

AN OPEN ARCHITECTURE FOR DATA ENVIRONMENTS BASED ON  
CONTEXT INTERCHANGE

by

MARIA ELENA NEIRA

Ingeniero de Caminos, Canales y Puertos (1988)  
Master en Comunidades Europeas (1989)  
Universidad Politécnica de Madrid

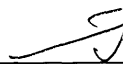
Submitted to the Sloan School of Management and  
the School of Architecture and Planning  
in Partial Fulfillment of the Requirements of the Dual Degree of

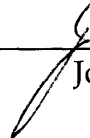
Master of Science in Management  
and  
Master in City Planning


at the  
Massachusetts Institute of Technology  
February 1995

© 1995 Massachusetts Institute of Technology. All Rights Reserved

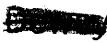
Signature of Author \_\_\_\_\_  
MIT Sloan School of Management  
October 3, 1994

Certified by  \_\_\_\_\_  
Stuart E. Madnick  
John Norris Maguire Professor of Information Technology  
Thesis Supervisor

Certified by  \_\_\_\_\_  
Joseph Ferreira, Professor of Urban Studies and Planning  
Director of Computer Resources Laboratory  
Thesis Reader

Accepted by  \_\_\_\_\_  
Jeffrey A. Barks  
Associate Dean, Master's and Bachelor's Programs

Accepted by \_\_\_\_\_  
Langley C. Keyes, Ford Professor of Urban Studies and Planning  
Chairman, MCP Committee

 ROTCH  
MASSACHUSETTS INSTITUTE  
OF TECHNOLOGY

APR 03 1995

LIBRARIES

# AN OPEN ARCHITECTURE FOR DATA ENVIRONMENTS BASED ON CONTEXT INTERCHANGE

by

MARIA ELENA NEIRA

Submitted to the Sloan School of Management and  
the School of Architecture and Planning on October 3, 1994  
in Partial Fulfillment of the Requirements of the Dual Degree of  
Master of Science in Management and Master in City Planning

## ABSTRACT

The management of large scale integrated computing environments presents several problems resulting from hardware and software incompatibilities, from systems architecture disparities, and from inconsistencies in the data itself. The situation in the data management arena is the most problematical because most of the current systems and their application programs do not explicitly declare the meaning, semantics and organization of their data. This important information the "data environment" as well as its manipulation is the central concern of Context Interchange Theory, a part of the ARPA-sponsored Intelligent Integration of Information (I<sup>3</sup>). The project as a whole develops knowledge-based techniques and tools to promote the sharing of data among networked systems; its client/server architecture is based on open system standards. After tracing the evolution of the data environment concept as applied to computer systems, we present a novel approach to their representation by means of a frame-based Context Definition Language (CDL). This language allows for interoperability among a network of systems by restructuring their data environments into inter-related Domains consisting of EntityObjects, Properties, MetaProperties and Links. Several examples are presented in CDL to illustrate context acquisition, and the capture of data environments, i. e., context knowledge. Finally, using the newly defined CDL, we build a C++ context prototype.

**Keywords:** Context Interchange, Data Management, Knowledge Representation Languages, Open Systems.

Thesis Supervisor: Stuart E. Madnick

Title: John Norris Maguire Professor of Information Technology

A mis padres

## ACKNOWLEDGMENTS

An endeavor like a dual master program cannot be undertaken without the support of many people. It is not possible to list here all those that one way or another contributed to the completion of this study. I would like to mention, however, those that for their position or because of the circumstances made a larger contribution towards the completion of this thesis.

At MIT, I must thank Professor Stuart E. Madnick, my advisor, who judiciously as well as steadily guided me through the steps of the program, including this thesis. During these years, he has been a great professor as well as a sensitive and intuitive mentor. I can never adequately express my infinite gratitude to this one gentleman.

At the Center for Earth and Planetary Studies, Washington D.C., I must thank Dr. Thomas Watters and Mr. Michael Tuttle who introduced me to the world of new Information Technologies. To both of them my gratitude for their patience.

At Universidad Politécnica de Madrid, I must thank Professor Carmen Gavira and Professor José Calavera who encouraged and helped me to persuade a research-oriented career. They deserve my most affectionate admiration.

A very special remembrance goes to Jesús Sarmiento, my grandfather, who left this world several years ago, and to whom I miss very much and keep very alive in my memory.

Finally, I must endlessly thank my parents, Elena and Jaime, and my brother and sisters, Santiago, Flor and Belén, who, far from here but very close to my heart, have been, are, and will always be, the major reason for the efforts that I have undertaken outside my home country. To them, my deepest gratitude for their tireless understanding, support and patience.

## TABLE OF CONTENTS

	PAGE
ABSTRACT .....	2
ACKNOWLEDGMENTS .....	4
TABLE OF CONTENTS .....	5
LIST OF ACRONYMS .....	9
LIST OF FIGURES .....	10
LIST OF TABLES .....	11
CHAPTER 1: INTRODUCTION .....	12
1.1 Interoperable computer networks .....	12
1.1.1 Physical and logical connectivity .....	13
1.1.2 Integration strategies .....	16
1.2 The context interchange framework.....	20
1.3 Outline of thesis.....	29
CHAPTER 2: DATA ENVIRONMENTS .....	31
2.1 On data structures and knowledge representation.....	32
2.1.1 Data types.....	35
2.1.2 Record-based models.....	37
2.1.3 Semantic models.....	39
2.1.4 Object oriented models.....	40
2.1.5 Integrated databases .....	41
2.1.6 Data repositories, data warehouses and metadata .....	42
2.1.7 Re-engineering data environments .....	46
2.2 On data environments and interoperability.....	47

2.2.1 Examining a data environments .....	47
2.2.2 Closed systems, open systems and their differences .....	49
2.3 Data conflicts over the network .....	50
2.3.1 1-Dimensional conflicts .....	51
2.3.1.1 Semantic conflicts .....	51
2.3.1.2 Domain-dependent meaning.....	53
2.3.2 N-Dimensional conflicts .....	54
2.3.2.1 Inter-domain conflicts .....	55
2.3.2.2 Domain-specific dimensions .....	56
2.3.2.3 Complex conflicts .....	57
2.3.2.4 Order conflicts .....	59
2.3.3 Naming conflicts .....	59
2.4 The architecture of open data environments .....	60
CHAPTER 3: CONTEXT DEFINITION .....	63
3.1 Why a new model and a new representation language for context? ...	70
3.1.1 Components of the architecture .....	71
3.1.2 The context application logic.....	72
3.2 A formal view of context interchange .....	73
3.2.1 Assumptions .....	73
3.2.2 Relations .....	78
3.2.3 Transformations in context .....	80
3.2.4 The ontology as facilitator of data exchanges .....	81
3.3 Context representation: CDL.....	82
3.3.1 CDL characteristics .....	83
3.3.2 Domains in CDL.....	84
3.3.2.1 Source context domain definition.....	85
3.3.2.2 Receiver context and domain-restricted queries .....	88
3.3.3 EntityObjects, Properties and their semantics .....	90

3.3.3.1 Identification .....	90
3.3.3.2 Ontology Links .....	93
3.3.3.3 Representations .....	99
3.3.3.4 Constraints .....	101
3.3.3.5 Missing value .....	101
3.3.3.6 Transformations .....	101
CHAPTER 4: CONTEXT ACQUISITION USING CDL.....	103
4.1 The network directory locker .....	104
4.1.1 Domain .....	104
4.1.2 Subdomains.....	106
4.1.3 Context interchange view .....	106
4.2 The Alumni/Gift Database .....	107
4.2.1 Domain .....	107
4.2.2 Subdomains.....	110
4.2.3 Context interchange view .....	118
4.3 The Topologically Integrated Geographic Encoding and Referencing files .....	124
4.3.1 Domain .....	124
4.3.2 Subdomains.....	124
4.3.3 Context interchange view .....	125
CHAPTER 5: PROTOTYPE.....	126
5.1 Scope of the prototype.....	126
5.2 Design issues.....	127
CHAPTER 6: CONCLUSION .....	128
6.1 What is missing today? .....	128
6.2 Context Interchange.....	130
6.3 Context acquisition: definition and views.....	134
6.4 Network interoperability through context .....	135
REFERENCES .....	139

APPENDIX NUM. 1: SOURCE CODE ..... 145



## LIST OF ACRONYMS

ADT	.....	Abstract Data Types
ARPA	.....	Advance Research Projects Agency
CAD/CAM	...	Computer Aided Design/Computer Aided Manufacturing
CASE	.....	Computer Aided Software Engineering
CDL	.....	Context Definition Language
CDT	.....	Complex Data Types
CIM	.....	Computer Integrated Manufacturing
CISL	.....	Composite Information Systems Laboratory
CIS/TK	.....	Composite Information Systems/Tool Kit
DBMS	.....	Database management System
DCE	.....	Distributed Computing Environment
DDL	.....	Data Definition Language
DML	.....	Data Manipulation Language
E-R	.....	Entity-Relationship
GIS	.....	Geographical Information System
IRDS	.....	Information Resource Dictionary System
IT	.....	Information Technology
IS	.....	Information Systems
I <sup>3</sup>	.....	Intelligent Integration of Information
MIA	.....	Multivendor Integration Architecture
MIT	.....	Massachusetts Institute of Technology
O-O	.....	Object-Oriented
OSI	.....	Open Systems Interconnection
OSF	.....	Open Software Foundation
		Topologicaly Integrated Geographic Encoding and
TIGER	.....	Referencing
USGS	.....	United States Geological Survey

## LIST OF FIGURES

	PAGE
1. Context Interchange in a Simple Source-Receiver System.....	25
2. Data Representation in Computer Programs.....	36
3. Example of N-Dimensional Conflicts.....	57
4. Client-Server Context Interchange Architecture .....	69
5. Graphical Nomenclature of Context Model .....	76
6. Simplified View of Context Interchange Model .....	79
7. Application of Context Interchange Model.....	81
8. Network Domain Description in Natural Language .....	85
9. Network Domain Description in CDL .....	86
10. MIT Alumni/Gift Database Description .....	88
11. Receiver Context Querying in Context_Domain.....	89
12. Context Definition Language Layered Structure .....	91
13. Interlocking of Hierarchies in the Shared Ontology .....	93
14. Is_Kind_Of Link .....	95
15. Part_Of Link.....	96
16. Relates_To Link .....	97
17. Example of Multi-Cardinality .....	100
18. Examples of Network Directory Records .....	105
19. Network Directory Source Domain Description .....	107
20. The MIT Alumni/Gift Database Schema.....	108
21. List of Fields Selected for Analysis.....	110
22. MIT Affiliation Code Table.....	112
23. Database Source Context.....	117
24. Local and Global Links.....	121
25. Subset of the Ontology .....	123

## LIST OF TABLES

	PAGE
1. Data Conflict Categories .....	22
2. 1-Dimensional Format Conflicts .....	52
3. Domain-Dependent Conflicts.....	54
4. List of Name Sub-components by Country .....	55
5. List of Dimensions and their Mismatches .....	56
6. Conflict Table .....	64

# CHAPTER 1

## INTRODUCTION

### **1.1 Interoperable computer networks**

Over the past few years, breakthroughs in networks and telecommunications technology have provided increasing connectivity among a wide variety of devices that manage the input, output, storage and processing of data. Systems connectivity has increased not only from improvements in these technologies but also from dramatic advances in hardware and software. These improvements in technology have created new systems architecture and new ways of computing, i. e., massive parallel processing, distributed processing, and cooperative and client-server computing.

They give organizations the opportunity to take advantage of more economical, more sophisticated and more reliable IT infrastructures, which have become capable of supporting a wider range of computer-based activities. In addition, the fact that these technological developments have carried with them drastic price reductions in computer-mediated communications and computing power, gives them strategic advantage and provides them with new methods to redesign and improve their IT-based activities, and expand them internally as well as externally .

Nevertheless, especially in data-intensive computing environments, all these benefits do not come without cost. There exist serious problems to be solved before we can achieve and take advantage of full IT systems connectivity. These difficulties, identified by Madnick, Siegel and Wang [Madnick & others 90] as logical and physical connectivity<sup>1</sup> issues result from the idiosyncrasies of the component systems in the IT infrastructure: each one has its own domain, its own architecture and, its own implicit organization. Moreover, the structure and meaning of the data residing within each system has its own unique data environment.

### **1.1.1 Physical and logical connectivity**

In general, connectivity problems, also known by the names of interoperability and systems integration problems, fall into four categories [Collet & others 91, Kent 79b, Lenat & others 90, Madnick & others 90, Malone & others 87, Navathe 92, Shen & others 91, Takagi 92, Thompson 89]:

- **Hardware incompatibilities**

They result from incompatible architectures and from the lack of standard interfaces among the many hardware components.

- **System software incompatibilities**

They include, among others, differences in transaction processing and concurrency control mechanisms, differences in interactive processing environments, and in real-time processing requirements.

---

<sup>1</sup>For a complete definition of the terms logical connectivity and physical connectivity the reader is referred to [Wang & Madnick 89].

- **Data incompatibilities**

They are chiefly those found within the data structures and relationships among those structures, and in the way both of them are implemented. Particularly important are the differences in atomic data types across systems; but the major problems arise when these atomic data types are incorporated into higher order types (data structures, tables, objects, entities, etc.). The solution to this problems of types, aggregates and relationships among them are radically different in each environment. In the Information Systems (IS) arena, these incompatibilities are manifested in the form of data model differences. That is, data definition language (DDL), data manipulation language (DML), data representation, database programming and/or query language vary from system to system. Yet even in the case of model and types homogeneity across environments, incompatibilities due to data conflicts<sup>2</sup> are very likely to occur.

- **Communication incompatibilities**

Including differences in network protocols, and in transmission modes, i. e., synchronous versus asynchronous , etc.

To overcome these difficulties is the new challenge of the IT systems integration paradigm. The task is especially difficult in those networks where accurate and timely data exchange among heterogeneous environments<sup>3</sup> is an inherent part of the systems specifications. The following examples are drawn from typical IT

---

<sup>2</sup> See Section 2.3 for definitions and classification of data conflicts.

<sup>3</sup> Throughout this thesis we will be using the term environment, particularly data environment meaning the set of internal and external characteristics which affect the overall properties of the system's resident data.

systems, where data heterogeneity is a major minefield in the route to integration:

- Multi-agent design systems such as CIM and CAD/CAM managing dynamic complex objects with substantial changes throughout their life cycles, and with time-stamped feedback flows among objects.
- Production-oriented scheduling applications, particularly those involving the integration of operations to reduce costs and increase effectiveness in throughput, reallocation, dispatching, inventory control, and other tasks related to plant operations control.
- Intelligent planning and flow control supporting distributed transportation. An example of this would be the 1992 Integrated Feasibility Demonstration of the DRPI, where a planning system developed and detailed a military forces employment and deployment plan as well as a simulator to analyze this plan with respect to transportation feasibility.
- Computer-based decision support systems that need to interchange timely and accurate information. An example would be government-related information systems, which contain large amounts of critical transportation management information, socio-economic data, population statistics, etc. In emergency situations it would be especially important for these systems to ensure the proper data flows and feedback. Unfortunately, most of information systems remain isolated, contain duplicated and overlapping information, and the proper integration strategies are not yet in place.

In addition to these technically-oriented issues, strategic and organizational issues also play an important role in systems integration. The increase in IT systems connectivity has impacted and changed the information flows, functions, coordination among functions, boundaries and structures of the organization itself as well as its culture; it even influences the IT systems integration paradigm --when the appropriate feedback mechanisms are in place. Issues such as computer security, data ownership, users' privacy, inter-organizational linkages etc. are, at different levels, key issues which must be taken into consideration in order to develop a sound, safe and successful integration strategy.

### **1.1.2 Integration strategies**

Nowadays, a typical information technology infrastructure consists of a collection of different systems residing in a variety of different, multi-vendor, mutually incompatible software and hardware platforms, and running a mixture of business, scientific, and engineering applications, which have incompatible data environments.

The entire system (enterprise) needs to be linked into a coherent whole, by tying together all their components, irrespective of the vendor platform into which the systems are running, of the system role, of the data model, and of the data environment. Moreover, the enterprise may also want to extend its business by linking enterprise-wide systems with the business suppliers, distributors and customers' environments. Our integration strategy should be able to respond to all these demands.



In order to accomplish this, we need to open the component systems, i. e., to develop open systems architecture as the basis of our systems integration scheme.

Everyone seems to recognize the potential benefits of system openness and of a systems interoperability strategy. Numerous initiatives in the private, public and international arenas --Open Software Foundation (OSF), concurrent engineering, Open Systems Interconnection (OSI), system integrators, ARPA-sponsored knowledge sharing effort, MIA's architecture, ARPA-sponsored Intelligent Integration of Information effort, re-engineering, etc.-- have been undertaken recently to achieve these ends. These initiatives, at different levels and with different scopes, have addressed the systems interoperability problem, and have tried to solve it by creating new technologies and tools with such names as protocols, common system interfaces, wrappers, standards, mediators.

It may seem that the obvious and logical solution to interconnect all these systems, move applications and move data is to open their architectures by adopting inter --and intra-- systems standards-based interfaces. But the open standards solution is not yet in place, and, as we pointed before, this is even more truth in the data management arena.

In the past, many of these systems integration problems were studied in a smaller scale, and were solved by analyzing each of the components, and by imposing standards on their hardware and software. Regarding data integration, previous studies [ACM 90, Kent 79b, Shen & others 91] have shown that heterogeneity, semantic discrepancies, and conflicts of meaning are the norm when comparing data from different sources. The most common way to solve the

problem has been to create an integrated schema --very often called enterprise model--, which can be either composite or federated, to glue the components. Implicit to this approach is the fact that integrators have been coping with component heterogeneity:

- **Hiding differences**

Hiding (hard-coding) the differences among systems is present, at different level, in every integration project.

