indS>=S)
              indS=1;
          else indS= indS+1;}
 indC= indC-1; }
```

Figure 2.1: The matrix algorithm

```
while (indL1<=L){
     cost=0;
     ndC=0;
     while (indC1<C){
               indC2=indC1;
               check dependencies between tables at indC1 and indC1+1;
               while (indC2<C){
                 if (m[L][indC1] “is different then” m[L][indC2+1]
                     AND  dependencies “is different then” null then
                                  tmpV=1;
                                    else tmpV=0;
                                    C2=C2+1;    }
                 indC1=indC1+1; }
              cost=cost+tmpV;
   assign cost; }
   indL1=indL1+1;
```

Figure 2.2: The cost algorithm

At the first part of the algorithm (Fig.2.1), the code generates the different distribution possibilities and fills the matrix with their related combination. The second part (Fig.2.2) checks the dependencies between the tables and calculates the cost of the generated distribution cases.

6.1.2- FKDO Use Case

Our use-case (see Figure.3) is composed of the set of table {A, B, C, D, E} with the set of foreign keys $\{X_{BA}, X_{BC}, X_{DC}, X_{DE}\}$ which has to be distributed among 2 locations $[S_1, S_2]$. Every foreign key is represented by a relation between the two corresponding tables. For example the annotation FK (B,C) represents a foreign key if the table C imported from the table B.

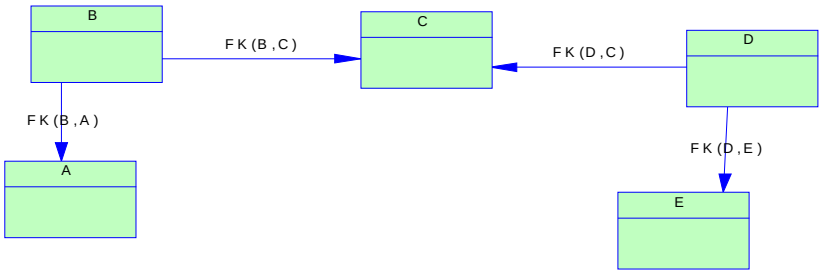


Figure3: Use-case model

The different data extracted from the use case model is:

- Sites $S = [S_1, S_2]$
- Tables T= {A, B, C, D, E}
- Foreign Keys $X = \{X_{BA}, X_{BC}, X_{DC}, X_{DE}\}$

This primary collected data is sufficient to allow the FKDO formula instantiation. It results in the following equation (see Figure4.):

$$\text{Min} \sum X_{IJ} \quad \text{I, J} \in \{ \text{A, B, C, D, E} \} \text{ and } \text{X} \in \{\text{X}_{BA}, \text{X}_{BC}, \text{X}_{DC}, \text{X}_{DE}\}$$
$$/ \text{X}_{IJ} = 0 \text{ if } (\text{I, J}) \in \{ S_N \} \text{ where } S_N \in \{\text{S}_1, \text{S}_2\} \text{ and}$$
$$\text{X}_{IJ} = 1 \text{ if } \text{I} \in \{ S_N \} \text{ and } \text{J} \in \{ S_M \} \text{ and } N \neq M \text{ where } S_N, S_M \in \{\text{S}_1, \text{S}_2\}$$

Figure 4: Instantiation of the FKDO formula

The FKDO algorithm generates 32 different distribution possibilities in the basic generation (without fixing any table on any site). This set corresponds to the $n^{j'}$ possibilities. We suppose that at the advanced generation, the designer decides to have the table A allocated at the site S1 and the table at the site S2. This choice reduces the number of possibilities to 8. The cost of the different allocation is varying between 1 and 4 (see Table. 1).

A	B	C	D	E	Cost
1	1	1	2	1	2
1	1	1	2	2	1
1	1	2	2	1	2
1	1	2	2	2	1
1	2	1	2	1	4
1	2	1	2	2	3
1	2	2	2	1	2
1	2	2	2	2	1

Table 1: Distribution possibilities matching with the designer's preferences

The designers have to choose between the different distribution possibilities generated by the FKDO algorithm. At this step, the decision is taken while considering the different responsibilities areas. For example, in our use-case, the optimal distribution design will be between two different cases which induced the same and the optimal cost (Fig. 6), while considering the designer distribution criteria:

- {{ A, B, C } ∈ S1, { D, E } ∈ S2}, Cost=1;
- {{ A, B } ∈ S1, { C, D, E }∈ S2}, Cost=1;

The last distribution option which presents the same distribution cost than the two selected possibilities is eliminated from the solutions set due to its weakness of distribution. In fact only one data table will be located on the first site and the four others will be located on the second site. Such solution can be interested in some specific cases. Indeed, in the case of a limited treatment capacity site, this distribution solution presents a good alternative for the design considering its low granularity.

The two selected solutions have a one major difference. In fact the first solution locates the C table on the first site while the second solution locates it on the second site. If C represents a functional aspect related to the S1, or if its data is more requested and treated by the S1, then designer can choose the first possibility, otherwise, the designer can choose the second one. The ICDO algorithm presented above will be useful for the designers to take such decisions. In fact, in our use case, only two sets presents the same cost, but generally, the set of solutions that can be obtained in a real case can be more complicated and thus it is difficult to determine the best solution.

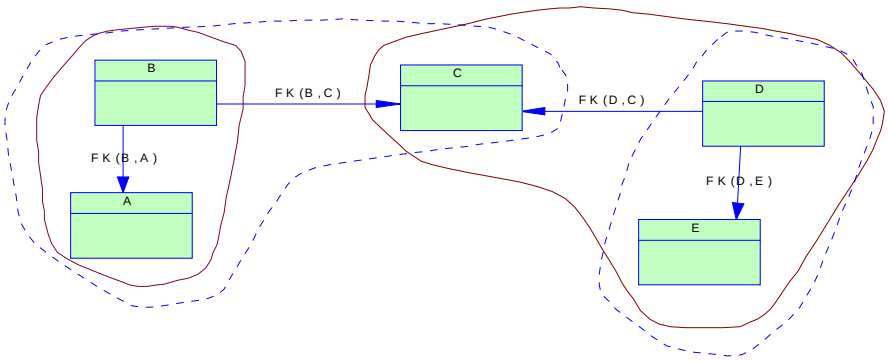


Figure 6: the FKDO optimal distribution design possibilities [Snene03a]

6.2- Integrity Constraint Dependencies Optimisation

The purpose of the ICDO algorithm is to assist designers and developers at the distribution design step. The FKDO algorithm as we have explained below generates multiple distribution sets, which can have an equal transaction's cost. The ICDO algorithm is the second step in which the obtained solutions are examined by increasing the set of used constraints including the different other kinds of constraints. The proposed enhancement of integrity constraint specification (see Chapter VI) is the cornerstone of the ICDO algorithm.

In fact, the specification of the *risk class* indicates the direction of the data flow. Thus, we can easily establish the site that generates transaction dissemination towards the others. The specification of the constraint context determines the sites that are concerned with such dissemination. As the context designates the classes on which the constraint is defined, using the different distribution set, resolve the set of sites that are likely to participate in the transaction check related to the constraint. But depending on the used primitives that are New( ), Update( ), Delete( ), and called by the transaction, this set of classes is not always completely concerned. The range board identifies the different classes, which risk to be transgressed while running each sort of primitives. Finally the cross reference risk table, which regroups the different risk classes, is defined in the range board. It emphasizes the relation between the root class and the classes that will be affected by the transgression of any corresponding primitive. Applying these different enhancement enables designers to determine in a clear way the set of tables concerned with the execution of a primitive and thus to draw the

complete way of the data flow and the number of sites required for this primitive check.

6.2.1- The ICDO constraint

The ICDO formula will be essentially based on the different presented criteria which are essential to get the real cost of each constraint check. As in the FKDO algorithm, the main goal of ICDO is to reduce the number of site relations due to the different constraint's validations. Basically we have to resolve the optimisation constraint that is represented by the following formula (see Figure.7):

$$\min \sum_{i=1,\, j=1}^{n} X_{C_i\, C_j},\ \exists\, IC\left(Root_{(C_i)}, C_j \in IC_{Context}\right) where:$$

$$Context = \left\{C_j / Root\left(C_j\right), \exists\, ope \in \left(Operator\right) / val(C_i)\right\}$$

$$\begin{cases} X_{C_iC_j} = 1\ if\ \left(i \in S_i, j \in S_j\ \right) and\ i \neq j \\ X_{C_iC_j} = 0\ if\ \left(i \in S_i, j \in S_j\ \right) and\ i = j \end{cases}$$

Figure 7: the ICDO formula

Where:

- $Root_{(Ci)}$: represents the root class of the integrity constraint C_i. The root class is the starting point of the cost calculation.

- $IC_{Context}$: represents the set of tables involved in the integrity constraint check process. This set as explained does not include the tables that are not concerned by the called primitive. In fact, as mentioned at the ICDO formula the context is defined by the set $\left\{C_j / Root(C_j), \exists\, ope \in \text{ (Operator)/Val } (C_i)\right\}$. *ope* represents the set of primary primitives (New(), Update(), Delete()) and $val(C_i)$ represents the fact that for this primitive C_i is required for its validation.

- X_{CiCj}: represents the existing relation between the root class C_i and the C_j class that belongs to the *IC* context.

The generated matrix will be in the same form as the matrix generated by FKDO algorithm. But the global cost of the distribution set is calculated from the sum of costs of all constraints validation. In fact, for each proposed solution different constraint's costs must be optimized. Thus, the maximum cost generated by ICDO algorithm will be the number of the different existing cross reference. This cost will be added to the cost generated by FKDO for each distribution solution.

The algorithm of the ICDO formula is as fellows (see Figure.8):

```
indL1=0
L= n^j'
IC ={IC_1,...,IC_N}
cost = 0;
while (indL1<=L){//matrix check
  cost = 0;
  indC1 = 0;
  C = j;
  indC2=indC1;
  while (indC1<C){
     IC1=IC_1
      while ( IC1∉ IC_N ){
      tmpV = 0;
      if indc1[] ∈ IC_Context then {
                  while (indc2 <C){
                   if (indc1[]≠indc2[] AND dependencies ≠NULL)
                                        {
                                        tmpV = tmpV+1;
                                        end if;
                                        }
                    indc2=indc2 + 1 ;
                    end while;    }
      end if;                    }
      tmpV1 = tmpV + tmpV1 ;
      IC1= next IC;
      End while;     }
    IndC1 = indC1 + 1;
    tmpV2 = tmpV1 + tmpV2 ;
   end while;     }
cost = cost + tmpV2;
affect cost (indC1+1 []);
indL1= indL1 + 1 ;
end while ;    }
```

Figure 8: the ICDO algorithm

The generated final cost of each distribution set is the sum of the different integrity constraints dependencies cost. This weighting value assigned to every dependency is varying between 0 and 1. If the two concerned tables are located on different sites and if a constraint dependency exists between them, which means that the running primitives establish a call to a risk class which is the second table, the cost in incremented by 1. Otherwise, the cost remains changeless.

6.2.3- ICDO Use case

We propose the same ICDO use case then the used one in the FKDO algorithm. We propose an enhancement of the example proposed above with the necessary components for the good ICDO running. As explained in the FKDO use case, 8 solution sets are generated and two of them have the lowest connection costs.

The proposed integrity constraint is as follows:

$IC_{Expression}$:

$\forall\, b \in B, \exists\, a \in A \wedge c \in C \;/\; b.a \in b.c$

Context (B, A,C)

Figure 8: the use case constraint

This constraint implies that for every instance *b* belonging to the class *B*, it exists an instance *a* belonging to the class *A* and an instance *c* belonging to the class *C*, while the *b.a* relation is a subset of *b.c* relation.