- **Standardizing**

Eliminating their differences, i. e. imposing proprietary standards; and

- **Developing static integration methodologies**

Deciding at design time what the (static) component modules are, and what their (static) contributions to the new integrated system ought to be according to the requirements of the project.

These solutions are no longer feasible. Experience has shown that many of the systems that followed the integration strategy described above, resulted in failure or gave a poor performance. The reasons and lessons that we learned from those failures fall into the following main categories:

- **Centralization problems**

Centralized integration strategies are not possible when the component systems work in an independent fashion, so that the adoption of mandatory and/or proprietary standards is not feasible.

- **Scaling problems**

Even when a feasible centralized approach can be put in place, it has been demonstrated that classical data integration methodologies do not scale up. For example, while it is possible to keep an integrated global schema for a low number of component systems, the approach with a large number of them becomes intractable.

- **Static design problems**

Loosely-coupled networked systems with a large number of independent dynamic component nodes are becoming more and more common. Static architectures, i. e., architectures where all their modules are known at design time are no longer adequate, because every time there is a change in one of the components, the whole system needs to be re-worked; thus design methodologies, fairly static at the present time, ought to be augmented with dynamic capabilities.

- **Data interpretation problems**

Regardless of the system, of its centralized or decentralized components, of its size etc., the major problem appears reside in the data itself. Data has no public meaning and no public structure. Rather, it appears to result from the private structure, architecture and evolution of the system, and can only be exchanged and integrated by a process of pre-consensual domain --agreement on what it is and how it is to be used. State-of-the-art software tools lack the power to understand what the data is. They also lack the power to monitor data changes in component systems. Finally, the data management function is the function most affected by a broad and heterogeneous user community composed of application developers, data administrators, end-users, network

managers and systems analysts. As a result, the function is strongly dependent on environmental and organizational idiosyncrasies; data is, among all the system components, the one which is most heavily affected by internal and external organizational linkages and forces.

In a new environment characterized by rapid changes in IT-related technologies, in IT-supported operations, in the IT function, and very frequent changes in IT component systems, new integration approaches ought to be put in place. In this broad area of IT systems integration we will look at how new data integration approaches should be designed in terms their integration goals, their integration requirements and their implementation techniques.

## **1.2 The context interchange framework**

The interoperability issue that this thesis addresses is that of achieving data integration among a collection of loosely coupled and independent computer systems interconnected through the network. Particularly, we focus on the definition and development of tools and techniques to promote intelligent sharing of data among those interconnected systems.

Our strategy to provide data interoperability consists of identifying, explicitly declaring and resolving the incompatibilities originating from the different underlying assumptions attached to each data environment. The solution that we propose is framed in the Context Interchange integration strategy [Goh & others 94, Neira & Madnick 93, Siegel & Madnick 89a, Siegel & Madnick 89b, Siegel & Madnick 91].

The work focuses on data modeling and data knowledge representation and on their associated architectural issues rather than on the processing needs. The methodology that we plan to follow is described below.

The thesis begins with a historical analysis of data environments, emphasizing the evolution of data definition schemes within them [Backus 78, Chen 76, Codd 70, Davis & others 91, Hammer & Mcleod 81, Kent 78, Navathe 92, Shipman 79, Thompson 89]. Generally speaking, we notice that the task of data definition and representation has been understood and studied assuming stand-alone static systems. The analysis includes, among others, the well known record-based, semantic and object oriented paradigms. We will be analyzing how data definition is being captured and processed under each of the paradigms. With this study, we hope to gain a better understanding, and develop a better definition, of data definition and data knowledge representation --called metadata, DDL, schema, data dictionary, etc. For each of the data definition paradigms that we analyze, we cover the following topics:

- Where the data knowledge is located in the system.
- How this knowledge is documented and accessed.
- What part of the data environment, if any, is not recorded by the representation tools readily available in the system.

Our analysis continues with a review of the existing tools and methodologies supporting data sharing among a set of (component) systems [Atre 92, Bright &

others 92, Collet & others 91, Goldfine & Koning 88, Gupta 89, Lenat & others 90, McCarthy 84, Sheth & Larson 90]. The review includes integrated database systems, data repositories, warehouses, IRDS and re-engineering. We notice that in this area, much of what has been done to design open systems architectures, has been done in the software and hardware arenas. Meanwhile the systems' openness regarding their data environments remains, to this day, poorly understood, and has been neither extensively studied nor formalized.

Table 1

**Data Conflict Categories**

<b>1-DIMENSIONAL</b>	<b>SEMANTIC</b>	FORMAT
		UNIT
		SCALE
		PRECISION
<b>DOMAIN -DEPENDENT MEANING</b>		
<b>N-DIMENSIONAL</b>	<b>DOMAIN DEPENDENCY</b>	
	<b>DOMAIN -SPECIFIC DIMENSIONS</b>	
	<b>COMPLEX</b>	
	<b>ORDER</b>	
<b>NAMING</b>		

A complete model to represent data environments and to support exchanges among those data environments needs to attach data organization and meaning explicitly to the model's structures. Our research has identified several conflicts that prevent open data flows. We believe that these are due to a systematic misrepresentation of the data, and that this situation ought to be corrected by defining the above mentioned data environments. The conflicts fall into the

categories presented in Table 1. These categories will be explained in detail in Chapter 2.

To resolve these conflicts, we need to create, besides the data environment definition, additional elements in the network capable of detecting and resolving the conflicts; this takes us to our next step which is to present a complete picture of the architecture that enables data integration and open data environments: this architecture is Context Interchange.

Context Interchange Theory is a novel approach to achieve data interoperability. From the context interchange point of view, to exchange meaningful data among systems, we need to define the meaning, semantics and organization of all the data environments, i. e., data contexts, involved. In addition, we need to create global tools, i. e., a shared ontology, and context mediators to undertake context comparisons and transformation management. Briefly, the context interchange architecture (Fig. 1) defines the following modules:

- At system level, context interchange defines context clients with a data source context --used to export data-- and a data receiver context ---used to import data-- whose role is to capture the data environment characteristics --data meaning, semantics and organization. Both contexts are included as an active part of the system and they interact with the other modules of the context architecture.
- At the global level, context interchange defines a context server providing network libraries of ontologies, and network mediators modules to manage context conflicts and their resolution.

- The local modules act as clients sending requests to the global modules residing in a context server. That is, they are architected as a high level application program compliant with a distributed computing standards, i. e., client/server model (Fig. 4).

Our first contribution to the context interchange theory is comprised of the definition of a formal model of "contextualized" data exchange that organizes data around a set of conceptual entities --concrete and abstract things-- and properties. The model draws a clear distinction<sup>4</sup> between atomic and aggregated data. The formalism is built upon the following assumptions:

- The domain of a system is described as a set of conceptual entities (things), i. e., **EntityObjects**<sup>5</sup>. This finite set of concepts (things) is named using the vocabulary provided by the shared ontology. It represents the outermost boundary of the system domain. Subsets of the system domain, i. e., subdomains can be created choosing a subset of the domain's EntityObjects.
- Every EntityObject has a finite set of associated descriptors, i. e., **Properties**<sup>6</sup>. They are used to store information about the concept. In our formal description of context, these variables determine the dimension of EntityObject in the domain where it resides.

---

<sup>4</sup> This is consistent with our classification of data conflicts briefly introduced above.

<sup>5</sup> In this document we refer to the terms EntityObject, Property, MetaProperty, Link and Domain (capital letter) to refer to their unique meaning as defined for the purposes of this thesis. If they are not capitalized, then, they refer to their standard dictionary definition.

<sup>6</sup> In the context of this thesis, the terms 'property' and 'attribute' are equivalent.



- A Property must be able to represent partial knowledge about the EntityObject and accommodate multiple sub-properties, i. e., **MetaProperties**. They are needed to complete the representation of EntityObjects and Properties outside their domain boundaries. MetaProperties contain explicit declarations of the meaning and representation of the data to which they are attached.
- Domains can communicate by attaching to them a set of global relations, called **Links**. Through this scheme, we include each domain in the global ontology.

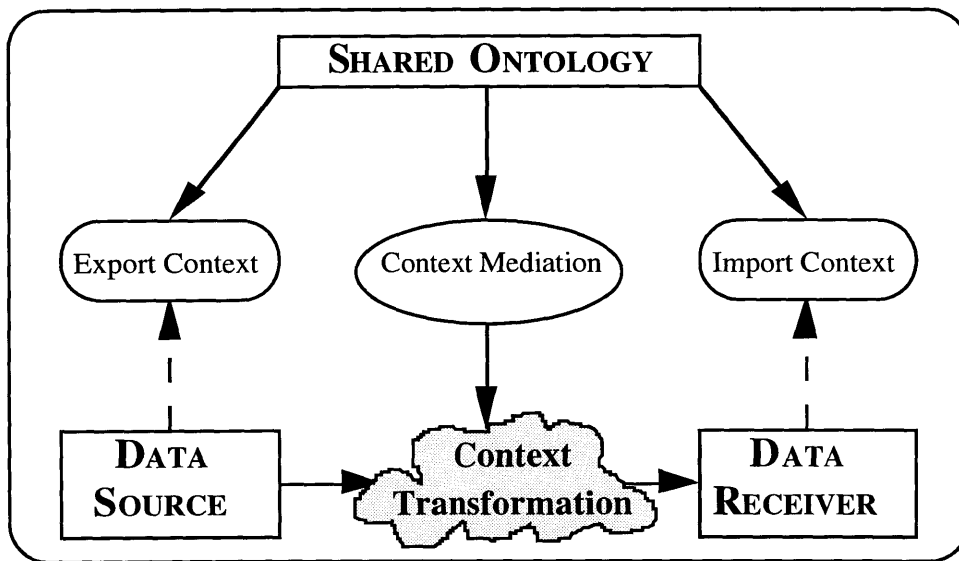


Figure 1: Context Interchange in a Simple Source-Receiver System<sup>7</sup>

The model is formalized based on Set Theory; particularly, we structure every domain as a set of n-tuples connected by relationships (Links) to outside domains. The relations are modeled using Transformations Theory paradigms, i.

<sup>7</sup> Source: [Siegel & Madnick 91].

e., cross product, cluster product and tensor product analysis. Relations are key elements to define the various ways to combine data from different sources.

Our next point deals with context interchange architectural issues, mainly we present context interchange as an application compliant with the OSI's network standards. The application consists of client nodes that transmit their data exchange requests for information on Domains to the context server. This architecture serves as the basis to structure, inter-connect and locate the modules described above.

Continuing our work in the development of context interchange theory, we present the design of the Context Definition Language (CDL), a new representation language for context knowledge, supporting interoperability among a collection of independent networked systems. CDL's role is to manage the procedural and declarative knowledge of the set of n-tuples that characterize the domain.

Overall, CDL's structures are frame-like structures [Branchman & others 85, Minsky 68, Norvig 92] specially suited to hold context knowledge. The knowledge they contain is intensional --the CDL structures can be used in recognition and comparison processes. These reasoning processes (augmentation, generalization and matching) are provided by the global modules of the context interchange architecture; they intervene to perform the data environment comparison and transformation management defined above. All the CDL components have been created under a modular design approach that encompasses the definition of sub-units, i. e., layers of data definition. Each one deals with the representation of a different aspect of context knowledge.

CDL frames have different structures depending on the role they play in the context architecture:

- **Source domain description**

Network nodes and their aggregated domain-specific structures at high-level as Domains with EntityObjects.

- **System-level structures description**

These are the domain-specific EntityObjects and the Properties attached to them.

- **Receiver description**

They are represented as global network queries requesting any combination of EntityObjects, Properties and Domain information.

The definition of a shared (global) ontology against which to match the categories --EntityObjects and Properties in CDL's nomenclature-- found in the various network node domains, constitutes an important CDL-related issue that we emphasize in this thesis.

To support our analysis and conclusions with real examples, we choose to analyze different data environments. Each of them pertains to a very different domain, and we give a detailed description of its contents. Afterwards, for reasons of brevity, we choose a subset from them to represent and embed in the newly defined CDL frames. In doing so, we demonstrate the feasibility of context analysis and context acquisition in operational network nodes.

Finally, we develop a prototype, using the C++ programming language, demonstrating how to build a data environment definition directory. The environment definitions are incorporated into a global hierarchy of domains. The prototype assigns to this global hierarchy --global ontology library-- the function of a network domains directory. Every domain is assigned a virtual file. The file stores, in the form of CDL objects, the network nodes containing information in that domain.

The basic contributions of this thesis to the context interchange approach to systems integration are:

- The design of CDL layered structures to capture context knowledge. CDL design responds to the need of a standard language for information exchange among a collection of network nodes.
- A model of the network systems as a collection of independent and autonomous nodes. Each node is expressed as a Domain described using the common vocabularies provided by the shared ontology. Furthermore, we can locate and access these nodes by accessing and manipulating their CDL representations.
- The decomposition of context interchange application into modules that meet open system architecture standards.

### 1.3 Outline of thesis

The following chapters of this thesis attempt to describe a new architecture to design open data environments based on the context interchange framework. In doing so, the research questions will be addressed thoroughly.

**Chapter 2** presents the background literature on data environments. The first part of the chapter provides a summary of the history of the knowledge representation techniques used to capture data environments; these techniques range from types, through data models, to repositories. The second part of the chapter presents a complete classification of the data misrepresentations present in those environments.

**Chapter 3** presents the architectural aspects of context theory. A formal model to describe data environments as a collection of independent networked systems is presented. Finally the chapter covers the definition of the CDL and its frames to encode the data environment characteristics, i. e., context knowledge at different levels.

**Chapter 4** analyzes different domains, demonstrating how to incorporate context to them; that is, how to open their data architectures.

**Chapter 5** using the C++ programming language, implements a context prototype demonstrating how to build a network library of domains (each domain containing its characteristic ontos). The library is consulted in domain matching and transformations needed as part of data exchange processes.

**Chapter 6** summarizes our theoretical developments and work, and comments on the many advantages of our data integration strategy. It also includes conclusions and future research.

## CHAPTER 2

### DATA ENVIRONMENT REPRESENTATION

This chapter presents the evolution of the techniques and the tools used for data environment representation, i. e., context knowledge representation, in computer systems. The work focuses on knowledge representation by means of data types, data modeling, data dictionaries, system schemas, network directories, re-engineering, thesaurus and repositories, and on the associated architectural issues rather than on the data processing needs. The survey of literature aims to identify:

- Where in the system is this knowledge located.
- How it is documented and accessed by the users.
- What part of the data environment, if any, is not incorporated into the environment.

At the same time, our survey includes an evaluation of the expressiveness, and flexibility of the above mentioned paradigms to reflect an accurate picture of what the context of the system is. Thus we will be looking at models, types, dictionaries, etc. from a knowledge representation point of view.

Afterwards, we present a classification of the most commonly found data disparities that prevent meaningful exchange. The analysis stresses that these incompatibilities, originated from conflicting data environments, can not be detected and/or resolved by applying state-of-the-art tools.

Incompatibilities emerge only during information exchange processes; then the lack of a complete representation, the lack of completeness, and the many implicit assumptions become the key issues that keep data locked to a particular environment. Yet these implicit assumptions and misrepresentations constitute part of the information about data needed for meaningful interchange. We need to incorporate that information to the system.

Therefore, our previous analysis demonstrates that data interoperability requires more than what current representation schemes offer regarding data environments. As corollary to this demonstration, we identify the need to close the gap between current representations and a complete representation, i. e., context knowledge. By adding meaning, semantics and organization to their definitions, data environments become self-describing and open sources and receivers. That is, they become the import and export nodes of the context interchange architecture.

## **2.1 On data structures and knowledge representation**

In this section we undertake an in-depth review of the existing schemes used to capture data environment information. Overall, they consist of the data structures provided by the computer language and/or the system's data manager



program, plus the semantics imposed upon them during the logical data modeling phase. These imposed semantics aim to model the domain of the system using a very small set of distinctive data structures (categories). The field uses mechanisms that allow designers to hide details and concentrate on the common target features of data objects. The methodology and tools are very similar to those used in the field of knowledge representation to describe domain categories, and they are generated by applying the following techniques:

- **Conceptualization**

Is a technique used to provide a way for naming different concepts that belong to the real world but which are difficult to manipulate in a computer environment.

- **Categorization**

Is a technique that groups under a common denomination all the entities of the world that present equal values under a set of properties that are chosen to define the category.

- **Abstraction**

Is used as a way to detail and concentrate on general common properties of data objects. The different models that we will be presenting, capture several types of abstraction mechanisms. Abstraction is also part of the processes of categorization and conceptualization.

- **Generalization**

Is used to hide the differences among concepts and categories and group them into a higher level, i. e., more general concept or entity.

In the field of information processing, imposed semantics are often times called data representation standards. As we will see later on this chapter, some highly developed and mature data environments, contemplate the design of a module to define data model and data semantics standards. This module --repository, warehouse, etc.-- is a central point in the architecture of the system and is accessible across an existing portfolio of networked nodes consisting of applications, databases, and development environments. Its main role is to provide a place to store the characteristics of the component data environments, enabling communication among them.