The range board of this integrity constraint is:

	New( )	*Update ( )*	*Delete ( )*
B	**Yes**	**Yes**	**-**
A	**-**	**Yes**	**Yes**
C	**-**	**Yes**	**Yes**

Table 2: the use case range board

This range board defines the different risk zones related to every possible primitive. These risk zones are related to the defined integrity constraint below. For example, in the case of a transaction calling a New() primitive, only the class B will present a transgression risk.

Risk class	**Primitive**	**Cross reference**
B	New ( )	A,C
	Update ( )	A,C
	Delete ( )	-
A	New ( )	
	Update ( )	B
	Delete ( )	B
C	New ( )	-
	Update ( )	B
	Delete ( )	B

Table 3: the use case cross reference risk table

The matrix generated by the ICDO algorithm is similar to the FKDO matrix. The cost is obtained from the sum of the different costs related to the different integrity constraints transactions. By examining the obtained ICDO cost related to the two

optimal distribution sets underlined by FKDO cost matrix, we observe that two sets present the lowest connection cost (see table.4). These sets are:

- {{ A, B, C } ∈ S1, { D, E } ∈ S2}
- {{A,B,C,E}∈ S1,{D}∈ S2}

A	B	C	D	E	ICDO Cost
1	1	1	2	1	0
1	1	1	2	2	0
1	1	2	2	1	4
1	1	2	2	2	4
1	2	1	2	1	8
1	2	1	2	2	8
1	2	2	2	1	4
1	2	2	2	2	4

Table4: ICDO connection's costs

Finally, the distribution combination generated by ICDO algorithm is the following (see figure.9). This combination guarantees the lowest communication cost between the different sites. This cost is essentially due to the different integrity constraint check calls. In this distribution, only one integrity constraint check will requires the establishment of a site call.

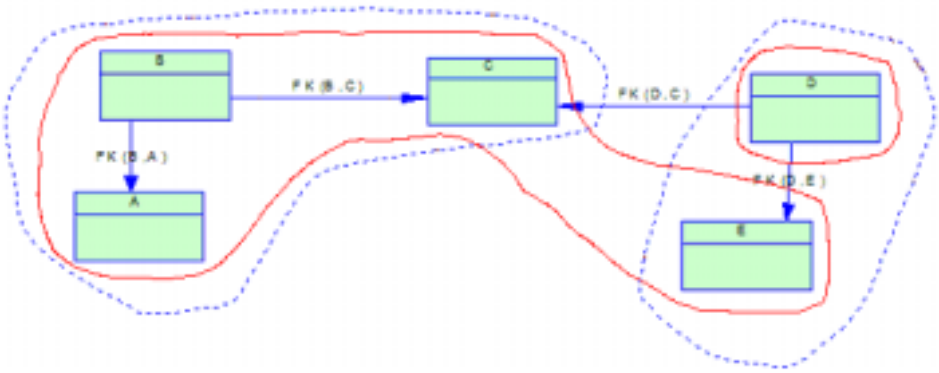


Figure 9: the ICDO optimal distribution design possibilities

6.3- Global Integrity Constraints Optimisation

The combination of FKDO and ICDO, gives us a distribution design proposition that presents the lowest possible cost while respecting the produced system integrity. All the treated constraints are considered as having the same weighting. But, in practice the importance accorded to each integrity constraint is different from others. For example, in a system of human resource management, the constraint consisting in the verification of the user identity is used more frequently than the constraint related to

the creation of a new employee. This vary case can vary depending on the kind of the organization. In fact, in the case of an enterprise offering temporary job, the creation of new employee can be used more often than the user login. We can consider that two main components determine the constraint weighting. The first is the constraint frequency. This frequency is calculated from the number of this constraint calls and the number of total calls. The second component is the importance accorded to this constraint by end-users and designers. In fact some constraints that can be rarely used can deal with critical data and thus must have an important weighting. In this case, the distribution generated set will take into account this importance as a call cost and the data required for this constraint will be more probably located at only one site. The GIDO formula is the following (see Figure 10):

$$Min(Global\ Cost) = Min \sum_{i=1,j=1}^{n} X_{ij} + Min \sum_{i=1,j=1}^{n} X'_{C_iC_j}$$

$$where: X = FK(I,J)\ and\ X' = IC\ (Root(C_i) \wedge (C_j \in IC_{Context}))$$

$$and \begin{cases} X_{ij} = 1\ if\ i \in S_N\ and\ j \in S_M\ and\ N \neq M\ else = 0 \\ X'_{C_iC_J} = Wg_{X'_{CiCj}}\ if\ i \in S_N\ and\ j \in S_M\ and\ N \neq M\ else = 0 \end{cases}$$

Figure 10: the GIDO formula

Where:

- $Min(Global\ Cost)$: represents the minimum global cost obtained by the addition of the obtained FKDO algorithm and ICDO algorithm adjusted by the different constraints weightings.

- X_{ij}: represents the FKDO cost. As the integrity and the availability of the database is firstly depending on foreign key consistencies. The cost of each foreign key dependency must be higher than any other constraint. Thus, this cost still unchanging and equal to 1.

- $X'_{C_iC_J}$: represents the ICDO cost. This cost is equal to $Wg_{X'_{CiCj}}$. This weighting is: number of C_IC_J call / total constraints call. This cost can evolve depending on the call frequency. Firstly, the number of integrity call is an estimated number based on designers' hypotheses. Then this cost can be calculated at runtime and thus new distribution combination can be generated. This weighting is adjusted according to the importance of the constraint.

The GIDO algorithm is the following (see figure 11):

```
indC=C-1;
indS=1;
indC1=0;
indL1=0;
while (indL1<=L){
     cost=0;
     ndC=0;
     while (indC1<C){
               indC2=indC1;
               check dependencies between tables at indC1 and
indC1+1;
               while (indC2<C){
                 if(m[L][indC1]!=m[L][indC2+1]&&dependencies !=
null)
                                    tmpV=1;
                                   else tmpV=0;
                                   C2++;    }
                 indC1++; }
              cost=cost+tmpV;
   affect cost; }
   indL1++;}
L= n^j'
IC ={IC_1,...,IC_N }
Wg_IC1 =1;
while (indL1<=L){//matrix check
  cost = 0; indC1 = 0;C = j;indC2=indC1;
  while (indC1<C){
     IC1=IC_1
      while ( IC1∉ IC_N ){
      tmpV = 0; Wg_IC1 =1;
      if indc1[] ∈ IC_Context then {
                  while (indc2 <C){
                   if (indc1[]≠indc2[] && dependencies
                     ≠NULL)
                                    {
                                    tmpV = tmpV+Wg_IC1 ;
                                    end if;
                                    }
                    indc2=indc2 + 1 ;
                    end while;    }
      end if;                   }
      tmpV1 = tmpV + tmpV1 ;
      IC1= next IC;
      End while;     }
    IndC1 = indC1 + 1;
    tmpV2 = tmpV1 + tmpV2 ;
   end while;     }
cost = cost + tmpV2; indL1= indL1 + 1 ;
affect cost (indC1+1 []);
end while ;    }
```

Figure 11: the GIDO algorithm

The algorithm is composed of two parts. The first is the FKDO algorithm which is still unchanging and conforms to the algorithm purposed (see figure 2.2). The second part is the ICDO algorithm version in which the different weightings are taken into consideration.

6.3.1- GIDO Use case

The set of costs obtained from the FKDO and ICDO algorithms matrix are added to obtain the global cost. In the presented use case, the weighting of the used integrity constraint is equal to 1. In fact, as the use case contains only one constraint, the number of the constraint calls divided by the total number of call is equal to 1. Between the three generated sets, the global costs are 1 for the second set, 2 for the first set and 6 to the 7th set (see Figure 12).

A	B	C	D	E	FKDO Cost	ICDO Cost	Global Cost
1	1	1	2	1	2	0	2
1	1	1	2	2	1	0	1
1	1	2	2	1	2	4	6
1	1	2	2	2	1	4	5
1	2	1	2	1	4	8	12
1	2	1	2	2	3	8	11
1	2	2	2	1	2	4	6
1	2	2	2	2	1	4	5

Table4: global connection's costs

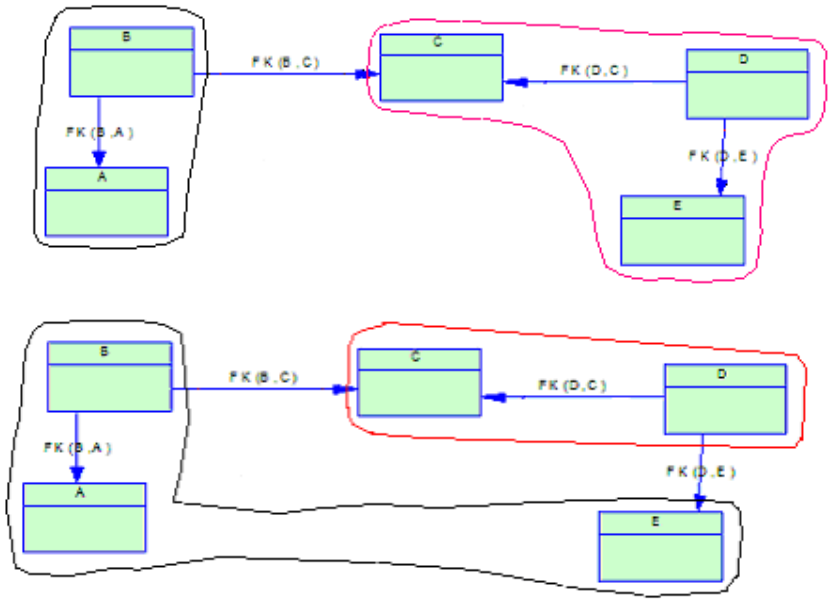


Figure 12: GIDO distribution sets example

Finally, the optimal distribution set is composed of the set {{A,B,C}∈ S1{D,E}∈ S2} (see Figure 13). This set presents the lowest constraint calls' cost. Moreover, this distribution set presents the lowest risk to the system integrity. In fact, the specified integrity constraint has its needed set of data located on the same site.

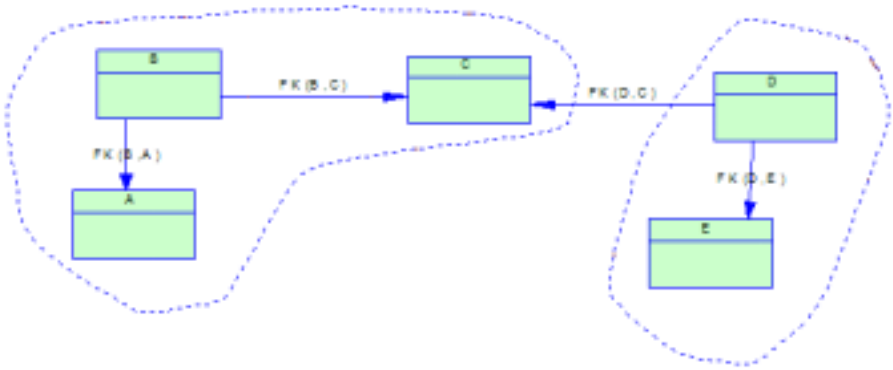


Figure 13: the optimal distribution combination

In addition, only one foreign key call is needed to establish a call between the two sites. By this fact, if one of the two sites is not responding, the other can continue working normally as long as there is no call established between them.

6.4- Conclusion

Different approaches exist to help designers at the distribution step of a distributed information system project. These approaches are generally based on the "designer experiences" and all their purposes are to establish a set of different concepts, without giving to the designer a real process able to assist him. Our approach as demonstrated above is focused on three main goals. The first is to give the designer a real view of the distribution system. This is done by generating the different distribution sets and comparing them according to their respective transaction costs. We consider that the most important system transactions are generated from the integrity constraint verification and validation. Indeed, any function requires before its running, to be verified that it does not transgress the data integrity and consistency. The second goal is to underline the most advantageous solution while giving the possibility to the designer to interact and influence this solution. This flexibility is concretized by the ability of the designer to fix needed data on predefined site and to order the different integrity constraints by determining the respective weighting. The third goal is to obtain the lowest distribution cost while maximizing the system integrity and availability.

Next chapter presents the benefits of the use of these algorithms and their results on the distributed information system design and implementation project. In fact, these algorithms help designers and developers to have a clear idea about the different distribution sets and their related cost, but different other constraints have to integrated

to achieve the distribution. These constraints are collected from different project levels and must be communicated to the different partners. For this purpose, we define the needed knowledge at every project level and its result on the other levels.

Chapter VII
The Overlap Knowledge Pattern for Distributed Information System Design

The distributed information system knowledge pattern represents the common knowledge shared between designers and developers. This knowledge represents different design and implementation aspects that must be placed together to ensure a best resolution of the distribution problem. In fact, the distribution step is carried out conjointly at the design and the distribution step without a real coordination between them. This is due to the lack of methodologies and of tools which can help the different project partners. Indeed, the existing methodologies focus on the expression way of distribution instead of answering the question: how to obtain the distribution? The existing data distribution approach proposes different algorithms for data fragmentation and allocation [Özsu99]. This approach does not consider the different functional aspects and is highly complicated to be achieved without a support tool. The defect of these approaches is that they separate the functional and the data aspects which are harmful to the project continuity. In our distribution approach, this continuity is ensured by the overlap knowledge pattern. As explained in chapter IV, this pattern has three main components. These components are the site relations, the data locations and the integrity constraints. These three components represent different system facets. Any decisions related to these components must be taken jointly be the designers and the developers in order to ensure its correctness.

In this chapter we present the existing connections between these facets, the contribution of the set of the proposed algorithms to the distribution design and the benefits of the overlap knowledge pattern.

7.1- The overlap knowledge pattern components

Basically the overlap knowledge pattern is composed from two zones. The first contains the knowledge used by designers and which has to be transmitted to developers. This knowledge is usually not expressed at the design step due to the deficiency of the offered design structure which disables the expression of such information. The second is the developers knowledge which surrounds the technical used environment. This knowledge has to be communicated to designers at the design step to be taken in consideration. In fact, this knowledge has an immediate influence on the design result due to its characteristics. For this purpose, we define the needed knowledge overlap between the project partners (see Table1). The (•) designates the knowledge that is initially used by the participant at its project step. The (+) designates the knowledge transmitted to the other partner which is not initially exploited at this step. We underline also the knowledge generated from the integrity constraints design and implementation step.

	Design	Implementation	Integrity constraints
Triggering site	•	+	➘
Data location	+	•	➘
Permanent connection	+	•	
Temporary connection	+	•	
High use frequency	•	•	
End-users queries	•	+	
Design Justification	•	+	➘
Critical data	•	•	
Constraint Weighting	•	+	➘
Technical constraint	+	•	
Limited treatment capacity	+	•	
Network performance	+	•	
Site capacity	+	•	
Validation site	•	+	➘
Security constraint	+	•	➘

Table 1: Overlap Knowledge Exchange

As shown in table1, designers and developers share also some knowledge. Some of this shared knowledge concerns different project partners depending on the environment from which this knowledge is extracted. For example, the critical data which exists in both design and development steps do not have the same significance for them. From one side, the critical data for designers represents the set of data which has a high importance level accorded by the end-users. From the other side, the critical data for developers is the data that represents a high importance for the system running. To establish a better distinction between the designer and the developer

knowledge, we present a non-exhaustive knowledge list. This list shows, if exists, the interference of the integrity constraints at the design and implementation levels.

7.1.1 Designer knowledge

We present the usual knowledge used by designers at the design step of a distributed information system project. This knowledge is the result of the mix of the information extracted from the analysis and the design steps. The existing design methodologies do not offer enough flexibilities and structures to communicate this knowledge to the developers throw the design schema.

- Triggering site: this knowledge represents the set of information related to the different site of integrity constraints triggering. In fact, designer have to choose the activation site from the set of existing sites, in order to ensure the well running of the integrity constraints while respecting system characteristics. As explained in *chapter I*, different site and system level can be a triggering site. Usually, three level of triggering exist. The first is at the interface level. This can be done by choice restriction. It means that only the data which does not present any data transgression risk will be presented to the end-users for selection. This can be done only in the case where users do not have to input data such as for the identification functionality. The second level is located at the application site. The data check will be done by confronting the user-input data with the system data. This kind of check is usually done to verify for example the user session data. Indeed, some systems close the users session after an inactivity lap of time. Finally, the last level of triggering is the data location site. This check is done by verifying if the user data presents a risk for database transgression. This way of checking is used while attempting to update or to insert new data.

- High use frequency: this knowledge is obtained from the analysis and the design steps. In fact, end-users can designate the most used data and thus the data use frequency. This information is important for the system survival. Indeed, if this data is unavailable, one or more subsystem will be out of order. For this purpose, designers have to underline this data to developers in order to implement it in a manner that ensures its best availability.

- End-users queries: due to some organisational constraints, users can require the allocation of data to a specific site. These constraints can of different ranges and importance. For example, confidential data can be subject to user queries. In fact, usually users predefine the host site for such data in order to ensure to maximum data confidentiality and the prevent the data from being allocated on sites that present a high risk for it.

- Design justification: this knowledge surrounds the design purpose. It presents the main goal of design and arguments the designers choice. In many cases, the produced design schema still difficult to be well understands by the developers

due to the lack of information about the decisions taken at the design step. For example, the data related to an attribute “address” can be designed in different way. It can be grouped into one unique attribute which regroups the data about the street, the street number, the town and the country, under the form of a text field. Or, it can be dissociated over different attributes, such the town attribute, the street attribute, the country attribute and the street number attribute. This dissociation is done in order to ensure the running of some functionality which are based on or more of these attributes. It can be done also for an evolution purpose. This set of functionalities can be required by end-users for a later evolution. For developers, as this evolution requirement is invisible to them, this way of design can be harmful to the system performance and they can choose to regroup the different attributes under the form of a unique one. In this case, this functionalities upgrade will not be possible. The design justification has to be done for every design decision taken according to any system requirement which is hidden to the developers.

- Critical data: the critical data represents the set of data that represents a high importance level for end-users. It can be necessary to regroup different subset of these critical data and to allocate them on predefined sites. For this purpose these data have to be clearly indicated to the developers in order to respect the association done between these sets.

- Constraint weighting: it represents the knowledge about the constraints importance and way of use. In fact, constraints can be invocated in different frequencies and with different importance. For example, we can suppose that an integrity constraint that checks the correspondence between the user session and the allowed functionalities will be run only one time at the user connection. But, the correct check of this constraint is capital for the good running of the system. In fact, in the case of a missing constraint check, the end-user connected will be able to run unauthorized functionalities. The weighting is thus determined according to the relation (frequency, importance). It can be done by examining the running mode and the behavior of the end-users. This knowledge actively involves in the decision of data distribution.

- Validation site: this knowledge concerns the location of the integrity constraints validation. Indeed, the validation can be carried on different sites depending on the specification of the constraints. This validation can be done on the triggering site or any other site involved in it. This decision represents the final step of the integrity constraint flow. Usually the validation must occur on the site where the validation result will be used. But, in some special cases, this validation has to be located on a different site. Designers have to specify it for developers to ensure that the constraints are implemented in the correct way.

7.1.2 Developer knowledge

This knowledge is extracted from the technical project environment. It represents a set of information about the implementation and the runtime environment that must be considered at the design step. Such knowledge influences the design result. In the case of the non-respect of this knowledge, developers have to carry on the design schema a set of modifications in order to adapt it to their constraints which can be harmful to it. The most important knowledge is:

- Data location: in the classical distributed information system design methodologies, the manner and the result of the data allocation are chosen by developers without involving the designers in this step. Developers have to carry on this distribution without having enough knowledge about the distribution purpose. They usually produce this distribution only according to the technical and system performance constraints.

- Permanent and temporary connection: depending on the existing technical environment, the connection kinds offered are varied. Permanent connection is a connection established continuously between two or more sites and is requested in the case of frequent exchange between these sites. Temporarily connection is a non-permanent connection established between two or more sites. This connection kind is used in the case of the frequency of data calls between these site is low. Depending on the technical and distribution environments, the decision of the kind of the connection can be different from the requested one. For example, every site presents a limited number of possible simultaneously connections. In the case of a high set of sites, the number of permanent connections has to be decreased in order to prevent this site from being unreachable by the sites that have temporary connections.

- High use frequency: the technical aspect of the different data calls and the integrity constraints checks require the establishment of a set of data calls that can increases the data use frequencies. Moreover, the check of the technical constraints requires new data relations. These relations influence the use frequencies. To reduce the different calls inter-sites, developers can decide to locate the involved data on the same site.

- Critical data: the critical data for developers represent the necessary data for the system running. Usually this data is used to nit the system or to init the user session. For example, the use of J2EE platform requires the creation of a set of data composed from the user identification data under the form of a temporary table and located on the site of the function call before permitting to the user to connect to the system. This data set represents a subpart of the needed data to the system initialization. In the case of the impossibility of getting this data, the system initialization cannot be realized and the connection function still disable [Pawlan00].

- Technical constraint: some platforms require special data adjustment in order to satisfy their constraints. Depending on the used technical platform, this data is varied. In the case of the use of a J2EE platform [Pawlan00], a numeric OID identifier must be added to every data table. This OID is used for the queries running and for the data localization. In fact, the proposed method for data selection which is *Find_By_Primary_Key()*, requires a numeric attribute to be run. If the primary keys defined in the design schema are not numeric, the select queries will not be able to run.

- Limited treatment capacity: the treatment capacity of every site that belong the system depends on the used platform and the physical characteristics of this platform. A high number of functions triggering and of integrity constraints checks on a site can generate an overload of this site. For this reason, developers have to manage the site of allocated treatment on every site in order to ensure a balanced load between the different sites.

- Network performance: the network performance depends on the amount of the data flow between the existing sites and the connection quality. The decision about the permanent or temporary connection can also influence the performance. In the case of a low network performance, the risk of disconnection must be well considered at the distribution step. In fact, to ensure the system running continuity, the number of calls and connections between the different subsystems must be decreased.

- Site capacity: this knowledge is resulting from for example in a special case of distributed information systems which is the distributed embedded system, every site has a predefined capacity which can not be exceeded. Due to this fact, the amount of allocated data must respect the capacity of the site in order to prevent it for the crash. Developers have to redefine the allocation by allocating the critical data firstly and then the other sets.

- Security constraint: depending on the used platform, some security constraints have to be considered. For example, in the case of using a distributed platform composed from an Oracle distributed database server coupled with a J2EE Server, the data concerning the user session must be allocated on the same site then the security check triggering site [Kassem00]. If this data is located on a different site, the data server will reject the user connection. Such constraints must be considered at the data allocation step.