Notice that, in many cases, the main goal of these tools has been not to represent data environment and data knowledge per se, but to achieve at physical (disk storage) level efficient ways to organize and access large-scale data banks. Also notice that the different models that we are presenting use a very rich collection of abstraction, conceptualization, categorization and generalization mechanisms. However, all of them use a very limited collection of representation structures to implement these many types of data knowledge representation mechanisms. Obviously, the result is that the mappings from real world concepts (things) to the computer structures are many-to-many. This multiple mappings create ambiguities and misinterpretations that prevent, we will see exactly how, the exchange of data across environments.

In this chapter, the evolution of data environment representation (Fig. 2) is traced emphasizing in the Information Systems (IS) arena. We choose to do so because IS has been historically the area of Computer Science that has adduced and formalized in different application environments --industry, university, R&D,

etc.-- and under very different processing requirements --transaction, interactive, etc.-- the representation of a data environment. Moreover, we must admit that the field of data environment representation in computer programs has been almost exclusively addressed, and largely dominated, by IS/DBMS programs. Thus data environment representation is largely influenced by the representation and modeling of data as applied to the design of character-based database structures.

Under various denominations - data types, data models, data repositories - these schemes have been used, some of them for more than twenty-five years, to describe "reality" --the part of the world that the system manipulates. This thesis presents them in chronological order. This chronological organization allows us to follow the evolution of the field from its beginnings, with only primitive types and physical storage concerns, to the present times, where representation of data meaning and semantics are being considered as the main concerns.

The reader should keep in mind that the part of the data model that we target is the part concerned with the data description, DDL (Fig. 2), as opposed to DML, database programming language and database query language.

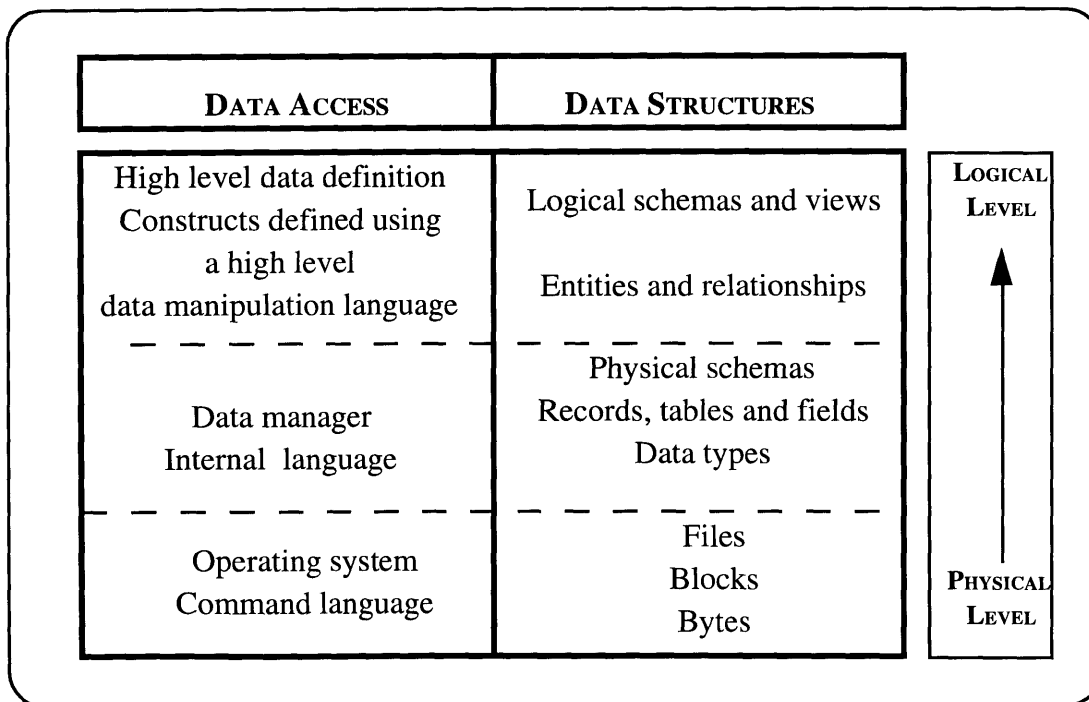
### **2.1.1 Data Types**

Data types [Martin 86] are the basic units provided by every computer language as the way to encode variables representing real world objects. They range from simple primitive types to complex types. They can be classified into three major categories:

- Primitive types such as binary, integer, real, character and string.

- ADT/CDT: text, ordered set, matrix, time series, vector, structure and list.
- User defined types

Types constitute the simplest scheme for data representation in computer programs. The abstraction mechanism is implemented mainly by defining categories with a name (category name) and a system-wide type associated with that name. In general, type-based data environment representation is not very rich and not flexible either. The many user defined categories need to be translated into a small set of pre-defined system types. As a consequence, the capabilities of this approach for representing specialized knowledge, super and sub-categories and complex objects are fairly limited.



*Figure 2: Levels of Data Representation in Computer Programs*

Nonetheless, we must stress that the concepts of ADT/CDT are very powerful when incorporated as part of a data model, for example, as part of the semantic data model<sup>8</sup>.

### **2.1.2 Record-based models**

There are many different data environment representation schemes grouped under the common denomination of record-based models. The first of these models to be developed was the flat file model where data was stored in a file with no structure, i. e., flat file. Normally, the records were stored with an implicit structure and organization for retrieval efficiency purposes. In this model, data knowledge representation has similar characteristics, and therefore similar limitations, to the ones present in data type models.

Search and storage efficiency, that no data representation concerns, created, as an evolution of the primitive flat file models, the hierarchical and network models [Benyon 90, Kent 78, Kent 79a, Navathe 92, Wiederhold 84]. These models present the database as a collection of (cross-connected) files. Cross-connections among the files are provided by embedded indices. Both models emphasize physical implementation as opposed to the logical implementation of the database data definition.

The relational database model [Codd 75, Date 85] is the most popular of the record-based models. Compared to most other models, it has a formalism which is based on predicate calculus. The model organizes data in table-like files called relations. Each relation is a set of tuples, of the same type, where each tuple is essentially a row of data in the table. A relation's logical structure is defined by a

---

<sup>8</sup> See Section 2.1.3.

group of attributes; each attribute is a column heading in the table. Each attribute is assigned to an attribute domain. This domain is restricted to the definition of the data type and the allowable range of values for all the possible occurrences of the attribute.

In this model, everything is represented in the above mentioned two-dimensional table construct<sup>9</sup>. This table data structure contains all the available descriptive information about the data. The structure of the database (i. e., database schema) is implemented creating, besides the tables themselves, logical connections (links) among tables, e. g., using the pk-fk (primary key - foreign key) feature.

The functional model [Backus 78, Shipman 79], ultimately derived from lambda calculus, represents entities, attributes, and their relations and operations. It models all of them as functions, with their associated domains and ranges, or derived functions. The database consists of a set of objects which are either entities or attributes and functions to perform the mapping between objects. Derived functions are inversions, compositions and restrictions of other functions and can be used in the model to define the relations among existing entities.

These record-based models have limited their abstraction capabilities to the definition of a limited set of categories. They were not flexible and expressive to a limited extent. Their limitations have been widely documented in [Kent 79b, Kent 91, McCarthy 84]. These models lack powerful logical representation schemes, and focus on database physic implementation and data structures.

---

<sup>9</sup>The relational system has the closure property with respect to the table structure; that is, it is a closed system with respect to tables.

Their knowledge representation problem is very well phased and summarized in [Davis & others 91] where the authors assert that data representations are not data structures, and that is one of the limitations of these modeling paradigms.

Nevertheless, data structures can be turned into representation tools by imposing semantics on them. This is the key idea that moved researchers to develop the theory of our next section: semantic models.

### **2.1.3 Semantic models**

Semantic data models [Chen 76, Elmasri & Wiederhold 81, MacCathy 88, Thompson 89] constitute a significant move, with respect to the previous modeling paradigms, toward richer representation schemes with more powerful conceptual design tools. The data semantics paradigm aims to attach more meaning to the data. It does so by defining data and data relationships, and unique environment-defined mappings to the standard data manager-defined structures, e.g., relational structures.

The most important move that semantic models brought about is that, for the first time, they incorporated a clear distinction between physical data and logical data. This dichotomy is implemented by defining within the system two levels:

- Specification level.
- Information structures and transactions level.

To go from one to the other, there is a translation process, i. e., mappings, between the specifications and the real data constructs. There are two key

characteristics of this mapping process, the first is the addition of constraints not present at the specification level. The second is that the mappings between the two levels are not unique.

The most widely spread of these models is the Entity-Relationship (E-R) model. Chen [Chen 76] was the first author to propose a high-level information description scheme as part of the data modeling and , of course, as the first step in the modeling process. It is intended only as a logical database design tool. It defines entities, relationships and attributes with no data operations for them.

#### **2.1.4 Object-Oriented models**

From the logical design point of view, these models [Atkinson & others 90, Kim 92] are essentially indistinguishable from those based on the previous semantic paradigm, i. e., it is always possible to add object-oriented (O-O) features to a semantic data model. Thompson [Thompson 89] states that, in some senses, semantic models are a generalization of the concepts of these object oriented models but they do not have the encapsulation and implementation structures that the O-O approach does.

O-O eliminates the distinction between entities and relationships characteristics of the semantic models. The O-O data models apply to both the conceptual and logical design phases. They are similar to semantic models in the structural abstraction concepts and the possibility to perform value and type composition. But they differ from semantic models in that they have embedded as part of the model, formal ways to:

- Use system identifiers.



- Implement inheritance.
- Implement information-hiding and encapsulation.

Up to this point, we have been studying the data environment representation paradigms<sup>10</sup> that concentrate on stand-alone data environment representation. Now we shift to study the strategies that have been proposed for modeling data in the presence of a collection of existing systems.

### **2.1.5 Integrated Databases**

There have been several IS integration projects<sup>11</sup> [Atre 92, Gupta 89, Madnick & others 90, Sheth & Larson 90]. Their approaches look at integration from two different points of view:

- **Composite databases**

This strategy creates a global schema to hide the component systems and their differences, and presents the user with a unified view of the data.

- **Federated databases**

This strategy creates many accessible local schemas along with the tools necessary to interact with them.

From our point of view, they provide similar tools for data environment representation. More importantly, if discrepancies exist, they are resolved at

---

<sup>10</sup> Sections 2.1.1, 2.1.2 and 2.1.3.

<sup>11</sup> A complete survey of research prototypes can be found in [Gupta 89].

design time with static schema reconciliation techniques. It is a very common solution to establish an enterprise model that provides an extended view mapped from different databases irrespective of their models. An example of this would be CIS/TK [Madnick & others 90, Wang & Madnick 89b] a prototype HDBMS with real-time access and retrieval capabilities to data residing on disparate autonomous information systems. CIS/TK integration capabilities include data model and data semantics reconciliation.

A more recently developed project, the Carnot project [Collet & others 91, Lenat & others 90, Shen & others 91], takes a unique approach to systems integration. Its unique strategy is based on the definition, development and implementation of a dynamic schema: the Cyc knowledge-base.

#### **2.1.6 Data repositories, data warehouses and metadata**

We must remember that, strictly speaking, tools supporting repository services have been around almost as long as computers. They have been known as libraries, thesaurus, glossaries, etc. However, they have been systematically and redundantly encoded within multiple facilities with dissimilar syntax, taxonomy and semantics. They were never thought of as a network systems integration tool. But it has been only recently that the proper attention to their central role in systems integration has been recognized.

Since a few years ago, several researchers have suggested the need to store, explicitly within the system, knowledge about the data, e. g., metadata [Goldfine & Koning 88, Mark & Roussopoulos 86, McCarthy 84, Siegel & Madnick 89a, Siegel & Madnick 89b], rather than storing only the structural data properties. In order to provide a systematic way to store and access the (metadata)

environment information, new tools such as repositories, warehouses, and knowledge bases have lately emerged. Later trends have demonstrated that these tools need not only provide for metadata storage but also provide an organized way to access metadata, to cope with data replication and synchronization, and to keep track of data distribution, location, interchange, sub-environments (views), with possible changes in all of them. In all these cases, the repository contains only metadata, i.e., Repository-Objects, Repository-Relations, etc.

These elements, part of the intersystems' architecture, can be defined as being specialized databases/knowledge bases, where all the enterprise modeling is stored and dynamically accessed. In addition, this database/knowledge-base may also include information related to project management, as well as related data requirements, design, implementation, testing and environment development. The data stored could be in the form of documents, specifications, source code, test data and a myriad of other forms essential to the specifications of the system.

Repositories and data warehouses come in very different varieties. In general, they provide data integration capabilities needed for the development and maintenance of information systems. They have been developed to suit the needs of environments where the application portfolio consists of many applications built around a series of common databases. Their power lies in the fact that application programs, computer languages, system software and database management products can use a common set of data definitions stored in the data warehouse or repository.

The reason why these tools are widely used is because they support the many emergent forms of distributed computing. They constitute the sources for determining:

- How to access data.
- How to represent data.

For example, a data dictionary, sometimes called a directory system can be designed to comprehensively support the logical centralization of data about data --this is why they were thought of as metadata storage facilities. In many cases, they not only provide storage and access to this type of information but also support cross reference information about the metadata (which is normally placed in the above mentioned directory). The dictionary provides information about what the data is and how it can be accessed. It is an automated facility meant to support data administration in a distributed environment.

At present, the repository pretends to become a broader concept that includes tools for data access, data sharing and data management, and, in turn, it pretends to play a more crucial role across the system's life cycle. For example, repository technology is widely used in the application development environment (ADE) where it acts as the system's knowledge-base and serves as a key element for the CASE tool integration strategy. For example, the repository stores information on how the components of an information model and its associated databases are related. It also gives the power to navigate and show the relationships among application components.

From our point of view, the major challenges that these tools [Devanbu & others 91, Giordano 93, Jones 92, SIGMOD 91] will have to address in the near future are:

- **Life cycle**

How to represent the evolution of the system and its data; particularly the maintenance of accuracy and integrity constraints.

- **Content**

To define what their exact contents ought to be is a very difficult design question. This is especially true in new integrated environments e. g., new client-server technologies, where PC-based files, spreadsheets, and many other kinds of network-accessible objects are present. The meta-representation of these objects in a warehouse or repository is going to make their implementation a harder task i. e., they must be able to cope with not only normalized sets of databases but also with files where data is represented at many very different levels of abstraction.

In spite of the ongoing evolution of these tools, current repository technology just provides "data repository services" --data definition storage services. It does not provide "data integration services." Both of them, when combined will constitute a very powerful and sophisticated set of "integrated data management services" and supporting tools: that is the direction in which these technologies are evolving.

### **2.1.7 Re-engineering data environments**

The process of re-engineering [Andrews & Leventhal 93] consists of an engineering step backward followed by an engineering step forward. During the backward step, re-engineering takes the past design and extracts the essential problem domain content. During the forward step in engineering, the problem domain content --model of the application-- becomes the basis for re-implementation in a new medium. The rationale for re-engineering lies in the fact that great savings from reusing code, and reusing the data accessed by that code, can be realized. Re-engineering is applied to a wide variety of software engineering areas such as programming code, databases, inference logic and expert systems.

In our field of study, data environments, re-engineering is used to migrate between (database) modeling paradigms and between different implementations of a database paradigm, for example from one vendor's relational database to another's.

Re-engineering provides valuable background and many lessons to our study. First, it has demonstrated that the data knowledge and data structures contained in the model are not enough to give a full definition of the data. In order to define what the data is, it looks not only to the model constructs and to their implementation in an application schema, but also to observed patterns of data, and semantic understanding of the application. Second, it has demonstrated that extensive dependency analysis and domain analysis are the necessary starting points to understand the data and move it to a new stage. Semantic clues have been identified as a critical part of its success.

Through the exclusion of implementation-specific information from the design process, system dependencies may be reduced. Their removal allows the user who is unfamiliar with the system to understand its functionality and contents as if he or she were inside the system.

## **2.2 On data environments and interoperability**

### **2.2.1 Examining data environments**

A data environment can be thought of as having two kinds of information associated with it:

- **Explicit knowledge**

This includes the schema, the data dictionary, the data type definitions or any combination of these. This knowledge is the target of existing data environment representation schemes. It is an active part of current operational systems, and it is captured by means of the DDL.

- **Implicit knowledge**

This includes the different assumptions about the data that have been encoded using the DDL. These assumptions include, among others, units and scale of numerical data, meaning of data, temporal domain and scope of data. They create conflicts that current systems cannot cope with. Our analysis of data conflicts<sup>12</sup> will show a complete classification of implicit knowledge in the form of 1-dimensional, n-dimensional and naming conflicts that can prevent meaningful data exchanges.

---

<sup>12</sup> See Section 2.3

Most of the existing systems have been developed based on the representation of only explicit knowledge, which results in the de-coupling of meaning, structure and representation of data. Furthermore, they have been developed as stand-alone units each with its own implicit syntax and semantics.

In the past, systems isolation has allowed data environments to be systematically misrepresented. Accessing the data was limited to local users with the support of data administrators, i. e., the persons who imposed the valid mappings, the units, the constraints, etc. on the data. This "closed system" status made the analysis and representation of the data environment unnecessary.

After years, oftentimes, this situation caused problems whenever a person new to the system was faced with a mature undocumented data environment, in the absence of data administrator support: The (implicit) data environment characteristics were neither defined nor kept anywhere.

On the other hand, other projects which attempted to fully represent a data environment failed to do so, due to the semantic limitations of the data models, mainly the many-to-many nature of the mappings between model structures and schemas.

However, the situation has changed. The computer network is now a reality, and systems need to adapt to it. As happened before with other parts of the system architecture, data environments should open their architectures and adhere to the open systems paradigm.