7.1.3 Overlap knowledge

As explained above different knowledge sources exist across the distributed information system project. In fact, the knowledge can be obtained form the environment surroundings the project or from the other existing knowledge (see Table.2). For example, the site treatment capacity is an information obtained from the existing set of sites and the integrity constraint validation site is an information

obtained from the designers knowledge surroundings the integrity constraints environment. Two main knowledge sources are defined which are: the design step and the implementation step. The knowledge obtained from the first step is the different needs expressed by the end-users such and underlined in the design schema. The implementation step provides knowledge about the used technical platform, the implementation languages and the sites technical characteristics. This knowledge must be exchanged between them in order to obtain a well-designed and implemented distributed information system. In fact, the structure offered by the different design methodologies does not take into account the knowledge surrounding the project. This defect of knowledge exchange is harmful for both the design and the implementation of such systems. The overlap knowledge contains the knowledge initialized at their corresponding step and transmitted to the other step.

	Design	Implementation
Triggering site	•	The way of implementation and the site of triggering are predefined
Data location	Designers involved in the Decision of data allocation. They can specify special allocation cases.	•
Permanent connection	Designers can request this kind of connection in the case of high estimated call frequency between two or more site.	•
Temporary connection	Designers can request this kind of connection in the case of low estimated call frequency between two or more site.	•
High use frequency	The frequency is adjusted according to the system requirement.	The frequency is adjusted according to the design requirement.
End-users queries	•	Developers are informed about the end-users queries and take them into account at the development step
Design Justification	•	It ensures a better understanding of the produced design schema.
Critical data	The system critical data are underlined at the design step.	The design critical data is distributed according to the predefined priority.

Constraint Weighting	•	Developers have a clear specification of the integrity constraints priority and importance.
Technical constraint	Designers integrate the technical aspect in the design step to minimize the number of modifications carried on the schema at the development step.	•
Limited treatment capacity	Designers reduce the number of functions and constraints triggering on the site with a limited treatment capacity.	•
Network performance	Depending on the performance, designers adapt the distribution of different data on the sites.	•
Site capacity	Designers must proceed to the distribution according to the priority of data assignment to not exceed the capacity of each site	•
Validation site	•	The implementation of the output of integrity constraints minimize the data flow cost
Security constraint	The produced design schema includes the predefined security constraints	•

Table2: the overlap knowledge exchanges

The different algorithms proposed in the chapter VI, contribute to the clear definition of some of knowledge components. In fact, this set of algorithms is used at the distribution step which represents the most important overlap zone between the design and the implementation step. The consequences of these algorithms are expressed in table 3.

	Integrity constraint
Triggering site	The triggering sites are defined at the distribution step and not at the integrity constraints definitions. Their definitions are done according to the other design and system constraints
Data location	The algorithms generate different sets of distribution possibilities. For every possibility, a data flow cost is calculated according to predefined criteria and to different weighting assigned to every integrity constraint that requires this data.

Design Justification	Designers can allocate different tables on different sites before beginning the distribution design. The algorithms take into account this requirement and have enough flexibility to enable them expressing such constraints into the distribution step.
Constraint Weighting	The assignment of weighting to the constraints is done at both the design and the development step. Designers usually have an estimation of the priority and importance of the weighting. Developers adjust these weighting according to real frequencies of constraints calls.
Validation site	The validation site is an important component of the integrity constraint specification. This information is not expressed at the produced enhancement specification. The choice of the site is taken by designers and developers while considering the different design and system constraints
Security constraint	The security constraints have to be considered at the beginning of the design step in order to be easily integrated in the design schema. These constraints can be considered as a subpart of the integrity constraints due to their importance to the system integrity.

Table 3: the integrity constraint impact on the overlap knowledge

7.2- Knowledge overlap pattern benefits

The knowledge overlap pattern is used to collect and to share the different common knowledge of the project partners. These partners have to resolve together the set of constraints which constitute this pattern. As explained above, the consequence of optimising the set of integrity constraints is the resolution of the different constraints collected from the other components. The set of proposed algorithms (see chapter VI) helps both designers and developers to satisfy these constraints by providing them with an optimal distribution set. This distribution design is done according to the different constraints.

- *Distributed information system availability*: reducing the number of calls between existing subsystems, increase the subsystems autonomies. In the case of the impossibility of establishing a connection with a subsystem, the others can continue running as long as no data is requested from this site or no integrity constraint need data from this site to be validated. Indeed, the distribution attempts to assemble the data according to the functional zones while minimizing the sites dependencies.

- *Distributed information system integrity*: the overlap knowledge pattern is mainly based on different forms of integrity constraints. For this purpose we have defined several enhancements for integrity constraints specification. These enhancements enforce integrity constraints semantics and adapt them to the distribution context. As the first distribution aim is to gather the different needed data by a constraint on the same site, the data integrity checks and controls require a low number of connections between different subsystems. In

the case of the disconnection of a subsystem, it can continue running locally and ensuring it is own data integrity.

- *Distributed information system design:* the overlap knowledge pattern contains a clear information system distribution design process. This process is based on the different presented algorithms which generate the set of distribution possibilities according to the different existing constraints and queries. These possibilities are compared on the base of their respective cost. This cost is calculated from the number of call that will be needed between the set of subsystems to satisfy the validation and the verification of the system integrity constraints.

- *End-users queries*: the pattern considers the end-users queries as the principal goal to be satisfied. In fact, these queries have to be included at the design step and well explained in order to inform developers about their subjects and their utilities. These constraints are easily added at the distribution design generation step by either increasing the constraint weighting and so maximising the possibilities that the data belonging to the constraint will be allocated at the same site, or by allocating directly these data on the predefined site.

- *Distribution design tool*: actually there is no existing CASE tool for distributed information system design. The M7TOOL represents an innovative tool in this area. This tool is build under a collaborative form which helps designers to easily integrate their project subpart. This tool is equipped with a clear process that generates the distribution according to some predefined criteria. Designers have the ability to modify and adjust these criteria and thus to simulate the different distribution situation and conclude which one is the best for the conceived system.

- *Flexibility of design:* the proposed distributed design schema can be easily configured to be adapted to the end-users queries. This possibility ensures a high design flexibility level. In fact designers or developers can indicate their constraints at the design generation step and the different algorithms will take them into account. The cost of every constraint can be adjusted by modifying its corresponding weighting and thus modifying its importance for the distribution design generation.

Although the importance of these aspects, there is no existing approach that takes into consideration the knowledge as a part of distributed information systems projects. The lack of existing tools considering this knowledge contributes to this discontinuity between design and development steps. To mend this gap, designers and developers have to communicate through processes and tools to ensure the continuity of the project achievement.

Next chapter presents a CASE tool that integrates the different distribution algorithms that are used for the distribution design. The integration of these algorithms into the

tool is the core of the distribution problematic we try to answer. Indeed, these algorithms are the first step of the distribution answer. To achieve the distribution generation, a first communication has to be established between the project partners. This CASE has been developed as a distributed information system. Its design and implementation have been done according to the defined framework. Thus, this tool presents two layers: the first is an experimentation system designed and implemented according to the defined algorithms. The second is a CASE Tool that integrates the distribution design framework with its different algorithms.

Chapter VIII
The framework experimentation: M7TOOL CASE

The previously proposed framework of distributed information system has been implemented in collaborative a CASE tool called M7TOOL. One of our main aims with M7Tool is to provide the participants of the distributed information systems projects with possibilities to work cooperatively, to have assistance at the distribution design, to share and reuse the implemented specifications and to have easy accesses to their workspaces from any Web browser. The particularity of this tool is that it has been itself conceived as a distributed information system. Its design has been done according to the specified algorithms. Indeed, this tool has been the first experimentation of the different algorithms and its goal is to provide assistance for distributed information systems projects participants to use these algorithms. For the experimentation purpose, three main data sites have been specified for this CASE tool based on the different zones of functionalities.

Several definitions and descriptions of CASE (Computer Aided Software Engineering) were given in research and industrial projects. A very general definition is provided by [Meyer88] " tools and methods to support engineering approach to software development at all stages of the process". This definition includes all kinds of computer-based support for any of the managerial, administrative, or technical aspects of any part of an information system project. A classification on what is and what is not CASE has been proposed by Harmon [Harmon93]. Initially, this classification was structured into four categories as shown in Figure 1.

The first category of CASE tool “Analyst-Design of workbenches” typically support structured analysis and design. This kind of tools often provides support for several specification models, e.g., static diagrams, state chart diagrams and life cycle diagrams. Life cycle tools focus on giving support all the way from a diagrammatically oriented systems analysis stage through to generation of code. The I-CASE category generates executable code from mostly textual specification in very high level programming language. It is based on a repository that store information and data resulting from all the system building process [Dodd94]. All of these CASE categories are focused on software engineering steps. They rarely offer the possibility to focus on Database design and implementation but generally propose a direct mapping from Oriented Object specifications to a physical diagram.

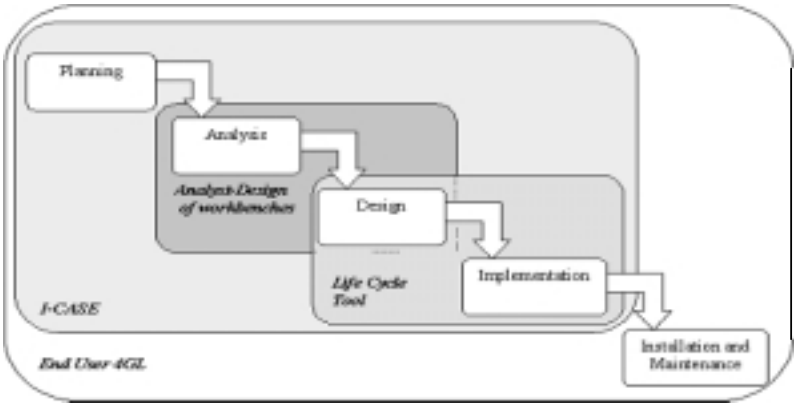


Figure 1. Categories of different case tools

For reducing these gaps and increasing CASE using the notion of 4GL’s has been proposed. It is dedicated to Information systems developers and for complex applications. It makes possible to design both data flow and functional process. So it includes the data consistency and the functional robustness.

In practise, CASE tools are less used than envisaged [Davenport92]. They still have many serious problems such as interoperability, reusability and generally a hard training. The inflexibility of this contemporary CASE technology is due also to the limited support for teamwork and distribution design. Generally, designers split the system specifications into disjoint subsets. Each subset is considered as an individual task, which makes them locked and impossible to modify by others. Many studies seem to indicate that independence between developments objects is extremely difficult to achieve [Frost94]. The efforts used for the schema integration of a project generate a significant over cost due to the covered zone. The distribution design still not supported by any existing CASE tool. Indeed, such CASE tools purpose different concepts and tools to express the distribution without giving any methodologies or assisting designers to choose the most suitable distributed schema sets.

The chapter is further structured as follows: the first section presents the technical architecture and the used platform for the tool implementation. In the second section, we present the result obtained at the distribution design step.

8.1- M7TOOL architecture

The implementation mode of the collaborative architecture of the CASE tool is restricted by some technical constraints. In fact, the tool has to be based on Web collaborative services to ensure its interoperability across the most used operating systems such as Microsoft windows, Unix, Linux and OS [Haselbring98]. Also, the Quality of Services QoS, answering time, support for both transient and persistent transactions and the mode of communication have to be satisfied [Schmidt00].

The J2EE specification from Sun [Kassen00] based on Enterprise Java Beans (EJB) technology satisfies our constraints and is considered as the best implementation architecture for our framework. The EJB architecture logically extends the Java Beans component model to support distributed applications. It supports application development based on multi-tier, distributed object architecture in which most of the application logic is moved from the client to server. The application logic is partitioned into one or more business objects that are deployed in one or more application servers [Rahul01]. Moreover it offers a range of perfectly adaptable software components for our needs which are the Entity Beans (EBs) and the session Beans (SBs) [Pawlan00], [Rahul01].

Enterprise Java Beans architecture supports both transient and persistent objects. A transient object is called a session bean, and a persistent object is called an entity bean. A session bean is created by a client and in most cases exists only for the duration of a single client/server session. A session bean performs operations on behalf of the client, such as accessing a database. Session beans can be transactional, but they are not recoverable following a system crash. Session beans can be stateless, or they can maintain conversational state across methods and transactions but it has to manage its own persistent data. An entity bean is an object representation of persistent data that is maintained in a permanent data store, such as a database. A primary key identifies each instance of an entity bean. Entity beans are created either by inserting data directly into the database or by creating an object. Entity beans are transactional, and they are recoverable following a system crash. They can manage their own persistence.

The use of a distributed environment in the development of the M7TOOL implies a multilevel architecture (see Figure.2). These levels are:

- the presentation level which contains the various graphical user interfaces;
- the business level which treats the data coming from the presentation level;
- the service level which validates the various information collected from the business level;
- the access regulation level which operates like a transitory memory before the final safeguard and which ensures the semantic validation;
- the data share level, which collects data from the different locations to rebuild a common repository for the different queries;
- the storage level which ensures the storage and the syntactic validation of the data.

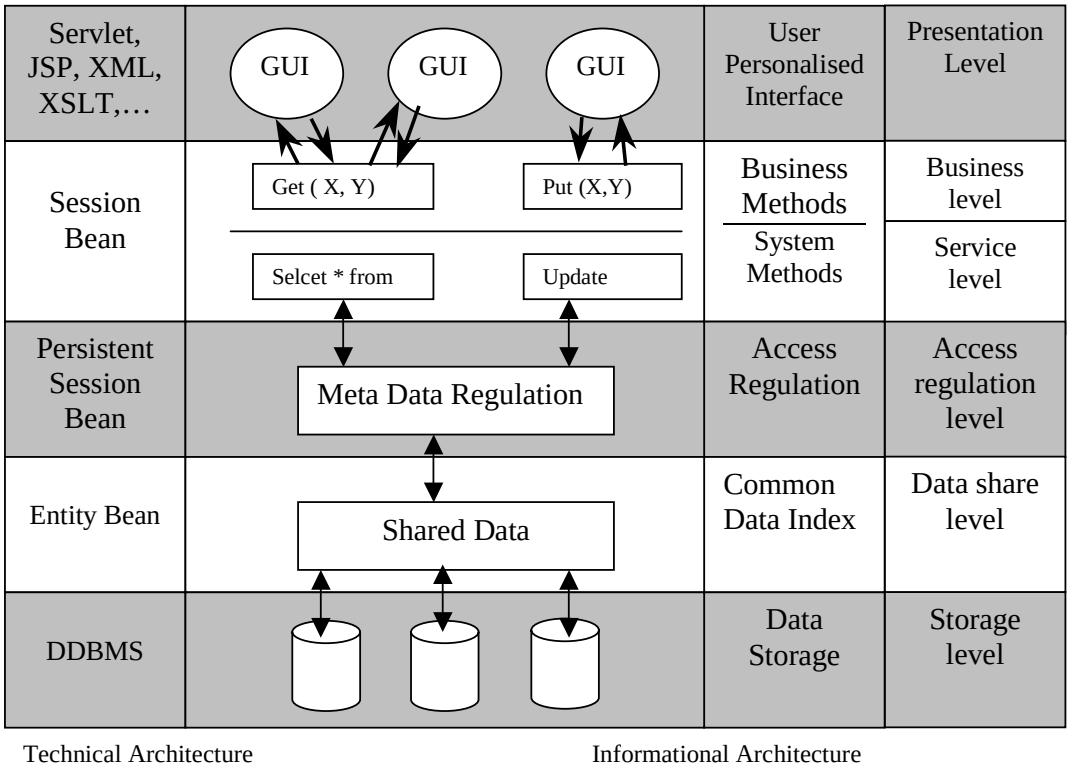


Figure 2. Technical architecture of M7TOOL implementation [Snene03c]

Below we give in detail different levels existing within the framework architecture.

8.1.1- The presentation level

The presentation level refers to those components responsible for creating and managing an application interfaces with its users. The different graphical functionalities of this level are integrated in a signed applet. The applet is signed to avoid the rejection of the connection by the user host machine. This applet communicates with the business level via servlets located on the same application server. In this level, we integrate some business rules for the users identification. For this purpose, we define four basic roles managed dynamically. It means that any user can be associated with any of these roles in a specified project and a different other roles. These roles are respectively:

- Specification Administrator: he holds all the rights on a specification;
- Schema Administrator: he holds all the rights on one or several diagrams;
- Designer: he works only on the diagrams that are authorised for him. The functions which are allowed to him are the writing and reading rights on these diagrams;
- Viewer: he can only read diagrams. Typically this role is associated with End-users.

We defined a Hierarchical organisation that respects the human structure during the realisation of a real Information System project [Sierhuis96]. We structured a specification into a set of schemas, each one dedicated to a subset of the system. The hierarchical model could be extended into many other layers according to the need expressed. We identified four roles that offered optimal possibilities for each designer to participate in the project. These roles are strongly inspired from the IS project organisation (see Figure. 3).

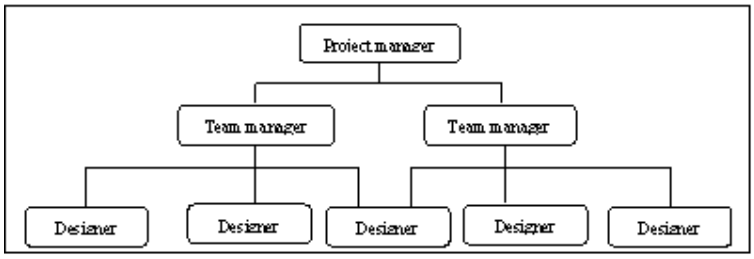


Figure 3. Design team organisation

The first role is the Specification Administrator. It corresponds to the role of the project manager. He is responsible of tasks assignment between different team manager and/or designers. He has full rights on the specification (see Figure. 4). His main task is to synchronize the data dictionary to avoid the semantic integration problems. Most of integration problems, results from the synonymous and the homonymous used by different designers during the classes and attributes definition phase. The establishment of a universal business repository related to the project, minimises the possibility to observe usual conflict between different terms used by designers. This repository represents also a knowledge base for the project domain. It increases the reusability of the conceptual diagram issued from the design phase, given that we defined clearly each term used in our diagrams.

The second role is the Schema Administrator. It corresponds to the role of team manager. He is in charge of managing different diagrams related to the specific sub domain of the project. He has full right access on his assigned schemas and restricted rights on the complete specifications (read, write by creating new schemas in his sub domain). He produces a finer granularity repository than the global one. He supervises the schema evolution and the different design process (static diagram, sequence diagram, state chart…).

The last active role in our model is the Designer role. He has limited right on his diagrams and on the whole specification. He cannot view other schemas unless he is authorised. This restriction is made in order to provide informal designer interaction.

We define also the Viewer role. He has only the reading right on authorised schemas. This role is typically dedicated to the final users since it allows them to supervise and to control the system design progress.

User	Spec. Admin	Schema Admin	Designer	Viewer
SPEC				
Create	Yes			
Write	Yes	Yes	Yes	
Read	Yes	Yes	Yes	Yes
Delete	Yes			
Add User	Yes			
SCHEMA				
Create	Yes	Yes		
Write	Yes	Yes	Yes	
Read	Yes	Yes	Yes	Yes
Add User	Yes	Yes		

Figure 4. Hierarchical organisation in M7Tool

8.1.2- The business level

It regroups the different methods offered to the connected users. It managed the user session by establishing the methods calls. The data collected from the presentation level are treated and sent to the service level. It is an intermediate level between the presentation level and the service level. It encapsulates the various treatments necessary to the design activities, which are refined according to the profile of the corresponding user.

The architecture is based upon the separation between the data and methods. To illustrate its capacities, we present the situation of a role modification during a user session. The technical architecture of this level is composed by SBs. SBs contain the methods that correspond to the user-session profile and they are initialised at every new session. If a role modification occurs during the user session, these SBs must be re-initialised to contain just the allowed methods according to the new profile taken by the user. The separation between data and methods avoids from loosing the existing non-persistent data when the user role modification occurs. In fact, only the session methods are affected by this re-initialisation.

8.1.3- The service level

The service level is a collection of system methods offered to the users. This level is implemented by SBs. They regroup generic methods that establish a bond between the users methods and the data. These methods are used to establish the first integrity data control and to dispatch the treatment location. Different kinds of system methods exist over this level. We distinguish two principal categories of them:

- primitives: this method contains all the transactions that manage the connection and the disconnection to the application server. These primitives constitute the first core of the identification layer.

- services: this method includes all the basic data management processes. It instantiates the functions of creation, modification, update and delete of data.

8.1.4- The access regulation level

This level represents the bridge between the different data locations and the different application servers. Since data are distributed between different locations, this level manages the amount of data in the data share level. This level is implemented by persistent SBs. They manage the data invocation trail in order to verify if the data is still used or not. If data is no longer used, they liberate the system memory by saving data on the Database. So this level reduces the volume of data in the data share level and consequently the system answering time.

8.1.5- The data share level

The data share level is used to represent our system components by a unique set of objects sharing the same informational space. This level is implemented by EBs due to their persistent data nature. Classes are represented as entities that manage their own integrity constraints. The various objects resulting from the design phase are validated by their corresponding meta-class. The set of these EBs manages the creation and the re-initialisation of SBs at the business level. These EBs store the non-persistent data resulting from the treatment interruption when a user changes his profile. The integrity of the relevant data is checked on this level. The matching between the data and their corresponding meta-class performs this validation.

8.1.6- The storage level

The last level represents the data storage space. In addition to the conceptual validation carried out on the data share level by EBs, a second validation is also carried out by the DDBMS to validate the data syntax integrity. We implement at this level functions that are used to detect the conceptual inconsistency and the major part of functionalities of integration of the various diagrams resulting from specifications.

8.2- M7TOOL functionalities

The M7TOOL is conceived and implemented in order to provide a collaborative design platform and a CASE tool which assists designers at the distribution design step by the generation of the different distribution sets and their corresponding system costs. For this purpose, the used middleware architecture presents an attractive architecture for this kind of collaborative application. It is typically based on 3-tiers level consisting of collaborative client portals at the first end, the computational resource, services or applications at the back end, and the server(s) in the middle [Kaur00]. In order to have a clear separation between different application logical layers (access control, design, data access…) we separate each service from others in a unique service pool. This separation increases the transparency between data and

functions. Service pools act as independent components and the middle-tier has the responsibility for establishing communication between them. We implement several basic functionalities in our framework. Each functionality is considered as an independent component. These services are:

- The process manager: it is considered as a security level with functions adapted to the collaborative design context. This process federates all activities around the framework.

- Collaborative synchronous design: we suppose that at this level, all tasks are done in synchronous or asynchronous mode. This process is strongly based on the first service (process manager). A designer creates his own schema or specification and adds new users by giving them different rights. If users are connected to the framework, they receive immediately a notification indicating the specification context, the owner and/or the inviter.

- Collaborative Asynchronous editing: users work in asynchronous mode without restrictions. The repository coupled with the data service guarantees the data integrity. Users are not obliged to inform their collaborative team about modifications done on schema. The repository manager compares the date of creation of schema components with the most recent date of connection of the designer and informs him about the modifications or the creations of new components by underlying them in different colours.