### 2.2.2 Closed systems, open systems and their differences

In the open systems world that we pictured in the previous chapter with a large number of systems connected through the network, there are incredible benefits to achieving data interoperability. Nevertheless, especially in data-intensive computing environments, all these benefits do not come without cost. There exist serious problems to be solved before we can achieve and take advantage of full IT systems connectivity.

Continuing with our discussion of open data environments, we now turn to study why data definition and representation requirements change in an open environment. From our point of view, these are the most problematic issues that we encountered:

- **Static and closed data environments**

Traditionally, systems have been closed, and their data environments are no exception to the rule. The closed world assumption inevitably creates closed world systems<sup>13</sup>. However, in networked systems the close world assumption does not hold any more, causing environment interferences whenever the norms from two data environments conflict: assumptions appropriate to the context of one application may not fit the context of other applications. In addition, data environment representation focuses on the representation of (static) declarative knowledge [Morgenstein 84], and it does not provide the means to capture dynamic changes, characteristic of open environments.

---

<sup>13</sup> The closed world assumption states that if a fact that is not provable (true) then it is by definition false. Closed systems are known for having a feedback mechanism that derives all the information from within the system itself, i. e., the system ignores changes in the state of the environment.

- **Local meaning vs. global meaning.**

Data do not contain nor transport meaning; interpretation of data is contextual and done at the local level. The local environment and its assumptions are not an explicit part of the description of the system and, therefore, they cannot be incorporated and transferred along with the data. The large majority of the data disparities that we encounter are due to the fact that they reside in different worlds and therefore they are expressed according to the conventions of their worlds.

As a result, we need to develop new technologies and tools capable of addressing the transformation of data environments from their closed to their open state, precisely defining all the changes involved in the process.

### **2.3 Data conflicts over the network**

Continuing our study on techniques and tools for data definition, this section presents a classification, and give examples of data conflicts across environments. The classification poses a categorization of the kind of problems that context interchange addresses, and , in that sense, gives us the basic guidelines for what we need to build a complete data environment definition, including the data environment transformations and comparisons that we intend to address in the next chapter.

We have grouped the conflicts under the common denomination of inter-data environment problems, to stress the fact that they constitute the data definition

variables. These variables vary, implicitly and explicitly, from one system to another causing incompatibilities among them and preventing the integration of their data.

### **2.3.1 1-Dimensional conflicts**

We define as 1-dimensional conflicts those that occur at individual data item level, as opposed to those that occur at a collection of individual data items level. This collection of atomic data units are grouped in a single unit by means of the DDL, or by other similar support tools provided by the data manager present in the system. We also include in this category conflicts defined across equi-dimensional domains, i. e., domains where the groups of data items are homogeneous, because they can be decomposed into a finite set of n 1-dimensional conflicts.

#### **2.3.1.1 Semantic conflicts**

This conflict is due to the existence of different representations of a single (atomic) conflict. This type of 1-dimensional inter-domain conflict can be resolved by the declaration of semantic values and transformations operations, such as functions, tables and heuristics among them.

This type of conflict has been identified and studied by Siegel and Madnick [Siegel & Madnick 89a, Siegel & Madnick 89b, Siegel & Madnick 90, Siegel & Madnick 91]. They state that semantic conflicts are tractable by the definition of semantic domains and their incorporation into the schema of the system, semantic values and semantic value transformations. A sub-division of semantic conflicts based on the different aspects of data representation is:

- **Format conflicts**

Heterogeneous formats exist for representing basic data types and/or basic user-defined data types. For example, in Table 2 we can see that the same one thousand amount can be represented in plain format (1000), USA format (1,000) and Europe format (1.000). In many instances, the format is a intrinsic characteristic of the system. That is, every system has its own way to represent its basic types --integers, real numbers, character strings and such. Therefore, format heterogeneity is present in almost every data interchange process.

Table 2  
1-Dimensional Format Conflicts

Format	Representation
Plain format	1000
USA format	1,000
Europe format	1.000
Exponential format	$10^3$
etc.	.....

- **Unit conflicts**

They are due to the representation of the same thing in two different reference systems. For example, we can find the same quantity expressed in one system as 1 kilogram and in other system as 1,000 grams. Other times the unit conflicts are far more complicated. In the case of spatial data, transforming from one unit (reference) system to another, e. g., USGS to UTM coordinates, implies the definition of complex transformation functions. They range from linear interpolations to cubic convolutions, and, in many cases, they can be applied to the whole object, and they are only defined in one direction, i. e., there is no inverse function. Unit transformations can be

performed intra-domain, in which case it is very common to have a master-slave style of unit conversion.

- **Scale conflicts**

They can be considered as a special case of unit conflicts. They occur when a domain expresses a global base unit implicitly transformed by multiplication or division by a constant term. For example, 1 and 1.000 can express the same quantity in the same unit system but, in the first case there is an implicit division by a thousand. The problem with scale conflicts is that the scaling factor is domain-unique, and depends on the characteristic of the context.

- **Precision conflicts**

This conflict is present when two systems have different implementations of accuracy. For example, 1.00 and 1.03 express the exact same quantity in two different domains. Rules for rounding off numbers when changing context should be established. As there are no general rules, it should be treated as case-based reasoning. In the case of non-numerical data, precision conflicts come in many varieties, including no type homogeneity across domains; in these cases, to solve the conflicts we need functions to perform inter data-type translations. For example, a location indicator for MIT can be Cambridge, 02139, Massachusetts, and many others.

### **2.3.1.2 Domain-dependent meaning**

A meaning conflict can occur when two domains perceive the world from different points of view. In this case, the conflict causes different interpretations of sub-domains.

Our example to explain this sub-domains conflict (Table 4) deals with looking at two different domains: a database containing health-related data on persons, and a database containing historical series of weather data. The conflict pertains to the common sub-domain of temperatures.

In the domain of human body temperatures [Shoman 91], the mapping function from numerical values (left row of the table) to qualitative values (right row of the table), the function would qualify a temperature of 37.00 °C as normal:

Table 3  
Domain-Dependent Conflict

Temperature (°C)	Temperature(qualitative value)
.....	.....
37.00	Normal
.....	.....

Now, if we study the same mapping function in the domain of weather temperatures, we would observe that the same mapping function indicates that 37.00 is "High." This is a clear example of meaning conflict due to the fact that qualitative description of numerical values is sensitive to the domain in which the functions are defined. Thus any attempt to define absolute qualitative temperature values needs to take as a parameter the domain in which the scaling is defined.

Another example of this kind of conflict is shown in Figure 13. In this case, the domain of Map Features interprets the rivers as linear features while the subdomain of Water Body sees the same river as a surface feature.

**2.3.2 N-dimensional conflicts**

N-Dimensional conflicts occur among data structures defined by the data manager residing in the environment. They involve the definition of high-level groups of (atomic) data units.

### 2.3.2.1 Inter-domain conflicts

Inter-domain conflicts are present when the same concept has different sub-representations across environments. Even though the same concept exists across the domains, its implementation is different among them. Table 4 shows the field representation and sub-fields representation of a person's full name. In this case, the intra-domain dependency differs depending on the country, which acts as a supra-domain. Here we are assuming that names of persons are broken up into three fields (Field-1, Field-2 and Field-3 in Table 4). Therefore we have the same property, for the same object but with different content in each domain:

Table 4

**List of Name's Sub-Components by Country**

Country	Field-1	Field-2	Field-3
Holland	Family Name	Street address	
Japan	Family Name	Given Name	
Spain	1st Family Name	2nd Family Name	Names
Sweden	Family Name	User Supplied String	
USA	First Name	Middle Name	Last Name

Tanenbaum [Tanenbaum 88] describes the problem represented in the example of Table 4. His work concerns the integration and queering of information present in public network telephone directories. The difficulties to achieve integration are due to the fact that different legal jurisdictions assign different structures to a same (name) concept; that is, a person's full name content differs

from country to country, i. e., country domains assign different properties and structure to the same concept; still, at data item level, in any of these cases (Table 4) it can be confusing to decide how to split into the above fields names such as John von Newman.

**2.3.2.2 Domain-specific dimensions**

Dimensional conflicts come from the fact that the same concept can be expressed with a different number of variables. In qualitative terms, we can explain this conflict by saying that different domains represent the same object at different degrees of specificity. The degree of specificity in which a particular object is represented depends on how much information the system-domain is interested in keeping about the object. Table 5 shows the different variables --Properties of an EntityObject in CDL nomenclature-- under which a person is expressed in two different databases.

Table 5

**List of Dimensions and their mismatches**

<b>City Hospital Database</b>	<b>City Hall Database</b>
Patient-Name	Full-Name
ID#	SSN
Place of birth	??
??	Citizenship
Blood-Type	??

Figure 3 shows another example of dimensions mismatch. There, the same information is expressed at different levels of abstraction by using different sets of variables. These different levels of abstraction are due to the representation of the same concept (amount of rain over a certain region) at different degrees of specificity. But the systems in our example need not be incompatible. In this case,



a hierarchical structure of meanings, like the one we will propose in Chapter 3 will enable us to compare different abstraction degrees and different types of the same concept, when the proper transformations are undertaken. That is, if there exists a function to transform daily precipitation onto their monthly mode and std-deviation and vice versa, then these systems will be compatible.

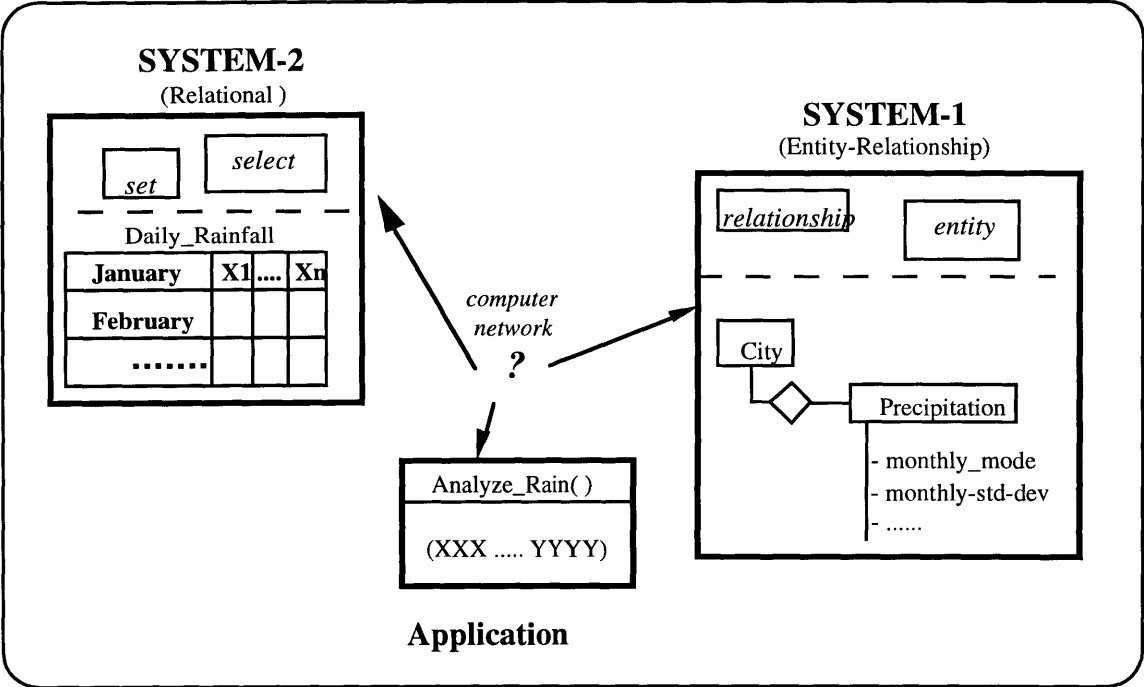


Figure 3: Example of N-Dimensional Conflicts

2.3.2.3 Complex conflicts

Another group of data conflicts are those involving concurrent engineering design projects. In these cases, the views of what usually are very complex objects, change radically from one phase to another of the project. These changes involve a mix of the previously described data conflicts: inter, intra, domain, naming, etc. conflicts. This involves a change in domain and a change in the structure which needs the re-interpretation of all of existing features.

Views of complex objects and their associated conflicts are very commonly found in two-dimensional, three-dimensional and n-dimensional data sets. Some examples of possible conflicting views are given below:

- A block in a city is represented in a record-based database by its address while in a geographical database it is represented as a polygon with an area.
- In a regional map a city has an area while in a World map the city is reduced to a point.
- Objects belonging to disparate domains but represented under a common structure, and, of course, under a common set of properties change their meanings when assigned to different domains<sup>14</sup>.

In addition, there are serious data integration problems when the data types do not fit standard types, relational structures or other common data structures. An example of this would be the images, animation, music, etc. present in multimedia systems. In such environments, issues of how to define the syntax and semantics of network-wide searches, and how the target information is to be merged and filtered are, to date, complex data integration issues that need to be resolved.

---

<sup>14</sup> For example, the reader can identify this type of view conflict in the Appendix 2 where with in the same data environment, the Alumni/Gift environment, the fields of the ALF-MASTER file change their meaning depending whether they refer to an Alumnus EntityObject or to a company EntityObject.

#### 2.3.2.4 Order conflicts

N-dimensional domains need not to be compatible with respect to the order into which they organize their sub --N-- domains. For example, suppose the conflict between a database that stores traffic flows as right-lane flow and left lane flow, and it wants to communicate with an application program that processes the same traffic flows as inbound-lane and outbound-lane flows.

#### 2.3.3 Naming conflicts<sup>15</sup>

Naming conflicts are due to different ways of writing a concept, property, etc. They can be classified into two major categories:

- Synonyms are encountered when different data environments use different names to identify the same subject.
- Homonyms are present when different systems keep a (single) common denomination for different objects.

We should point out that naming conflicts are very common at data definition level (metadata conflicts), and at data instance level<sup>16</sup>. We believe that both of them have the same nature and therefore we propose to treat them formally in context interchange theory by applying to them the same formalisms.

---

<sup>15</sup> For a more detailed study of naming conflicts the reader is referred to [Wang & Madnick 90].

<sup>16</sup> Even assuming that all the systems in the world decide to refer to a company name with the logical mane "CO", we would still have to elaborate on the inconsistencies of the spelling in the many systems containing this field.

## 2.4 The architecture of open data environments

In Section 2.2, we studied and classified data conflicts. From that study, we concluded that data from autonomous sources cannot be exchanged unless we define a priori what their meaning, their representation and their properties are.

On the other hand, in Section 2.1, the design approaches to data representation that we examined lead us to conclude that there has been a development of a certain small set of structures to represent the world's data. These structures are lacking powerful means to represent meaning, organization and semantics of data. At the same time, there has been a stand-alone style of design in which the missing parts of the system's context were not explicitly coded but part of the system's implicit characteristics. As a consequence, the current data modeling approaches lack tools to represent the context of data, i. e., data semantics, data meaning, internal functions to generate new (derived) data, etc. They lack tools to compare different environmental characteristics, tools to resolve incompatibilities and like mechanisms needed to achieve network interoperability.

From a knowledge representation point of view, current data environments use a very limited collection of representation structures which attempt to capture a very wide range of abstraction, conceptualization and categorization mechanisms<sup>17</sup>. Until recently, the fact that they were being used in stand-alone mode minimized the emergence of data environment conflicts as well as the emergence of their representation weaknesses. But now we need to extend the

---

<sup>17</sup> See Section 2.1.

knowledge representation capabilities of current data models to include modeling of these environment characteristics, as well as transformation functions for conflict resolution. However, we need to be capable of modeling not only the information which has previously been processed but also designing a repository-manager for that data definition information.

First, context interchange theory defines is the export-import context (Figs. 3). They are both included in the system as an extension to the type and/or the schema definition, functioning as an active data repository of the node which is named context. At this local level, context, acting as a repository of knowledge for data structure, data organization, data semantics and data meaning, closes the gap between data and information that previous data definition approaches did not. The import and export contexts are located above and attached to the existing model structures. The result of the process of import/export context implementation will include:

- Declarative domain knowledge in form entities and their lists of properties, represented according to a preexisting ontology.
- Explicit procedural definition of data organization, meaning and semantics in the local systems.

Then, at a global level there will be a context server containing:

- Global ontology.

- Mediators performing query execution and planning management as well as conflict resolution management.
- Transformation libraries storing the functions needed for context transformation.

Figure 4 pictures a network whose members are open data environments. Openness has been embedded in the network by incorporating context clients and servers. Later chapters of this thesis define (Chapter 3) and demonstrate (Chapter 4) how context can be articulated around the existing data and incorporated into the system in the form of frames containing declarative and procedural knowledge (heuristics, rules, tables, constraints, etc.).

## CHAPTER 3

### CONTEXT DEFINITION

This chapter focuses on modeling, formalizing and representing context interchange components. First, in order to define the components of the context interchange model, we choose different operational systems and study the variables involved in the definition of their contexts, the functions to manipulate the variables, and a decomposition of these functions into modules. We place particular emphasis on the definition of the interactions among them.

We show that the outcome of the previous study, i. e., modules of the context interchange architecture match the findings of Chapter 1 and Chapter 2, where we introduced a preliminary modular decomposition of context and its functions into:

- Autonomous source/receiver nodes.
- Network libraries (global knowledge-bases).
- Context mediators.

They constitute the building blocks of our interoperability architecture because the ability to open the system's data environment and ability to share information is provided by them.

In this chapter we present the knowledge content and structure of the above listed modules, i. e., their knowledge content and the structure by means of the newly defined CDL structures.