- Repository manager: due to some concurrent accesses, and to avoid dirty read/write operations, we implement an agent to supervise exported data to users. If it detects shared data between users who are not working collaboratively on same schema, it establishes a collaborative space between them. No delete or save operations will be allowed if users do not validate the common modifications together.

- Meta-Design process: this process is a graphical process. The modelling notations or languages can be easily configured and deployed in the tool. Generic shapes inspired from the most common object oriented methodologies (UML, OMT…) are proposed to the user. Users have the possibilities to add, delete or modify this set of notations. Only the Specification Administrator has enough rights to modify these notations. These modifications are not allowed after the creation of any entity by designers.

Every process is defined and implemented within a reuse approach. Interactions between this set of processes are matched by an inter-process data transfer service. It is strongly inspired from data process patterns [Cooper98]. This service manages all deployed process by regrouping all existing methods and attributes. The main goal of this service is wrapping sets of data needed by client. The client does not need to know from how many sources his data is coming.

8.3- The distributed design and implementation of M7TOOL

Small number of existing CASE tools provides mechanisms that enables designers to participate in the decision of distribution design. Indeed, the complex task of information system distribution has been always considered as a task integrated to the implementation level. Our proposed approach is based essentially on the cooperation and knowledge exchange between designers and developers to help them to obtain the optimal distribution design. The proposed algorithms exploit the aspects of integrity constraints that combines data and treatment to obtain a result which ensure an optimal distribution by increasing the system integrity and performance and decreasing the data flow inside the subsets that compose the information system. The experimentation of this approach is carried on the development of a CASE tool. The development of this CASE tool has two main purposes :

- The first is the experimentation of the set of proposed algorithms FKDO, ICDO and GIDO. This experimentation has to be done in an environment where the distribution is mandatory and where different choices can be done according to the project parameters. In our experimentation case, the complexity of the proposed data schema and the set of existing constraints make the data allocation hardest and difficult to be achieved without having a clear defined process that ensure the optimality of the generated results (see Figure.5).
- The second is the implementation of this set of algorithms in order to provide designers and developers with a design environment based on the presented distribution process. Such environment has to be adapted to the design of distributed information system. For this purpose, the experimentation has been carried out in a collaborative environment.

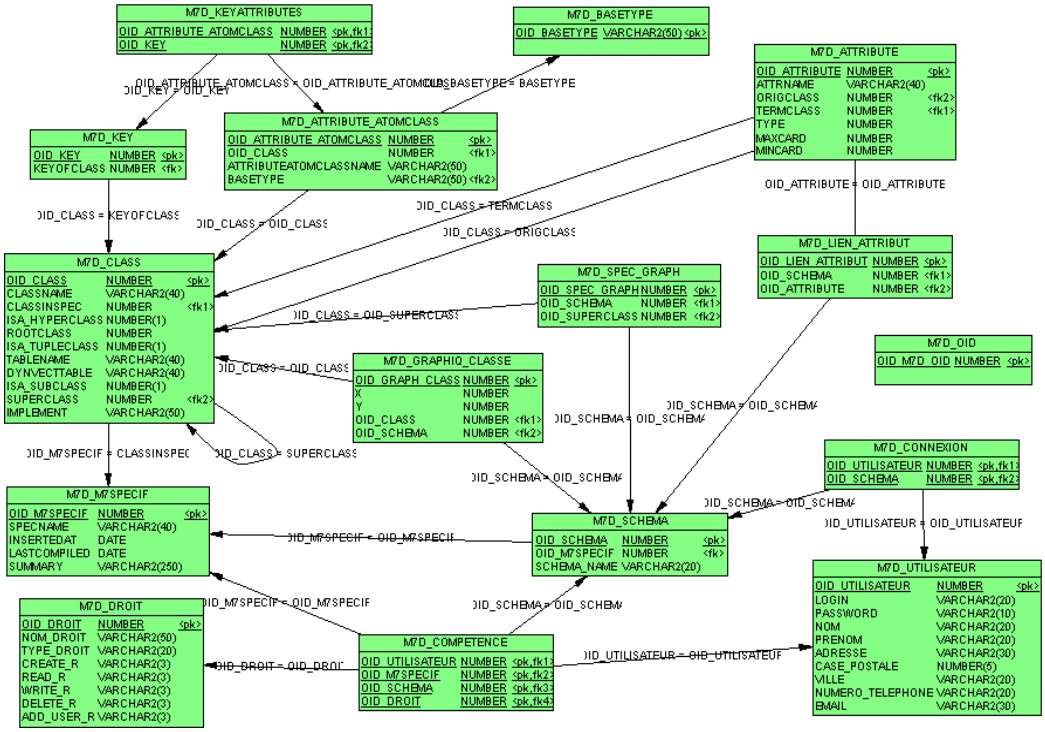


Figure 5. M7TOOL design schema [Snene03c]

The data storage level is composed from three sites. Every site is equipped with a distributed database management system. The database design schema regroups the different data tables and relations resulting form the design step.

The distribution design has been done according to GIDO algorithm. In fact, in this experimentation we have based our distribution on both foreign keys and integrity constraints. We have not considered all existing integrity constraints, but we have selected a subset of them to be used in the experimentation. In fact, as the purpose of the project is the development of a distributed and collaborative CASE tool, some integrity constraints are not directly related to the consistency of the database but to the collaborative design coherence. Indeed, we propose to designers to work cooperatively on their design schemas and to share their workspaces. As explained above, CASE tools still have many serious problems such as interoperability, reusability and limited support for teamwork. In this context, the proposed collaborative platform in M7TOOL seems to be a major advantage. The selection of the constraints used in the GIDO algorithm has been done according to their importance and their approximate frequencies of use. The result of the GIDO algorithm is composed of three subsystems.

The first subsystem is composed of the data graphical subsystem. It contains the data set that concerns the graphical design process. This subsystem contains the following tables (see Figure.6):

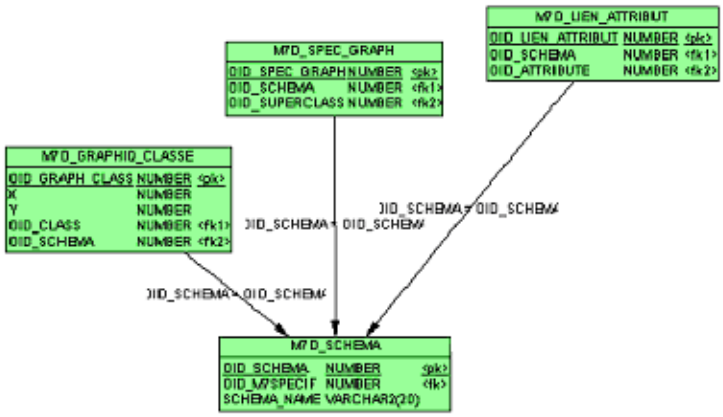


Figure 6. The first distributed set of M7TOOL

The second subsystem contains the coordination data and the security processes. It includes different data resulting from the collaborative and identification processes. This subsystem is the following (see Figure.7);

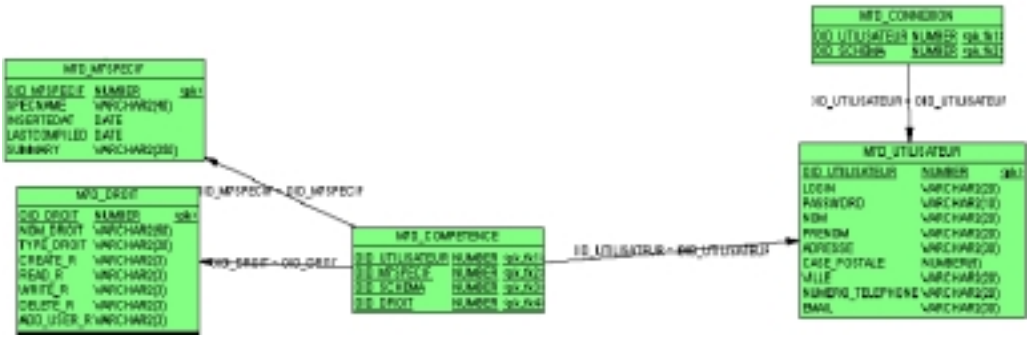


Figure 7. The second distribution set of M7TOOL

The last subsystem is composed of the different data treatment components. It is centralised around the class “M7D_Class”, which recovers the different concepts related to the distributed information system design. This subset is the following (see Figure.8).

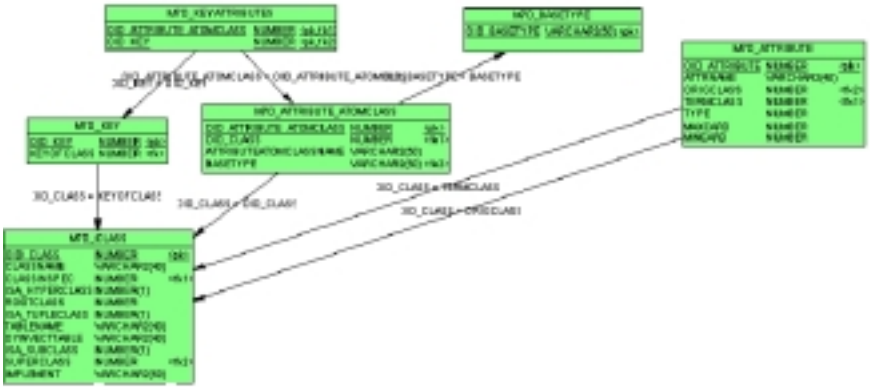


Figure 8. The third distribution set of M7TOOL

These distribution sets present the lowest distribution cost according to the different results obtained from GIDO algorithm. At the same time, these sets increase the system performance and availability. In fact, in the case of the failure of a subsystem, the others are able to continue working without interruption if no calls are required between them and the failed site. For example, if the user has been connected to the system and begins designing and the second site that represents the data of coordination and security process is out of order, the designer will not be interrupted, however, no more users can be connected until the second site is running again.

The obtained result underlines the different existing functional areas. This result cannot be obtained by a simple component based approach. In fact, it is hard to separate these results without a clear distribution mode. Indeed, as mentioned in chapter III, the component based approaches do not propose to designers a distribution methodology but different concepts that help designers to represent their different subsystems. In this case, designers carry out the distribution according to their experiences. Also, the obtained results do not express the system integrity constraints.

Indeed, the three approaches presented in chapter III which are UML, Catalysis and CADA do not take into account the design of integrity constraints and do not propose to designers any concept to represent under it these constraints. Finally, these approaches suggest that the distribution design has to be done according to the existing functional areas but without giving enough help to designers to detect these areas.

The second approach which is the data based approach, proposes a set of algorithms to establish the design schema allocation and fragmentation according to them. These algorithms do not take into account the system functional aspect and thus do not consider the existing integrity constraints. This approach is mainly based on queries optimisation by reducing the data flow between subsystems without considering the fact that the integrity constraints represent the major existing data flow. There is also no existing design tool which helps designers to generate the distribution solution according to the proposed algorithms. Finally, these algorithms do not give to designers the possibility to assign some tables to a predefined site.

8.4- Conclusion

The step of distribution of an information system is as explained in chapter III essentially based on two main strategies of allocation and fragmentation of data between the existing sites. The set of algorithms proposed in this thesis combined with the knowledge patterns, represents an innovative alternative to these existing strategies. The practised experimentation of this approach has demonstrated the efficiency of such distribution. In fact, the produced schemas of the CASE tool data model is distributed into three sets. These sets are composed essentially from data tables that cooperate for the achievement of the same functional methods. This experimentation has proved the efficiency of such approach that is based on integrity constraint s for distribution. The use of a complex data model which presents a huge number of distribution possibilities and so an optimal distribution more difficult to be achieved without a clear process, confirms the usability and the efficacy of the proposed method.

Chapter IX
Conclusions and future directions

In order to provide a more formal and clearer support for the distribution step of a database schema than is traditionally provided by existing methods, it is necessary for the different project partners to be able to define common goals and unified methods to obtain a more consistent distribution combination. Designers need to be sure that the implemented solution respect the aim of produced database schema, and more specifically that the integrity and the coherence of the distributed information system will always be guaranteed. Developers have to get a design schema that respects the different technical constraints and guarantees a rational system performance and a perfect security.

In this thesis we have described strategies for establishing a well-defined distribution of a database schema based on set of multiple criteria. This set is composed from functional, organisational and technical criteria. All these criteria are represented by the integrity constraints. In fact, we demonstrate that these constraints regroup different design and technical layers that are sufficient to reconcile the different partners taking part into the project.

In this chapter we briefly summarize the contributions of this thesis and discuss the extensions and other results that we are currently exploring.

9.1- Contributions

In this thesis we dealt with the different components of the distributed information system project. These components are surrounding different levels and are critical for the good achievement of the project. For every level we defined a formalised structure under the pattern form to establish a well defined and a structured way to convey the

different knowledge from a project partner to another. Moreover, we defined an overlap knowledge zone between designers and developers in which they share common information and the most important distribution decisions.

These different patterns are:

- DIS design knowledge pattern: it treats the knowledge surrounding the design step. This knowledge represents information obtained from the analysis step and expresses the main directions of the system;
- DIS implementation knowledge pattern: it contains different generic knowledge about the constraints imposed by the different technical platforms. It specifies also the different performance and QoS objectives.
- DIS overlap knowledge pattern: it is essentially composed of shared decisions that must be taken both by designers and developers. It contains information about the integrity constraints design and their way of implementation, data locations and the different sites relations.

On the basis of these different patterns, we extracted some critical levels of cooperation between the different project partners that have to be well defined and correctly treated in order to obtain the most suitable distribution combination. Thus, we suggested a framework composed of a succession of algorithms that propose a set of suitable distribution solutions. This framework calculates the cost of the different possible distribution combinations while taking into account the end-users, designers and developers constraints. The first algorithm is based on the foreign key distribution optimisation. Its aim is to decrease the number of the different site relations resulting from foreign key matching. The second algorithm treats the other different integrity constraints. Its result is a set of distribution solutions. The different generated solutions minimise the number of site connections and thus maximise their autonomies. They also reduce the data flow and so deliver better system performance and better answering time. Finally these algorithms structured in a global framework that takes in consideration the various constraints importance and according to it, the framework recalculates the different solutions while matching every constraint with a weighting value. This value evolves continuously according to the frequency of the constraint call.

Finally, to facilitate this treatment of the different integrity constraints, we have proposed an enhancement of the expression of these constraints. This improvement is composed of miscellaneous tunings. Indeed, the existing form of the integrity constraints is not enough informative in the case of a distributed information system design. The proposed specification is enhanced by new definition criteria useful to express the different execution root and the propagation of the risk across the different existing table. These criteria are:

- Risk class: it indicates the root class that by applying one of the system primitives may violate the integrity constraint due to its data modification.
- Range board: it represents the set of primitives likely to transgress the integrity constraint when applied to a specific risk class

- Cross-reference risk table: it regroups the different risk classes defined in the range board. It emphasizes the relation between the root class and the classes that will be concerned by the transgression of any corresponding primitive.

These three extensions are combined to help bridge the gap between the design stage of a distributed information system and its development. Rather than attempting to abolish this gap between the design and the development entirely, our basic philosophy has been to provide the project participants with a formal process that helps the different patterns to exchange their different knowledge for a better understanding of their respective constraints.

9.2- Future directions

This research has brought to light several areas that deserve further exploration. We outline three main lines of future research. The first direction to pursue is to enhance the different patterns of the frameworks in order to support the project participants in their respective tasks. The second direction is the development of protocol easily plugged in a distributed information system. This Protocol has to enable the evolution of the distribution solution at runtime. Finally, the third direction is the integration of the proposed distribution process in a CASE tool.

Pattern enhancement framework

In this thesis, we have defined three knowledge patterns. We have focused on the overlap knowledge pattern by defining different algorithms to allow the implementation of the database diagram distribution while respecting the various components of the pattern. One obvious area for a future work, then, would be the extension of the other different components of the design and implementation knowledge pattern. Such an extension of the patterns should require a significantly more formal approach to the definition of the different patterns components than the simple form defined in the current implementation. However, different techniques have been developed for information system design and distributed system design, which could be adapted to our different knowledge patterns.

In fact, different methodologies of information system design exist. Their major defect is that the distribution stage is often disregarded. We have to adapt these methodologies to the new needs of the distributed information system projects. The analysis phase must enable the emergence of the various end-users distribution requirements. The design phase is supposed develop the system models in this trend for a better needs expression and thus in a more understandable way by the developers. Also, different distributed system implementation processes are used. But they rarely describe the knowledge surrounding the processes steps and thus makes the adjustment of the design schema to the implementation requirements unfeasible by the designers and so delegates the modification operations to the developers which are not generally the most suited for the this realization. Another problem is the diversity of the existing

technical platforms and their heterogeneities. The extraction of generic technical characteristics will be subject to the review of this set. As different standardizations of these platforms are actually planned by different consortium, it is still hard to regroup in a clear and concise manner the various information necessary for the improvement of the implementation pattern.

Runtime distribution evolution

We have proposed an adjustment algorithm for the reassignment of the data tables between the diverse sites. Actually, this algorithm is used for simulation purpose and cannot be run at runtime. It would be useful to develop this algorithm under a plugged form in order to enable its integration in a distributed database management system (DDBMS). This extension will be done under the intelligent agent form. By this way, the weighting criteria will be adjusted according to the effective call of every constraints and the regulation of the distribution combination will satisfy the reel system requirements. The major problems that will be encountered to achieve this extension are the difficulty to integrate it in an existing DDBMS. In fact, the commercial form of the different DDBMS does not allow the code modification and generally does not take into account our adopted criteria such as the foreign keys.

CASE tool for DIS design

Many interesting directions have emerged form the definition of the different knowledge patterns and the implementation of the algorithms framework. One issue that we consider as the most useful one is the development of a CASE tool for the distributed information system based on our thesis results. Our actual implemented tool M7TOOL has proved enough performance by testing it with the FKDO algorithm [Snene03c]. We are currently developing the missing framework algorithms.

References

[Abrial96] J.R. Abriel, *The B-book*, Cambridge University Press, 1996.

[Akehurst99] D. H. Akehurst, A. G. Waters, *UML specification of distributed system environments*, Computing Laboratory, University of Kent at Canterbury, Technical Report 18-99, May 1999.

[Allen98] P. Allen, S. Frost, *Component-Based Development for Enterprise Systems Applying the SELECT Perspective*, Cambridge University Press, 1998.

[Andrade96] J.M. Andrade, M.T. Carges, T.J. Dwyer, S.D. Felts, *The TUXEDO system, software for constructing and managing distributed system applications*, Addison Wesley, 1996.

[Apers88] P.M.G. Apers, *Data allocation in distributed database systems*, ACM Transactions on Database Systems, September 1988.

[Backus78] J. Backus, *Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs*, Communications of the ACM, Volume 21, Number 8, August 1978.

[Barbara82] D. Barbara, H. Garcia-Molina, *How expensive is data replication? An example*, in Proc. 2nd Distributed Computing Systems, Feb. 1982, pp. 263-268.

[Barbara92] D. Barbara, H. Garcia-Molina, *The Demarcation Protocol: A Technique for Maintaining Arithmetic Constraints in Distributed Database Systems,* In Extending Database Technology Conference, LNCS 580, pages 373--397, Vienna, March, 1992.

[Bernstein80] P. A. Bernstein, B. T. Blaustein, E. M. Clarke, *Fast Maintenance of Semantic Integrity Assertions Using Redundant Aggregate Data*, In Proceedings of the Sixth Conference on Very Large Data Bases, pages 126--136, 1980.

[Blakeley89] J. A. Blakeley, N. Coburn, P. Larson, *Updating Derived Relations: Detecting Irrelevant and Autonomously Computable Updates*, ACM Transactions on Database Systems, 14(3):369--400, 1989.

[Bodart83] F.Bodart, Y Pigneur, *Conception assistée des applications informatiques: étude d'opportunité et analyse conceptuelle*, Masson, Paris 1983.

[Boertien04] N. Boertien, M. W.A. Steen, H. Jonkers, *Evaluation of Component-Based Development Methods, Information Modelling Methods and Methodologies*, IDEA Group, 2004.

[Bolognesi92] T. Bolognesi, *Catalogue of LOTOS correctness preserving transformations*, Lotosphere deliverable, Lotosphere consortium, Netherlands 1992.
[Boogards90] K. Boogards, *A methodology for the architectural design of open distributed systems*, PhD Thesis, University of Twente, Netherlands 1990.

[Booth81] G.M. Booth, *The distributed system environment: some practical approaches*, New York, Mc Gran Hill, Inc, 1981.

[Bubenko94] J. Bubenko, *Enterprise modeling*, Ingénierie des systèmes d'information, 2(6), 1994.

[Busse99] S. Busse, R-D. Kutsche, U. Leser, H. Weber, *Federated Information Systems : Concepts, Terminology and Architectures*, Forschungsberichte des Fachbereichs Informatik Nr. 99-9, Technische Universität Berlin, 1999.

[Casal02] J.A. Casal, J.A. Garda, R.G Vazquez, S.R. Yarrez, *A practical experience in analysis and design of distributed information systems*, I+D Computation, Vol.1, No.1, July 2002.

[CCITT88] CCITT, *Method for characterization of telecommunication services supported by an ISDN and network capabilities of an ISDN*, CCITT Blue book, CCITT, Geneva, Switzerland 1988.

[Cer87] S. Ceri, B. Pernici, G. Wiederhold, *Distributed databse design methodologies*, IEEE Proceedings, 1987.

[Ceri83a] S. Ceri, S.B. Navathe, *A comprehensive approach to fragmentation and allocation of data in distributed databases*, Proceedings of the IEEE COMPCOM conference, 1983.

[Ceri83b] S. Ceri, S.B. Navathe, G. Wiederhold, *Distribution design of logical database schemas*, IEEE Transactions on Software Engineering, 1983.

[Cheesman00] J. Cheesman, J. Daniels, *UML Components: A Simple Process for Specifying Component-Based Software*, Addison-Wesley, Upper Saddle River/NJ, 2000.

[Chong00] S. Chong, K. Liu, *A Semantic Approach for Modeling and Designing Agent-Based Information Systems based on Roles and Norms,* Agent-Oriented Information Systems (AOIS-2000).

[Codd79] E.F. Codd, *Extending the relational database model to capture more meaning*, ACM TODS, 4, 1979.

[Colin02] A. Colin , D. Muthig, *Component-Based Product-Line Engineering with the UML*, international conference on software reuse, USA, 2002.

[Cooper98] J.W. Cooper, *The design patterns java companion*, Addison Wesley Design patterns series, 1998.

[D'Souza98] D. F. D'Souza, A. C. Wills, *Objects, Components, and Frameworks with UML: The Catalysis Approach* ,Addison-Wesley Object, Technology Series, 1998.

[Dale98] J. Dale, *Mobile agent architecture to support distributed resource information management*, PhD thesis, University of South Hampton, June 1998.

[Date83] C.J Date, *An introduction to database systems*, Volume 2, Addison-Wesley, 1983.