Table 5  
Conflict Table

CONFLICT TYPE		EXAMPLES
1-DIMENSIONAL	SEMANTIC	FORMAT <i>MA vs. Massachusetts</i> <i>1.000 vs. 1,000</i>
		UNIT <i>UTM vs. State Plane (static)</i> <i>USA \$ vs Japanese ¥ (dynamic)</i>
		SCALE <i>1 vs. 1000</i>
		PRECISION <i>1 vs. 1.003</i> <i>Cambridge vs. 02139</i>
	DOMAIN-DEPENDENT MEANING <i>37.00° [Normal in Human Body domain]</i> vs. <i>37.00° [High in Weather Forecast domain]</i>	
N-DIMENSIONAL	DOMAIN DEPENDENCY <i>Parts of a person's full name</i>	
	DOMAIN-SPECIFIC DIMENSIONS <i>Person [SSN, Name, Department]</i> vs. <i>Person [Name, ID, Blood-Type]</i>	
	COMPLEX <i>Intrinsic topology changes</i>	
	ORDER <i>Traffic-Flow [inbound, outbound]</i> vs. <i>Traffic-Flow [outbound, inbound]</i>	
NAMING		<i>Rainfall vs. Precipitation</i>



The introduction of CDL transforms the definition of current systems. From now on, we view systems as a Domains containing a collection of domain-specific EntityObjects with Properties and MetaProperties. Links, also defined in this chapter, are meant to express the relationships between a system's Domain (local) and the (global) ontology. The Links implicitly define the relationships of a system's domain to other (local) domains.

### **In search for a definition of context**

The key issue that context interchange theory poses is that of defining and representing context at local level. With this in mind, our previous research [Neira & Madnick 93] aimed to provide a better definition of context knowledge and context representation, i. e., context definition and representation at system level.

In our study [Neira & Madnick 93], the MIT Alumni/Gift Database<sup>18</sup> was chosen as a case study for context, i. e., data environment implementation. To define the context of this database, we studied its domain, i. e., the part of the real world represented in the system, its data semantics, data access tools, i. e., data definition and data manipulation languages, its schema, i. e., how data is organized and how is accessed, and its associated implicit knowledge. As part of our methodology, we queried the database and observed that the absence of context, i. e. incomplete data environment definition, resulted in poor accessibility and misinterpretation of query results. We concluded that the database should be described as:

---

<sup>18</sup> For more information on this database, the reader is referred to Chapter 4.

- A **Domain**, which is the part of the real world represented in the system, more precisely it is the computer representation of that part of the world<sup>19</sup>.
- A finite set of **EntityObjects**, which can be thought of as the set of categories, or concepts (things), belonging to the domain of the system, e. g., *alumnus*, *company*, etc.
- A finite set of **Properties**, which can be thought of as attributes of the previous concepts, used to store information about the EntityObjects such as *CO-NAME* (company name), *IDENT-CODE* (identification of the class of the record), etc.
- A collection of (data environment) definition structures, in this case the ones defined by the existing data manager, involving a language and several data storage structures used to encode and manipulate the data.

These four modules capture the “meta” information describing the content of the system. In this particular case, *Domain*, *EntityObjects* and *Properties* constitute the declarative knowledge, and the *DBMS* is the procedural knowledge needed to manipulate the other three.

At database field level, we created a *CDL* structure representing the fields' context. This new knowledge representation structure contained, besides the

---

<sup>19</sup> In this paper, this domain is defined in more detail, and it is expressed in terms of what we called the supra-context or high level description of a network node. The supra-context is envisioned as a help facility containing part of the assumptions about data in that particular environment. It will be used by the context mediators when executing global queries.

knowledge provided by the database structures, other declarative knowledge and procedural knowledge (table look-ups, heuristics, rules, database queries, and others) stating fields' meaning, semantics, logical location, etc. of the data environment.

The lesson that we learned is that the system domain can be described as a finite collection of ontos chosen from the global ontology. That is, to describe the domain of the system as a collection of concepts, and to reference them to a global system, i. e., a pre-established set of reference concepts. This chapter, building upon those original findings, presents a development of context interchange in two directions: a more complete definition of context and an extended CDL language.

The chapter continues with a formalization of the context interchange model. Based on Set Theory, domains are expressed as sets of ordered n-tuples. Afterwards, Transformation Theory is used to formalize relations among systems and to define domain transformations when transferring data from one to another, and to resolve data conflicts (Table 5).

Our work continues with a description of the knowledge contained in the modules proposed by the model, and the definition of a CDL suitable to store that knowledge. CDL is a language for context knowledge representation and transformation, i. e., CDL contains declarative and procedural knowledge. CDL components carry the meaning of what we identified in the last chapter as the context of a system. It is also capable of expressing the local concepts as members of the global ontology. As a summary of the previous paragraphs, the work in this chapter comprises:

- A formal model for context representation and context relationships.
- New CDL frames suitable to represent declarative and procedural context knowledge.

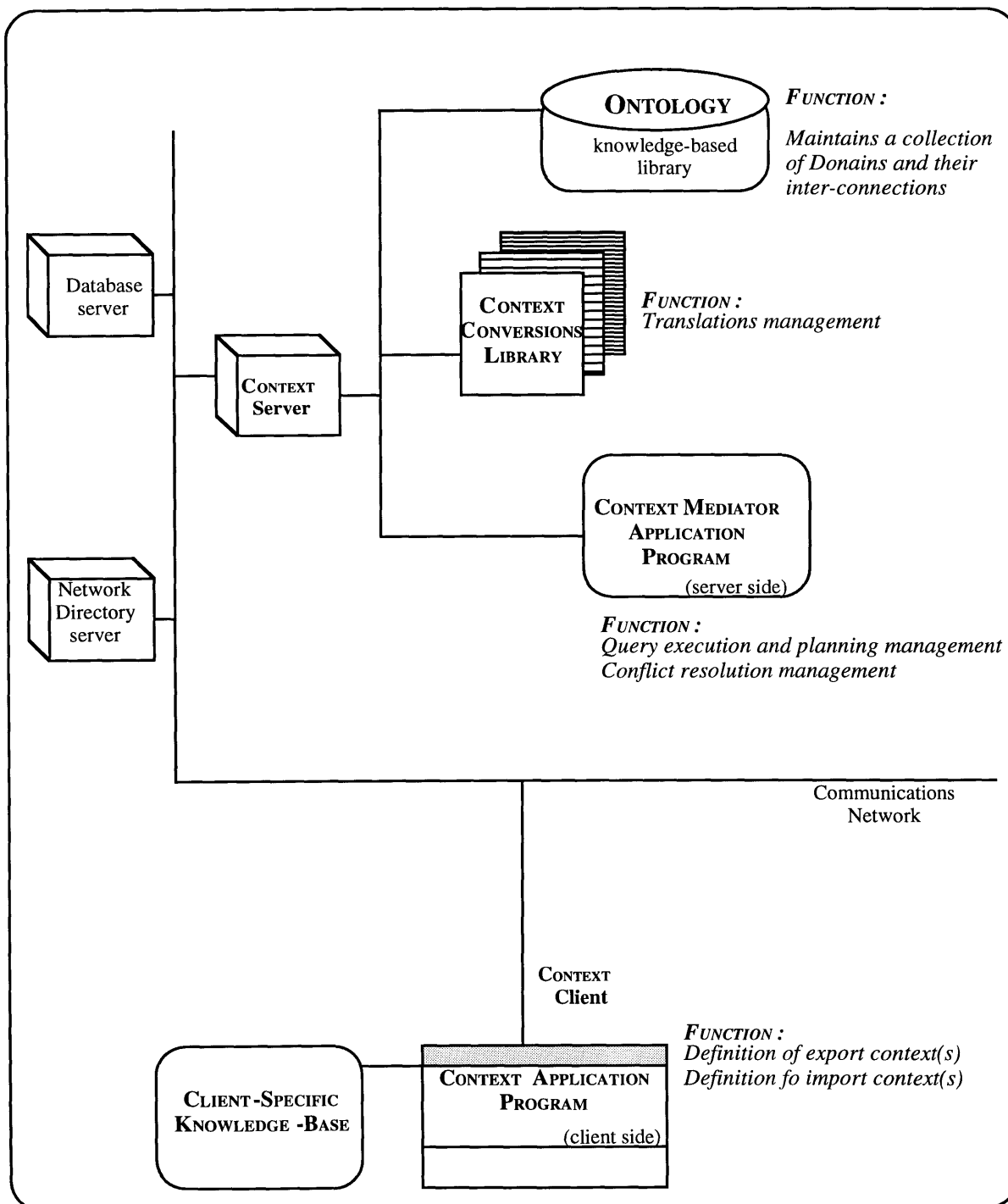


Figure 4: Client-Server Context Interchange Architecture

### **3.1 Why a new model and a new representation language for context?**

The goal of the context interchange architecture is to provide the modules for an intelligent integration of information based on context interchange paradigm. In this section we describe in detail the architecture of these modules.

First, we should point out that to be able to interconnect all existing systems, move applications, move data, etc., i. e., achieve a fully interoperable information technology infrastructure, these existing systems should open their architectures<sup>20</sup>. Open interoperability means that the new application should be built in such a way that neither the design nor the implementation of the system should block the application to a particular environment. In other words, open systems applications should be portable across all customer specified platforms.

What is the meaning of "open" regarding data? It means the interchange of meaningful data as well as the search of where that data is, exist as part of the system and as part of the network modules. Moreover, these modules create an architecture based on international (global) standards.

The architecture that we are going to define here supports these functions and, consists of a set definitions, rules and terms that are used as guidelines to the context interchange application. To guaranty cross systems portability, we define this architecture based on industry standards. This section proposes to represent

---

<sup>20</sup> The traditional way to do that is to adopt standards-based interfaces for each of the modules of the architecture.

and split context interchange tasks, and to locate them as part of an OSI-based application program.

### **3.1.1 Components of the architecture**

We view the context application program as embedded in a client-server computing environment. The Context architecture encompasses several components (Fig. 4) at local as well as global level.

At system level, the data environment --data meaning, semantics and organization-- are captured in a source context and a receiver context, and included as one more part of the system. The contents of this part of the system qualify them as the data environment's knowledge-base. In traditional information systems the most similar concept corresponds to the system schema, data types and the like analyzed in Chapter 2.

At the global network level, libraries of ontologies, context mediators and context conflict resolution modules are elements of the context (server) application infrastructure.

- Network libraries, i. e., ontology knowledge-bases, to store the concepts of the various domains of the federated systems. These ontology databases function as a meta-knowledge storage service. Their contents are made available to clients in response to their requests.
- Context mediator is in charge of query execution and planning management and it is also in charge of source location and identification.

- Context conflict resolution module. Once the high-level domain matching is done, the next step will be to perform intra-domain characteristics matching.
- Conversion libraries are also part of the global modules of the context application and they will be in charge of translation management.
- Context conversion libraries. Here some conversion libraries are available and are called by the Context Mediator generated conversion plan to undertake the necessary context transformations.

### **3.1.2 The context application logic**

So far in this section we have analyzed the components of context and how they fit. Now we shift to concentrate on how to define context as an application program that lives in the network, particularly how its components take care of the different processing steps, and the logic behind this architecture.

We specify for context an architecture based on distributed computing model, i. e., client/server model (Fig. 4). We will organize the network directories in an intelligent way, and store and add to these directories information about the various network Domains.

In a distributed computer environment, the local modules of the context application will contact the global modules to request Domain information regarding a particular query. The context interchange application takes the request and passes it as a query to a global directory, i. e., name services application module to lookup information on network node designations (names and addresses).



Anything that can be named and accessed individually (databases, application programs, etc.) is called a Domain. Each Domain has a corresponding listing (an entry) in the above defined directory service. Each entry contains the EntityObjects and the Properties that describe the Domain. All the entries are collected in lists called context directory services. The directories are organized into domain hierarchies --a directory can contain another directories. This way of organizing context directories is similar to the way in which directories of telephone users organized (by countries).

### **3.2 A formal view of context interchange<sup>21</sup>**

So far in this thesis, we have been concentrating on the study of the knowledge representation part and this is the part that we formalize in this point. Our formalism of context interchange uses Set Theory as a basis to represent any existing system --a set with elements with certain order laws and certain properties. Once we have organized the local domains, i. e., network nodes, we propose a formalism for data exchange among them based on Transformation Theory.

#### **3.2.1 Assumptions**

As we pointed earlier in this chapter, we view a large network connecting a collection of independent systems. The domains attached to each of them can be expressed as:

---

<sup>21</sup> In the mathematical expressions used in this thesis we follow Einstein's index notation where applicable.

$$D_i, i = 1, \dots, n$$

For the purpose of this thesis we will treat all of them as homogeneous although we understand that future refinement of this formalism may need to describe specialized nodes, and specialized Domains within a given system.

### **Domains as collections of concepts**

The set of general concepts (things) that the system knows of constitutes its Domain<sup>22</sup>. The Domain of a system is a set of conceptual entities (things), i. e., **EntityObjects**. This finite set of entities, expressed in the language of the shared ontology is the outer boundary of the system domain. We can refer, for example, to a person as an EntityObject defined by the system's environment. Thus each node's domain can be defined as a set union of all the EntityObjects that it contains:

$$D_i = \underline{\cup} E_{ij}, \forall i \ \& \ j = 1, \dots, m$$

Subsets of the system domain create system sub-domains, and their formal definition goes as follows:

$$(D_i)_l = \underline{\cup} E_{il}, \forall i \ \& \ 1 < m \ \& \ E_{ij} \dots E_{il}$$

That is, we also contemplate the possibility of having different contexts, or Domains, for the same system. They will be defined as context views and

---

<sup>22</sup> Stores information and/or processes information about them.

constitute well defined subsets of the general context. They can be thought of as domain projections contained in the general system Domain.

**Concepts (EntityObjects) as a collection of properties (Properties)**

Every EntityObject has a finite set of associated descriptors, i. e., **Properties**<sup>23</sup>. They are used to store information about the EntityObject. In our formal description of context, they will define the dimension of EntityObject in the domain where they are located. We consider in this thesis that the Properties are individual and single units of information<sup>24</sup>:

$$E_{ij} \in D_i, \forall i \ \& \ j= 1, \dots, m$$

$$E_{ij} = [P_{ij1}, \dots, P_{ijk}] \quad , k = 1, \dots, r \ \& \ j= 1, \dots, m, \forall i$$

Therefore, the collection of all the properties belonging to an EntityObject can be expressed as a set of n-tuples where n is the number of properties attached to the it. This set also determines the dimension of the EntityObject in  $D_i$ , its Domain. The definition of a Domain as a collection of EntityObjects with their K Properties constitutes the reference-base and dimension of the Domain space. In addition, a Property must be able to represent partial knowledge about the EntityObject and accommodate multiple sub-properties.

---

<sup>23</sup> See Footnote Number 6.

<sup>24</sup> As we will see in the next chapter, the nomenclature used to refer to properties consist of the EntityObject to which they belong, an arrow and the name of the Property:

EntityObject-->Property  
 e. g., EntityObject-->IdentificationNumberName



NOMENCLATURE	
SYMBOL	EXPLANATION
	EntityObject (global) or Property (global)
	EntityObject (local) or Property (local)
×	Cross Product
⊗	Tensor Product
⊙	Cluster Product
—	Generalized Link

Figure 5: Context Model. Graphical Nomenclature

### MetaProperties

Each EntityObject and Property can have attached to it a set of parameters, called MetaProperties whose value will determine transformations necessary to undertake when exchanging data from one domain to another:

$$P_{ijk} \in D_i \ \& \ E_{ij} \in D_i \ , \ \forall P_{ijk}$$

$$D_{ij}: P_k = P_k [\mu_s], \ s = 1, \dots, v$$

The previous formula expresses that the MetaProperties are a collection of parameters ( $\mu_S$ ) that the system needs to determine the characteristics of the Property outside the Domain boundaries. MetaProperties contain explicit declaration of the meaning and representation of that piece of knowledge.

The reader should notice that two implications of our previous definitions are as follows:

- Properties are atomic, an atomic concept is a non-decomposable. It can be physical, abstract, event, etc., but is a single piece of data element that can be regarded as a fundamental information unit whose meaning is assumed to be understood and, in the environment it resides, it needs no further definition. But the "understandability" of an atomic concept is local to the system. In data import/export processes additional information needs to be incorporated. We choose to express this additional information as MetaProperties.
- A non-atomic object is a physical, abstract, event, etc. whose meaning is described (defined) in terms of other atomic and/or non-atomic concepts. These non-atomic concepts are defined by the complex data structures provided by the system's data manager. They can be relational tables, objects, entities, relationships, etc. The additional information that we need to explain a complex data structure goes far beyond the concept of MetaProperties.

### **Constraints**

Each property is defined over a range of values that is most likely characteristic of the domain. This range of allowable values is a domain constraint:

$$E_{ij} = \text{dom} (P_k \in E_{ij}) = \{P_{ijk}\}$$

It is important to notice that once we have identified the EntityObject owner of the property, to complete its definition, in addition to providing the above mentioned MetaProperty values, we must provide the range of allowable values of that Property within the system.

### 3.2.2 Relations

Each system can communicate (receive and/or send information), bi-directionally (source and/or receiver), with other system nodes. Even though there are no constraints built into the model, depending on the characteristics of communicating domains, the process of data exchange among two given nodes need not be symmetric but dependent of the direction of the exchange.

#### Links

When exchanging information, we first have to assert whether or not the Domains involved contain common EntityObjects and common Properties. This is what Links are used for.

Domains can communicate by attaching to them a set of global relations, called Links, that glue each domain to a reference vocabulary system called the global ontology. By using this ontology, Links express sharing of nodes and provide the capability to refer to the same clusters across Domains. They constitute the most important tool of context abstraction, generalization and conceptualization.

In the context model (Fig. 4), the relationships among global Domains are also expressed, indirectly, using Links. In our context interchange formalism, Links

consist of a collection of pair-wise operations across Domains (Fig. 6). For example (Fig. 6), given the collection of EntityObjects belonging to Domain "A" ( $E_{ij}$ ) and Domain "B" ( $E'_{ij}$ ), we can express a set of pre-established relations (already present in the global ontology) to other Domains. In addition, we can create new relationships by using the concept of transformations that we introduce in Section 3.2.3.

Again, the key issue is to realize that domains can be modeled by linking EntityObjects, Properties and MetaProperties to a network of interrelated and predefined concepts, i. e., the **shared ontology**, and that the type of Link varies depending on the Domain.

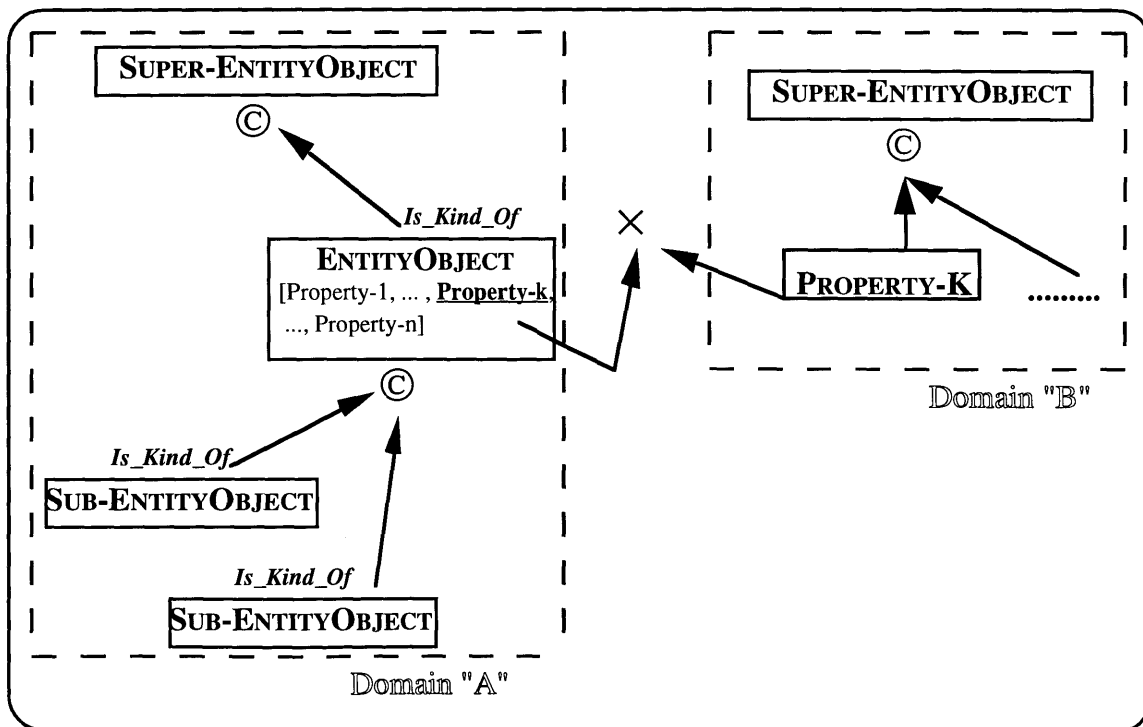


Figure 6: Simplified View of Context Interchange Model

### 3.2.3 Transformations in context

Links can also allow the domains to be transformed by different combination operations. We give the definition of the following ones:

- **Cross product** [  $\times$  ] is used to represent the connection of two different hierarchies. A cross product Link indicates a Cartesian product of the component domains. For example, the cross product of the domains shown in Figure 6 would be:

$$E_{ABki} \in D_A \times D_B$$
$$E_{ABki} = [P_{A1}, \dots, P_{Ak}] \times [P_{Bi}] \text{ ,, } i = 1, \dots, v$$

- **Cluster product** [  $\odot$  ] is used to express a hierarchical relation among entities and/or domains. These type of Links are suitable to represent the classical hierarchies. To express the cross product of two nodes in knowledge representation terms we will include an Is\_Kind\_Of Link to express this parent/son connection.
- **Tensor product** [  $\otimes$  ] will be used when there is a need to express all possible combinations of the elements belonging to two different EntityObjects.

#### Model's support of context comparisons

The above defined context transformations and relations facilitate the exchange operations among heterogeneous environments. Besides transformations, context interchange also contemplates semantic transformations. They are needed when representation conflicts are present. Remember from the previous chapter that transformation management is one of the responsibilities of the context mediator.



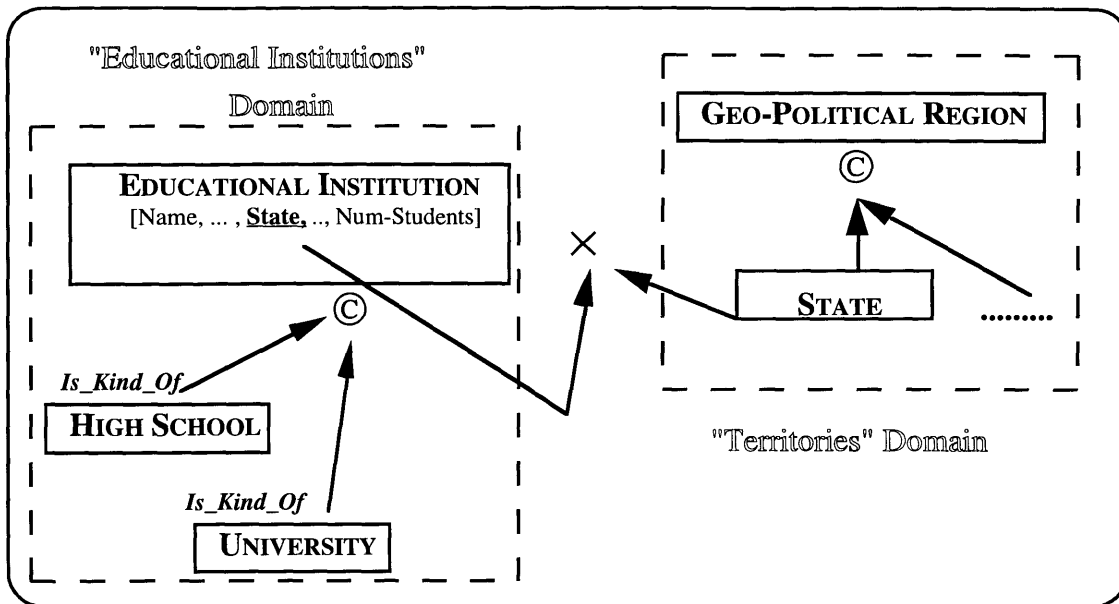


Figure 7: Application of Context Interchange Model

### 3.2.4 The ontology as facilitator of data exchanges

Recall that context interchange theory involves the definition of a shared ontology used as inter-systems common vocabulary, and that in our previous research we propose to structure the domain as multiple interconnected contexts. In [Madnick & Neira 93], we presented an example consisting of the development for a generic EntityObject, and a set of sub-ontologies needed to contextualize its Properties. Although the purpose of this paper is not to build an ontology, we must create the parts of it that we need to describe the domains of our examples. In this case, the names for EntityObjects and their corresponding Properties were chosen arbitrarily but assuming they exist as part of an existing global ontology used as inter-systems' common metadata vocabulary.

Organizing this ontological world model involves an extensive multiply interconnected network of units, for which storage, access and update procedures are available, as part of the inter systems connection architecture. Examples of this shared ontology and their use in the definition and access to data environments will be provided in Chapters 4 and 5.

### **3.3 Context representation: CDL**

The model that we built in the last section has defined Domains, EntityObjects, Properties, Links and MetaProperties as the building blocks of data environment definition. In this section, we are going to explore the internal structure and the contents of each of them.

From our point of view, the definition of data environments using context interchange theory encompasses two different representation tasks:

- **Knowledge representation** of the system's context that involves:
  - Representation of the system's declarative knowledge. It includes schema knowledge and data semantics.
  - Encoding system's procedural knowledge. It includes access languages and methods to the data contained in the system
- **Knowledge management and knowledge processing**, namely context translations, context mediation and context information search.

Particularly, we are going to present CDL definitions at different levels and allocate them to different parts of the network. This involves the design of context knowledge representation structures:

- Individual system representation as an export context.
- Import context in the form of a global query.
- To define the data structures of the system.

Our first step is to provide a global representation of content definitions (or domain definitions), and later, complete it by adding a definition of its syntax and its semantics and access methods. Once we have completed this task, we will have the building blocks and a reference system for the information existent in the local system.

For all the proposed CDL structures, we need to give a modular definition, flexible enough to present their potential users with the part of the data context that he/she needs, rather than with a standard unit that cannot be accommodated to represent the particulars of every system. In order to do so, we propose to design CDLs in separate modules, corresponding to orthogonal dimensions of data environments and to allow unit modules and their dimensions to be customized at the proper level.

### **3.3.1 CDL characteristics**

We choose to build CDL with a frame-like language structure. According to several authors [Devanbu & others 91, Minsky 68, Patel-Schneider & others 94] these type of representations are suited to describe sets of objects with complex relational structure, and it is also suited to represent domains that exhibit strongly hierarchical, taxonomic categories of objects.

A frame structure [Brachman & Levesque 85, Hu 89, Minsky 68, Norvig 92] provides a set of representation services for complex data collections and/or databases. Frames have the power to combine the various ways of representing knowledge (objects, rules, logic, etc.) and can therefore combine the representation advantages of all of them.

One very important piece of information that the CDL frames must capture is their associations with the global ontology. CDL must have the capability to interconnect hierarchies by supporting various types of links [Madnick & Neira 93] therefore, extending the semantics of a pure hierarchical structure, as defined in the previous section. What we are proposing is to link CDL frames so that the result is a network where, of course, the nodes of the network are the new CDL frames.

### **3.3.2 Domains in CDL**

This section describes the way in which we choose to name and write the information contained in the CDL frames, their components, Domains, MetaProperties, etc.

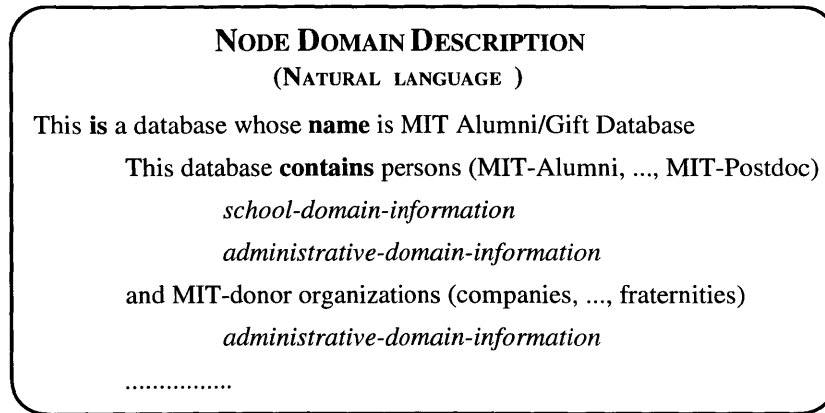


Figure 8: Network Domain Description in Natural Language

### 3.3.2.1 Source context domain definition

We will define the information present in the system as well as its characteristics. This can be done by giving a high level organized description of the system. Figure 8 presents a natural language version of what we just called high level description of the Alumni/Gift Database's contents. In addition we have to include also a high level description of the data manager, i. e., access means and organization of that content data. A suitable syntax to capture the above domain definition is the Node Domain description presented in Figure 9 that represents the MIT Alumni/Gift Database. In general, we defined a set of standard components as follows:

#### Context\_Name

It contains declarative knowledge stating the name of the node. It is the unique name, a network address or a logical name. It plays the role of the node's unique identifier. In our example (Fig. 10) we chose to fill the logical name of the agent, i. e., MIT Alumni/Gift in this field.

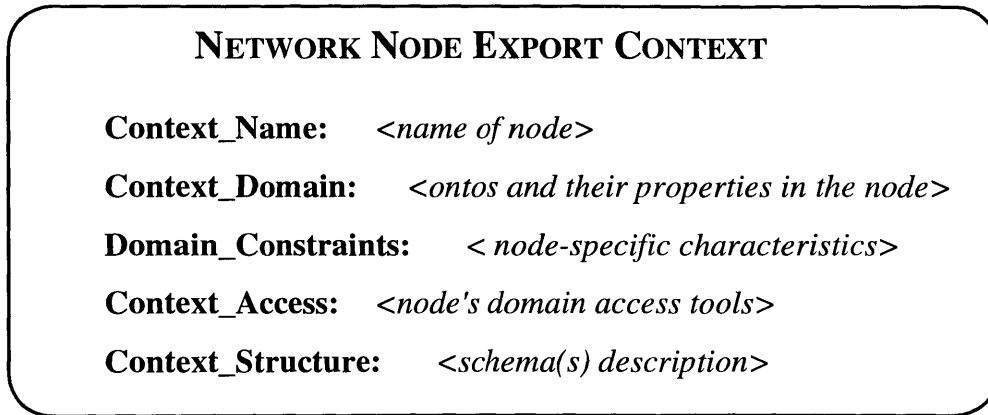


Figure 9: Network Domain Description in CDL

### Context\_Domain

It contains declarative knowledge stating the collection of EntityObjects and the list of Properties present in that domain. Therefore, it contains (Fig. 9) implicitly the data available, i. e., domain of the system, expressed in terms of the global ontology. Partitioning the domain and structuring the domain will lead to the definition of a hierarchy of context and a hierarchy of their associated knowledge bases. Furthermore, the establishment of these context hierarchies is the essential part when performing context comparisons and network-wide domain searches.

### Domain\_Constraints

It contains declarative constraint knowledge specific of this node. Usually, a phenomenon has certain characteristics that distinguish it from others and data environments are no exception to this rule. It specifies a set of conditions (constraints) that apply only to the data present in the node. They are organized into logical groups as follows:

- Time constraints
- Space constraints.

- Quality constraints.

The representation of this constraint knowledge is only necessary to access this system. It is not, by any means, universal domain (shared information) and therefore it should be kept at local level.

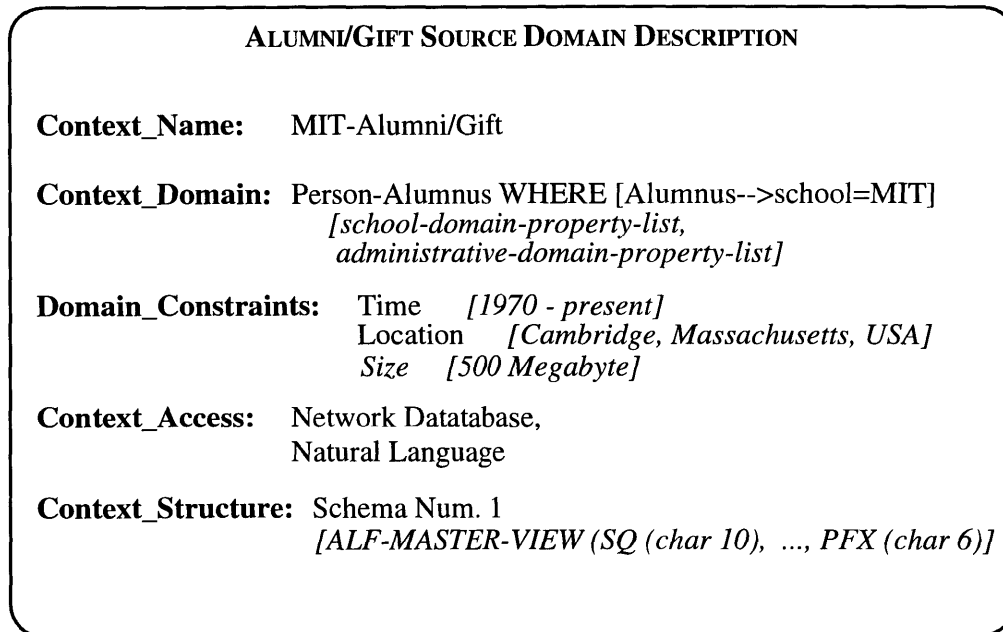
One common characteristic of this knowledge is its declarative nature. For example, most of the existing systems, i. e., databases contain the part of the world that is not supposed to change in declarative manner in the form of the well known integrity constraints. We call this type of knowledge constraint knowledge. The representation of this kind of knowledge [Brachman & Levesque 85, Minsky 68] is an advantage in the sense that search processes can use them to improve performance. The disadvantage is their static quality and inflexibility. This knowledge is used for detecting or correcting errors in input data. In the case of the Alumni Database we wrote its description in Figure 10.

The maintenance and accuracy of this knowledge is very tedious. Therefore, we will only represent those constraints that have a direct relation to the interoperability architecture, keeping local the rest of them. There is no need to expose every single system's complexity, otherwise the context application, and particularly the global ontology would become unmanageable.

### **Context\_Access**

It contains the procedural knowledge necessary to manipulate the node's data. The characteristics describing them, recorded here in the form of a list of predicates from a family of related characteristics or what we have called domain

characterization. This slot expresses the general constraints on all the members of the class that it belongs to (declared in Context\_Domain).