[Davenport81] R.A. Davenport, *Design of distributed data base systems,* Computer Journal,24, 1981.

[Davenport92] Davenport S., CASE tools- who needs them ?, Software Management, April 1992.

[Debenham02] Debenham and B. Henderson-Sellers, *Full Lifecycle Methodologies for Agent-Oriented Systems - The Extended OPEN Process Framework*, Agent-Oriented Information Systems (AOIS-2002).

[Denker99] G. Denker, H.D. Ehrich, *Specifying distributed information systems: fundamentals of an object oriented approach using distributed temporal logic*, Technical university of Braunschweig, 1999.

[DePaoli92] F. DePaoli, F. Tisato, *A model for real time cooperation*, Proceedings of the second European conference on computer supported cooperative work, Amsterdam, 1991.

[DeWeger99] M. K. De Weger, C. A. Vissers, *Issues in design methodologies for distributed information systems*, Center for Telematics and Information Technology, University of Twente, The Netherlands,1999.

[Dodd94] Dodd J., Developing Information Systems from components: the role of CASE, Business Objects: Software Solution, John Wiley & Sons, 1994.

[Dowdy82] L.W Dowdy, D.V. Foster, *Comparative models of the file assignment problem*, ACM Computer Survey, 14, 1982.

[Dupuy00] S. Dupuy, *Couplage des notations semi-formelles et formelles pour la spécification des systèmes d'information*, Phd thesis, Joseph Fourier University ,Grenoble, 2000.

[Ekberg99] A. Ekberg, *Enabling technologies for web centric applications*, PhD thesis, Lund institute of technology, November 1999.

[Espindola 04] A. P. Espindola, K. Becker, A.F. Zorzo, *An extension to UML Components to consider distribution issues in early phases of application development*, 37th Hawii international conference on system sciences, IEEE, USA, 2004.

[Ferreira91] P. Ferreira, M. Van sinderen, C. Vissers, *Advanced design concepts for distributed systems development*, Proceedings of the fourth workshop on future trends of distributed computing systems, IEEE Computer society Press, Los almitos, USA 1993.

[Fraser94] M. D. Fraser, K. Kumar, and V. K. Vaishnavi, *Strategies for incorporating formal specifications in software development*, Communications of the ACM, 37(10):74-86, October 1994.

[Frost94] Frost S., Developing Client-Server Systems Using OO Technology, Business Objects: Software Solution, John Wiley & Sons, 1994.

[Frost94] Frost S., Developing Client-Server Systems Using OO Technology, Business Objects: Software Solution, John Wiley & Sons, 1994.

[Grefen96] P. Grefen, J. Widom, *Integrity Constraint Checking in Federated Databases*, 1st IFCIS International Conference on Cooperative Information Systems, pages 38-47, 1996.

[Harmon93] Harmon P., Object Oriented Development Tools, Object Oriented Strategies, Cutter Information Corp, 1993.

[Hirschfeld96] R. Hirschfeld, *Three tiers distributed architecture*, Proceedings PloP 96, Allerton Park, IL, 1996.

[Hoare73] Hoare, C A. R. and N. Wirth, *An axiomatic definition of the programming language PASCAL*, Acta Informatica 2, 1973.

[Hoare74] Hoare, C A. R. and P. Lauer, *Consistent and complementary formal theories of the semantics of programming languages*, Acta Informatica 3, 1974.

[Hoffer75] J.A. Hoffer, *A clustering approach to the generation of subfiles for the design of a computer data base*, PhD dissertation, Department of operations research, Cornell university, 1975.

[Hui00] K. Hui, *Knowledge Fusion and Constraint Solving in a Distributed Environment*, PhD Thesis, University of Aberdeen, Kings College, Aberdeen, 2000.

[Jackson83] M.A. Jackson, *System development*, Prentice Hall, 1983.

[Jacobson92] I. Jacobson, M. Christerson, G. Overgaard, *Object-Oriented software Engineering – A use case driven approach*, Addison-Wesley, 1992.

[Jonkers00] H. Jonkers, M. Steen, N. Boertien,R. Slagter, *Component-based rapid service development*, Telematica Instituut, May 2000, Germany.

[Josifovski99] V. Josifovski, Design, *Implementation and Evaluation of a distributed mediator system for data integration*, Department of Computer and Information Science, PhD Thesis, Linköpings universitet, Sweden, 1999.

[Karlapalem96] K. Karlapalem, S.B. Navathe, M. Ammar, *Optimal redesign policies to support dynamic processing of applications on a distributed relational database system*, Information Systems, Vol21, No4, 1996.

[Kassem00] N. Kassem, *Designing Enterprise Applications with the Java 2 Platform*, Blueprints document, Enterprise Edition, Sun Microsystems, 2000.

[Kazerouni97] L. Kazerouni, K. Karlapalem, *Stepwise redesign of distributed relational databases*, Technical report hkust-CS97-12, the Hong Kong university of science and technology, Hong Kong 1997.

[Korth86] H. Korth, A. Silberschatz, *Database system concepts*, McGraw Hill, 1986.

[Kaur00] S. Kaur, V. Mann, V. Matossian, R. Muralidhar & M. Parashar, *The Applied Software Systems Laboratory (TASSL)*, 2000.

[Kruchten99] P. Kruchten, *The Rational Unified Process: An Introduction*, Addison-Wesley edition, 1999.

[Leitzelman98] M. Leitzelman, H. Dou, *Essai de typologie des Systèmes d'Information*,. International Journal of Information Sciences for Decision Making N°2, April 1998.

[Leonard92] M. Leonard, *Database design theory*, Macmilan Publication Ltd, Macmilan Computer science Series, 1992.

[Leser00] U. Leser, *Query Planning in Mediator Based Information Systems*, Informatik der Technischen Universität Berlin, Juin 2000.

[Levin75] K.D. Levin, H.L. Morgan, *Optimizing distributed databases: a framework for research*, Proceedings of national computer conference, 1975.

[Meyer88] Meyer B., Object Oriented Software Construction, Prentice-Hall, 1988.

[Mkrygiannis00] N. Makrygiannis, *Dispersed Information System Structures – Towards a balance between Integration and Independence*, Department of Informatics, Göteborg University,2000, Sweden.

[Murugesan02] S. Murugesan, *Agent-Based Information Systems*, Conference on Intelligent Information Processing, World Computin Congress 2002, Beijing, China, August 2002.

[Navathe84] S. Navathe, S. Ceri, G. Wiederhold, J. Dou, *Vertical partitioning of algorithms for database design*, ACM Transactions, Database system, 1984.

[Özsu85] M. T. Özsu, *Performance comparison of distributed vs centralised locking algorithms in distributed database systems*, Proceedings of distributed computing systems conference, May 1985.

[ÖZSU91] M. T. Özsu, P. Valduriez, *Distributed database systems: where are we now?*, Computer, August 1991.

[Özsu94] M.T. Özsu, P. Valduriez, *Distributed Data Management: Unsolved Problems and New Issues*, Readings in Distributed Computing Systems, IEEE Computer Society Press, pp 512-544, 1994.

[Özsu97] M. T. Özsu, P. Valduriez, *Distributed and parallel database systems*, Handbook of computer science and engineering, Boca Raton, CRC Press, 1997.

[Özsu99] M.T. Özsu, P. Valduriez, *Principles of distributed database systems*, Prentice Hall Edt, New Jersey, 1999.

[Pawlan00] M. Pawlan, Writing Enterprise applications with Java 2 SDK Enterprise Edition, Sun micro systems, Septembre 2000.

[Place 90] P. R. H. Place, W. Wood, M. Tudball, *Survey of Formal Specification Techniques for Reactive Systems*, Software Engineering Institute, CMU/SEI-90-TR-5, May 1990.

[Rahul01] S. Rahul, *J2EE connector architecture specification*, Sun micro systems, April 2001.

[Rivera-Vega90] P.I Rivera-Vega, R. Karlapalem, M. Ra, *A mixed fragmentation approach for inintial distributed database design*, Proceedings of International conference on data engineering, IEEE, 1990.

[Sacca85] D. Sacca, G. Wiederhold, *Database partitioning in a cluster of processors*, ACM Transactions, Database system, 1985.

[Schmidt00] Schmidt D., *The design and performance of a pluggable protocols framework for real time distributed object-computing middleware*, IFIP/ACM, Middleware Conference, New York, 2000.

[Sierhuis96] Sierhuis, M., Towards a Framework for collaborative modelling and simulation, Workshop on Strategies for collaborative Modelling and Simulation, June 1996.

[Snene03a] M. Snene, J. Pardellas, *Distributed Information System Design Based on Foreign Keys Optimisation: FKDO Algorithm*, 9th International Object Oriented Information System, Springer Ed,Geneva 2003, Switzerland.

[Snene03b] M. Snene, M. Leonard, *Distributed information systems project: Knowledge Pattern*, ITSIM03, Springer Ed, Kuala Lumpur, Malysia 2003.

[Snene03c] M.Snene, M. Leonard, *A collaborative CASE tool for distributed information system design: M7TOOL*, AICCSA03, IEEE Press, Tunis, Tunisia 2003.

[Snene04] M. Snene, J. Pardellas, M. Leonard, *Information system architecture: where we are?*, ICTTA04, IEEE Press, Damascus, Syria 2004.

[Sol92] H.G. Sol, R.L. Crosslin, *Dynamic modelling of information systems* II, North Holland, Amsterdam,1992.

[Sommerville92] I. Sommerville, Software Engineering, Addison Wesley, 4th edition, 1992.

[Spivey92] J.M. Spivey, *The Z notation*, Prentice Hall International, 1992.

[Tanenbaum02] A.S. Tanenbaum, M. Van Steen, *Distributed Systems–Concepts and Paradigms*, Prentice-Hall, 2002.

[Thiran99] P. Thiran, J. Hainaut., *Interoperability of Legacy Databases – A Combined Top-Down and Bottom-Up Approach*, Proceedings of the 8th Doctoral Consortium at the CAiSE*01, volume B 01-04,1999.

[Unger97] T. Unger, M. Höding, *A Practitioner's View to the Integration of Virtual Enterprise Database Systems by Federation Techniques*, Proceedings of the First East-European Symposium on Advances in Databases and Information Systems, Springer Ed, St Petersburg, 1997.

[Varadarajan89] R. Varadarajan, P.I. Rivera-Vega, S.B. Navathe, *Data redistribution scheduling in fully connected networks*, Proceedings of 27th Annual Alberton conference on communication, Control and Computing, 1989.

[Vissers91] C.A. Vissers, M. van Sinderen, E. Brinksma, *Specification styles in distributed systems design and verification*, Theoretical Computer science, 89, 1991.

[Waters99] A.G. Waters, D.H. Akehurst, *UML specification of distributed system environments*, Technical report, University of Kent at Canterbury, 1999.

[Weher02] H. Weher, Integrating Heterogeneous Data Sources into Federated Information Systems, University of Applied Studies and Research, Proceedings of Net.Object days, Germany, 2002.

[Wilson86] B. Wilson, S.B. Navathe, *An analytical framework for the redesign of distributed databases*, Proceedings of the 6th advanced database symposium, Tokyo, Japan 1986.

[Winters03] D. Winters, *Component Development with Catalysis*, Component Architects, International council of system engineering journal, USA, 2003.

[Wu98] J. Wu, *Distributed System Design* ,CRC Press Publishing Company, Boca Raton, FL, 1998.

[Yilmaz02] L. Yilmaz, *Specifying and verifying collaborative behavior in component-based systems*, PhD Thesis, Graduate School of Virginia Polytechnic Institute and State University, Unites States, 2002.