**Figure 10: MIT Alumni/Gift Database Description**

### **Context\_Structure**

It contains the schema(s) present in the system. It also contains references to the protocols, internal data organization, etc. more specifically it will contain the information that would allow using data manipulation languages to query the system. The information will point to the system's schema storage facility. The description of the structure of the agent may not be unique, i. e., a particular node can choose to be represented under different views.

#### **3.3.2.2 Receiver context and domain-restricted queries**

When a node is acting as a receiver it will do so by presenting, in the form of query-values the information that it is requesting. For example, from the same Alumni/Gift Database we could submit a global query asking for sources where



MIT Alumni information is stored as shown in Figure 11. For example, suppose that we are now located at the MIT-Alumni/Gift Database domain and we want to find out what other systems contain information on MIT alumni, and what information is available. Then, we will submit to context server residing in the network the query present in Figure 11.

Therefore our approach to the receiver context design will be to use the same structures that we designed for source context, inserting query-values in those properties, fields, etc. that we attempt to query. The implicit assumption in our design is that a domain can only submit queries to the global ontology within its own domain.

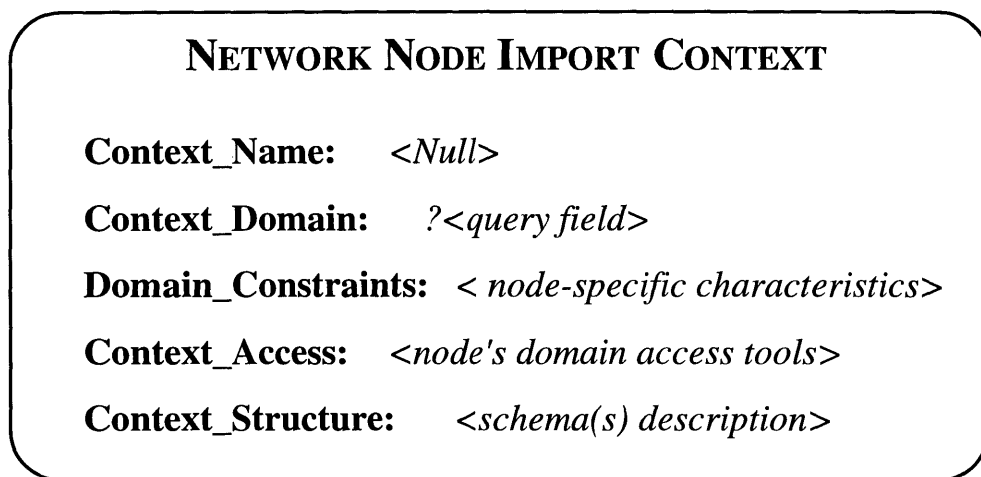


Figure 11: Receiver Context for Querying on Context Domain

However, we are going to allow context-based querying with no pre-defined Domain. That is the case, for example, of a user who wants to get information for the available network sources with no domain restriction, he or she will have to build dynamically a CDL frame instance similar to the one pictured in Figure 9.

The context server also provides execution management facilities for this kind of context-free queries.

### **3.3.3 EntityObjects, Properties and their semantics**

We will define here the content and form of the CDL language for the purpose of creating a context information data storage structure suitable to hold the information required to represent its context. The description of its component modules is the subject of the rest of this section.

We propose to describe the context of a system in terms of the different dimensions of data description. Furthermore, we claim that the components of this CDL property frame (Fig. 12) can be expressed as orthogonal dimensions of data description and presented to the user, if necessary, independently. At the same time, every dimension can have several meta-dimensions. Madnick and Siegel [Siegel & Madnick 89a, Siegel & Madnick 89b, Siegel & Madnick 91, Siegel & Madnick 92] have described them as semantic domains of the data. These semantic domains constitute the variables whose values are to be determined before any meaningful data exchange can be undertaken.

In the next sections, we describe the contents of the layers and their sub-components.

#### **3.3.3.4.1 Identification**

It tells the location (schema location) of a particular piece of information (entity-object) in the system schema. It also provides the name of the onto (thing) in the system.

We separate this part of the entity-object knowledge from the rest in order to differentiate the structure and usage of names from the structure of the EntityObject itself. This module is directly related to current repository technology in the sense that it can be used to facilitate data location across any portfolio of systems. The layer is subdivided into:

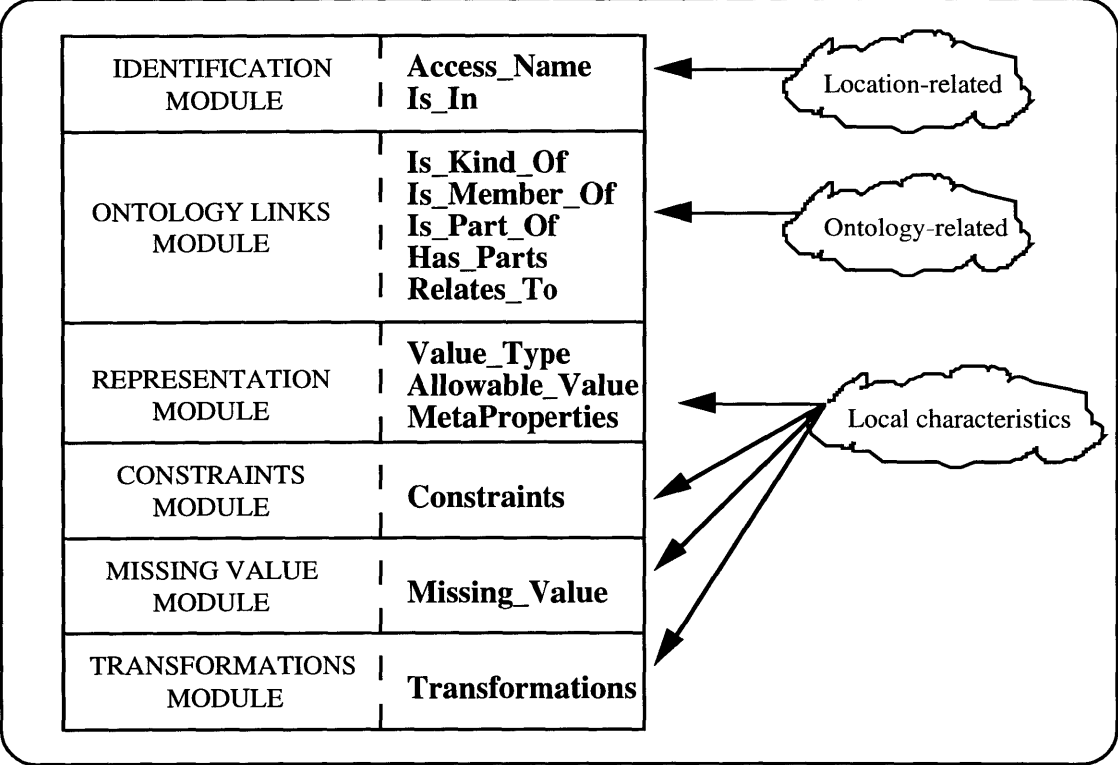


Figure 12: Context Definition Language Layered Structure

**Access\_Name**

It contains the Property’s name in the system. We could have broken this item into further detailed items, giving version-identifier, descriptive-name, alternate-name, etc., but, for the purpose of this paper, and for the sake of simplicity, we keep the context of a Property’s name an/or the context of an EntityObject’s Name as if it were fully defined by its Access\_Name. This field can be multi-

valued if, as we will see shortly in several examples, the system contains name duplications. Replicated data<sup>25</sup> enhances security and availability and our CDL should accommodate the representation of such a critical issue.

### **Is\_In**

It stores the logical location(s) of that property in the local schema. The examples shown later in this thesis are given in the form of a chain connected by dots with the name of the system followed by other location identifiers e. g., ALUMNI/GIFT.ALF-MASTER.PFX-CODE. Notice that, within a system, there may exist different sources of the same information.

This notion of location identifier can address data reduplication by including in the value of Is\_In the many locations where the same information is stored. It is important to have such a scheme because it gives our architecture power to cope with issues as different as data security reasons, poor systems design, etc., this field can be multi-valued if there is data duplication. In the case where additional information can be found in the same system, we store that information in the Relates\_To module but not here.

---

### **<sup>25</sup> Note on Access\_Name and data replication**

In the Alumni/Gift Database every field can be queried under two different field names or, using CDL Unit nomenclature every field has two different access names:

<i>company</i>	CO- NUM and BN
<i>identification number</i>	SEQ-NUM and SQ
<i>mailing name</i>	MAIL-NAME and AN, etc.

### 3.3.3.2 Ontology Links

Ontology Links are meant to connect the local Domain of a system to the global Domain residing in the context interchange server, i. e., the library of ontologies (Fig. 4). Links express the EntityObject, Property, etc. relative to a pre-existing global ontology. That is, the local constructs are matched, through this scheme, against an inter systems ontology. This ontology provides a uniform definition of the EntityObjects, MetaProperties and Properties that become the building blocks to describe the system, more importantly, the information content of the system, its domain, is defined by these set of Links rather than in terms of physically stored data.

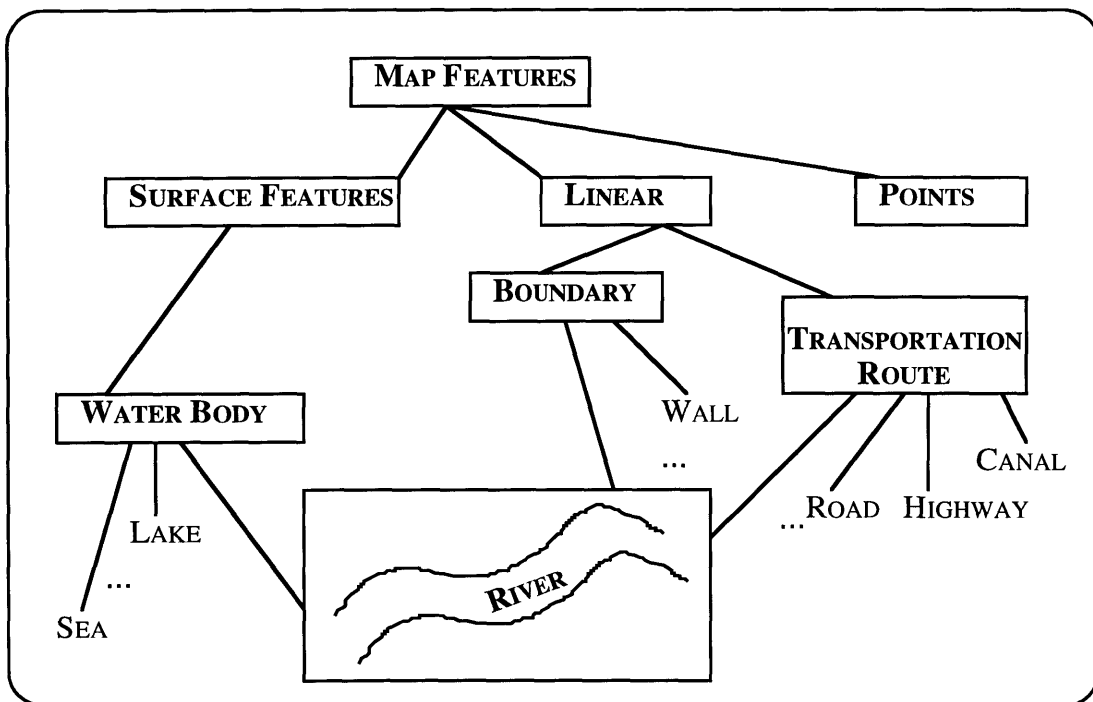


Figure 13: Interlocking of Hierarchies in the Shared Ontology

Conventional hierarchies, i. e., tree structure, are inadequate for our purposes. A river, for example, may simultaneously belong to the transport route, national

boundary drainage channel and water body classes. The taxonomic result is an interlocking of hierarchies (Fig. 13) known as network, or lattice<sup>26</sup>.

Entity-objects, properties and MetaProperties can be linked using types, tables, rules, etc. The rest of this section is dedicated to define the different types of links that an object or property needs to get connected to the global ontology.

Given the appropriate transformations, this linked ontology should be able to support aggregations, decomposition, heterogeneous formats, and organize and resolve data incompatibilities.

One important thing to notice is that we envision these links not only as residing in the global ontology, but also as residing and being part of the knowledge-based ontology that the local system needs to support its context definition in terms of global concepts.

The semantics of this ontology are supported by the definition of the following types of specialized links:

### **Is\_Kind\_Of**

Represents a hierarchical relationship (super-class/sub-class) in the ontological hierarchy. It is the classical tool used to construct generalized hierarchies.

This classification scheme is one of the reasoning methods that our links use, i. e., the property of an EntityObject in the super-class will be passed down to its sons

---

<sup>26</sup> A lattice allows a node to have more than one parent.

if no restriction is specified. An example of *Is\_Kind\_Of* Link is presented in Figure 14.

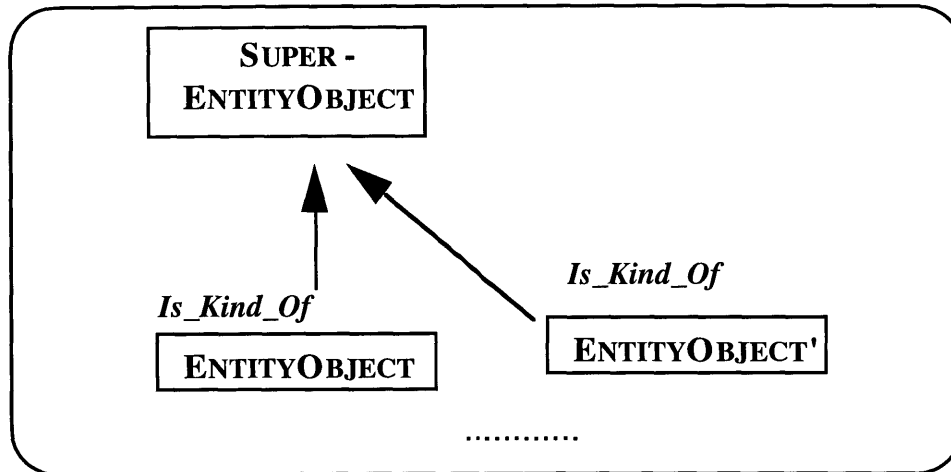


Figure 14: *Is\_Kind\_Of* Link

### ***Is\_Member\_Of***

It is a link that expresses membership to a particular class of EntityObjects. Properties or MetaProperties. This is the mechanism that we use to express membership of one of the local data categories to one of the members of our global ontology.

### ***Is\_Part\_Of***

This Link is meant to express the different levels of granularity in which an item (thing) can be represented. It imposes a relationship between a low level object and a high level object ,but not as a subclass/super-class relationship (the later case is covered by the previous Link). The importance of the semantics of this link lies not only at the global ontology level, but also at the local level. It expresses the local grouping of global ontos (things) under a single-thing; for example, a MAIL-NAME (local grouping of things) containing the name, the

prefix name, the suffix name and an address. This concept is similar to the one that the IRDS terminology defines as ATTRIBUTE-GROUP. In other words, this is the abstraction concept of building aggregate objects from component objects. In this way, we can construct a new EntityObject from existing ones (Fig. 15) for a given domain.

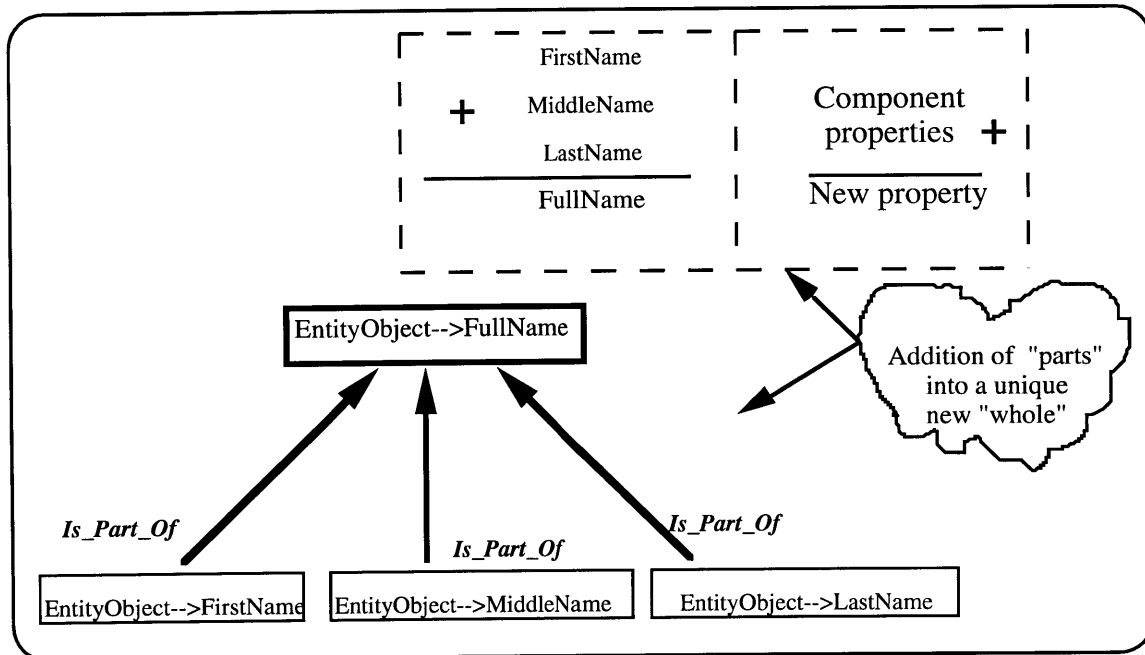


Figure 15: *Is\_Part\_Of* Link

### **Relates\_To**

This Link expresses connections with other items, even connections outside the exact domain of the system. We need this link (Fig. 16) because the category of a concept is completely determined by its environment, rather than by its intrinsic composition; that is, the category of a thing (essence) and its uses (roles), differ among systems. This fact reflects that the purposes of the system (use, role) determine what how the concept is perceived. For example, in the Alumni/Gift environment, companies and college students are perceived as the same thing



(ALF-MASTER File records), and therefore both have FIRST-NAME and LAST-NAME.

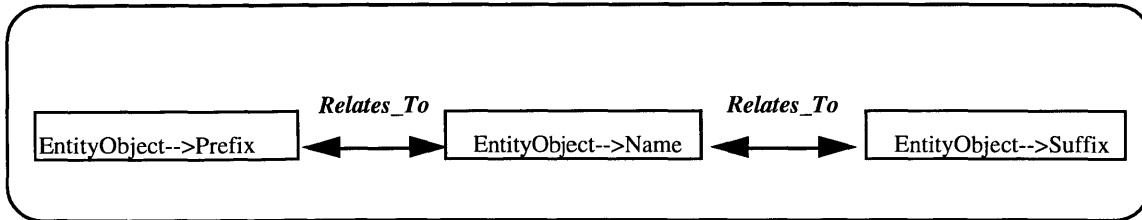


Figure 16: **Relates\_To** Link

In other cases, we need this Link to represent categories that overlap or inter-relate without being subsets. This is a tool that involves classifying similar objects into object classes. Namely, it allows us to define types of names for a given thing, e.g., person-name, legal-name, Soc-Sec-Num, etc., and to enable different access schemes for data. The following example is taken from one of the systems that we study in Chapter 4. It shows us the purpose of this type of Link: a person (that we chose to name as EntityObject) can be named in several ways, or in the nomenclature that we are introducing, a person can be referred to in many different ways:

- With an identification number called SEQ-NUM that we have referred to in the global ontology as:

EntityObject-->IdentificationNumberName.

- With the category name (Alum, Widow, Postdoc, etc.) to which it belongs. We have referred to in the global ontology as:

EntityObject-->ClassName.

- There are other types of names (Legal Name, Full Name, etc.) in the system and we have grouped all of them under Name. This scheme give us the flexibility to create subclasses for it within the domain of the system<sup>27</sup> :

EntityObject-->Name

Notice that this Link can also be used when different representations of the same property are present in the system, and most likely not all of them exist at the global level. Most of these representations are the local variations under which a concept or property can be expressed in a data environment. EntityObjects and/or properties connected by this link are characterized by having a common Is\_Kind\_Of Link value.

### **Has\_Parts<sup>28</sup>**

At the local level, it will allow us to refer to a set of different (local) things as a single (global) thing. This link will support decomposition of Properties and/or EntityObjects. Let's take again the example that we chose for the link Relates\_To. The Properties Suffix\_Name Name and Prefix\_Name cannot be linked to the Name Property with the above defined Is\_Part\_Of Link. In this case, we need to express a relationship among different domains, one is the Name domain and the other is the domain of particles to add to Name.

---

<sup>27</sup>To see this sub-classification, go to diagram at the end of this document.

<sup>28</sup> Has\_Parts and Is\_Part\_Of are not inverse properties and the next link (Relates\_To) and our example demonstrates why. Essentially Has\_Parts creates a strict relationship where the parts are equal to the whole while the Is\_Part\_Of does not fulfill that requirement.

### **3.3.3.3 Representation**

It contains explicit declarations of the environmental constraints imposed on the semantic and syntactic data values, representation semantics and syntax of the entity object in the system. All the semantic values are defined at this level. An example of this would be the representation of a Social Security Number (SNN) expressed as:

*SNN [ 0 ... 9, -] (integers and hyphen, or numbers in the 999-99-999 format)*

The representation is broken into:

#### **Value\_Type**

This expresses the syntax of the entity-objects and the properties in the system; we extend the traditional concept of type by adding to it definitions of format, units, scale and length;

#### **Allowable\_Values**

It contains the set of permitted values. They can be either single-valued or multi-valued. In addition, the special value Null (undefined) is present in every domain; It corresponds to the values that the attribute can take in this Domain.

#### **Cardinality**

This expresses whether the field is single or multi-valued. In the later case, the cardinality value is the number of single items in which the field can be broken (Fig. 17)

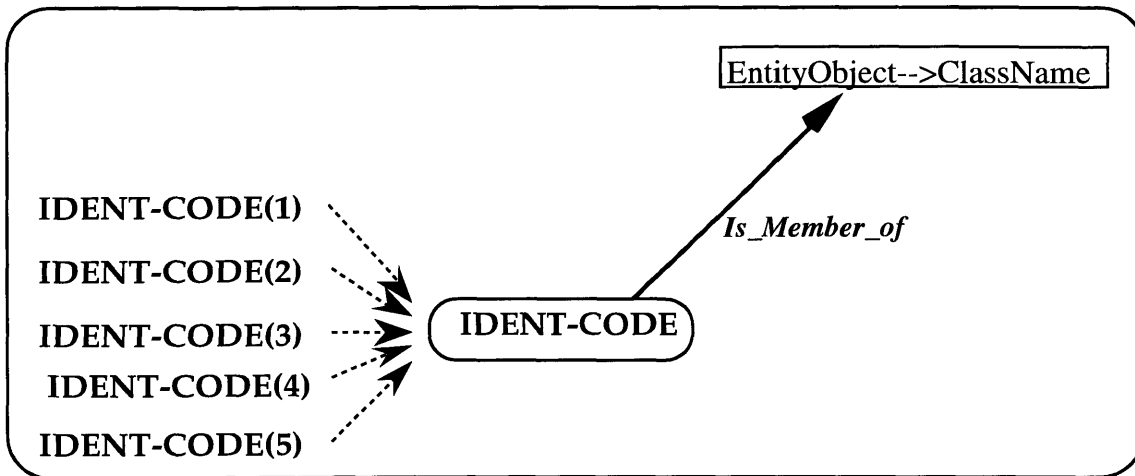


Figure 17: Example of Multi-Cardinality

### MetaProperties

MetaProperties are information needed to complete the definition of a property. They can be Properties and/or EntityObjects from outside or inside the domain of the system. They are attached to the local ontos to complete their representation definition. This definition implicitly creates two local categories of Properties and EntityObjects:

- **Primitives**

They have an intrinsic natural existence, e. g., color, number, distance, etc. All primitives have a Null list of MetaProperties.

- **Assignables**

They need additional knowledge, i. e., MetaProperties, to be described; Transformations<sup>29</sup> are used to determine MetaProperty values. For those

---

<sup>29</sup> See Section 3.3.4.7.

already present in the system, queries will be the mapping functions used to determine their values. For those not present in the system, other tools such as heuristics, rules, arithmetic operations, table look-up, etc. will be used.

Notice that MetaProperties are domain-dependent. They are not intrinsic to the data but are intrinsic to the context of the data, i. e. data environment.

#### **3.3.4.4 Constraints module**

This contains system structure related definitions. For example, primary key specifications, data access restrictions, etc. These constraints can limit the information to be accessed by defining database views.

#### **3.3.3.5 Missing value**

This is an error-handling scheme that will lookup other fields if no data is found in a CDL-based query. For example, it will dictate what to do in the case of incomplete data; that is, It describe other sources either inside the system or outside the system<sup>30</sup>, from which to infer the property value.

#### **3.3.3.6 Transformations**

It stores rules, queries, heuristics, etc. that will take the system schema to different contextualized stages. The transformations are application-dependent, and there are no general rules as to how they function. Every application, system, database, etc. will have to define their own transformations. To transfer information to a different domain, the use of this facility is very likely to be needed.

---

<sup>30</sup> A Source can be, for example, the user who submitted the query.

In this thesis, we lack a target context so we present a small set of absolute conversions (from local to hypothetical global properties) that are likely to be needed. However, we do not formally address the issues of context comparison and context conversion. These transformations, if included as part of the system's context definition, could function as a general description of the environment without the need for either conversions or comparisons of context.

## CHAPTER 4

### CONTEXT ACQUISITION USING CDL

The process of context acquisition is the subject of this chapter. We focus on the development and implementation of context in legacy systems, i. e., operational systems with a mature data environment. This is the first step needed to incorporate them to the context interchange architecture. We plan to perform context acquisition and context implementation in disparate data environments, i. e., environments with different domain and different supporting system structure. This will demonstrate the capabilities of CDL as a general, complete and non domain-specific tool to represent context.

For all the analyzed systems, we perform the same sequence of steps:

- First, we describe the structure, meaning and organization of the system and its data. We need to perform this initial study to define and understand an existing Domain and its associated data environment. At the same time, we also describe the means by which that information can be transformed in context.
- Second, we match the above defined context to the appropriate part of the shared ontology. In order to do that, we develop that part of the ontology (global Domain as well as local Domain), that we need to represent each of the systems.

- Our third, and last step, consists of encoding the system using the CDL language presented in Chapter 3. This last step also includes a study of what tools are readily available to extract data environment characteristics, and how the data environment representation scheme supports the explicit declaration of these characteristics by means of data definition languages, rules, heuristics, translation tables, metadata, etc. In other words, we will be looking at what is necessary to transform a system schema into a contextualized one.

## 4.2 The network directory locker

The network directory locker is part of the MIT network of lockers. It contains information about the Internet, the networks connected to it, how to use the Internet, how to use the directory, a network addresses (domains) database, etc.

### 4.1.1 Domain

Among the documents (Fig. 19) contained in the directory<sup>31</sup> are:

- The network directory produced at the University of Texas, which lives in `/mit/net-directory/net-dir`. It contains six parts: `net.directory.part[123456]`. Examples of records in this database can be seen in Figure 18. There is no further information in the database stating what the subfield meanings are<sup>32</sup>, why are they organized in that way, etc. That is, we found that there is no

---

<sup>31</sup> For reasons of brevity, we limit our description of directories to the most important ones.

<sup>32</sup> The subfields in this database file are separated by semi-colom (Fig. 18).



data semantics definition support in this database. The reader will notice that this is a constant in the systems we analyze.

- Regularly-produced network periodicals which describe changes in network usage and topology, new services, etc. are kept in /mit/net-directory/magazines.
- Information about network domain names is kept in /mit/net-directory/domain-info.
- A report prepared by the Rand Corporation which describes some issues concerning the Ethics and Etiquette of Electronic Mail, which is contained in /mit/net-directory/documents/email.ethics.

**EMDCCI11**; EEARN; 1267; IBM 4341-2; IBM VM/SP R4; RSCSV2/NETDATA; IBM Scientific Center Madrid, Spain, Centro Cientifico UAM-IBM, Universidad Autonoma Facultad de Ciencias, Modulo C-XVI, E-28040 Madrid (Spain); Gonzalo Martinez, GONZALO@EMDCCI11, +34 1 7342162; EARN

**EMDCSIC1**; no aliases; 2360; CYBER 180/855; NOS 2-5-3; NJEF/PUNCH; Consejo Superior de Investigaciones Cientificas CSIC, Centro de Calculo, CSIC, Pinar 19, E-28006 Madrid - Spain; Victor Castelo, CCEVC28@EMDCSIC1, +34 1 2616688; EARN

**EMDUPM11**; no aliases; 0266; IBM 4341-12; IBM VM/SP R4; RSCSV2/NETDATA; Universidad Politecnica Madrid, Spain, Universidad Politecnica de Madrid, Centro de Calculo, Avda Ramiro de Maeztu s/n, E-28040 Madrid (Spain); Carlos Otermin, U2301004@EMDUPM11, +34 1 2545000 ext397; EARN

Figure 18: Examples of Network Directory Records

- A standard host-table for the Internet is kept in /mit/net-directory/host-tables/hosts.txt. In the past, this table was updated once every two weeks or

so but it is no longer used; now the hosts are known via the domain name system.

- A standard host-table-type file for BITNET is kept in /mit/net-directory/host-tables/bitnet.links; this is updated whenever the BITNIC distributes a new version of bitnet.links.
- A directory called bin for programs to help with the use of these documents, and a directory "etc" which contains data files used by the programs residing in the bin directory.

As described, all the data is stored in a hierarchy of flat files. The information they contain can be accessed using the search programs offered by the UNIX operating system plus a set of additional customized programs to run some queries related to the directory where information on network domains and their owners is stored.

#### **4.1.2 Sub-domains**

In this case, by design choice, we consider that every file of the directory hierarchy that we are examining constitutes a unique sub-domain with (many) similar types of records (Fig. 18). This is also true in the case of a text or unstructured file.

#### **4.1.3 Context Interchange View**

In Figure 19 we picture a description of the context source description (expressed as a global sub-Domain) for the network directory of this section. In order to do so, we have assumed that the EntityObjects "Network" (and its name: Network--

>Name) and Computer\_Networks are part of the preexisting global ontology. In addition, we have also assumed that Computer\_Networks can have a set of network domains called documentation-domain-property-list, and that Computer\_Networks can have attached to them a list of users with Properties called User-->Name-domain-property-list.

	NETWORK DIRECTORY SOURCE DOMAIN DESCRIPTION
<b>Context_Name:</b>	Network-Directory
<b>Context_Domain:</b>	Computer_Networks WHERE [Networl-->Name] =Internet] <i>[documantation-domain-property-list,  User--&gt;Name-domain-property-list]</i>
<b>Domain_Constraints:</b>	Time <i>[1989-1992]</i> Location <i>[Cambridge, Massachusetts, USA]</i> Size <i>[Null]</i>
<b>Context_Access:</b>	File Hierachy, UNIX
<b>Context_Structure:</b>	Schema Num. 1 <i>[net-directory  bin  documents(mail-ethics, FTP-Directory.txt, Telemail-Gatway.txt)  net-dir (net.directory.part1, ..., net.directory.part6) ]</i>

Figure 19: Network Directory Source Domain Description

## 4.2 The Alumni/Gift database

### 4.2.1 Domain

The MIT Alumni/Gift Database stores records of MIT Alumni and MIT donors (person, corporation, society, etc.). Data is organized by a network database

management system into approximately forty (40) files<sup>33</sup>. A file has several either single-value or multi-value fields. Files are defined by their names and their fields while fields are defined by their names and their number of characters (numeric or alphanumeric).

Figure 20 shows the database schema; this is the only information about its domain and its data available to someone accessing this database. The lack of any further explicit description of the data under these files and fields would allow one, for example, to submit a query requesting all the records whose LAST-NAME starts with I, to retrieve the MIT alumni with surnames starting with I, and obtain, among others, I.B.M. as one of those assumed to be MIT alumni.

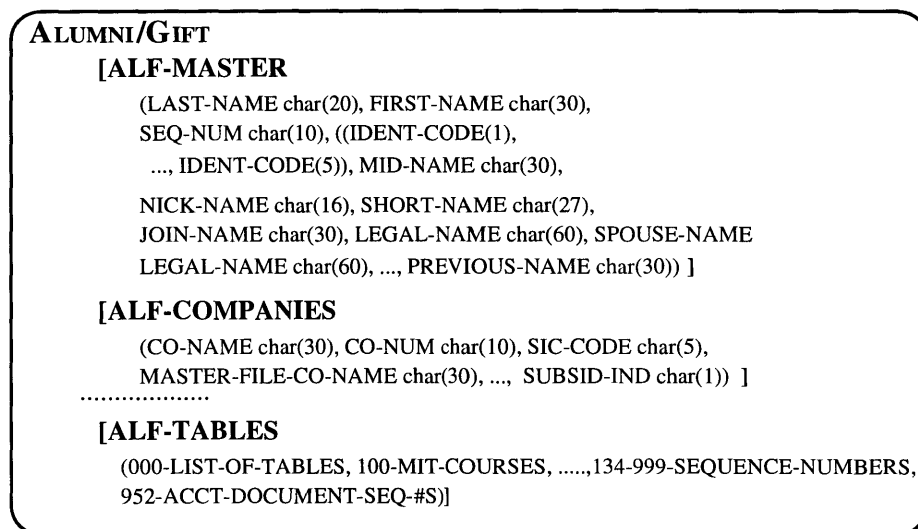


Figure 20: MIT Alumni/Gift Database Schema

---

<sup>33</sup> We access the Alumni/Gift through the CISL's Account. This account has access to three files: ALF-MASTER (restricted access through the ALF-MASTER-VIEW), ALF-COMPANIES and ALF-TABLES. The ALF-MASTER File is the largest file in the database, containing approximately 65% of the data.

Obviously, our last example demonstrates that Alumni/Gift does not provide a complete description of the database environment. In particular, the database stores two types of information whose specification is missing in the previous example:

- Administrative information on MIT alumni.
- Administrative information on MIT donors (companies, associations, MIT alumni, etc.); in the same file, ALF-MASTER.

The database allows the exclusive retrieval of alumni information but only if the IDENT-CODE and/or SEQ-NUM fields needed to infer the type of a record are in place. Yet this information is not explicitly present in the database.

The only concepts and their attributes, EntityObjects and Properties in the CDL's nomenclature , that can be identified at schema level are the forty categories corresponding to the forty files into which the database is structured.

Therefore, classifying the ALF-MASTER File's records involves incorporating knowledge into this basic structure, declaring that IDENT-CODE and SEQ-NUM contain the information needed to classify the ALF-MASTER File's records into categories. Furthermore, we can also extend these local categories, by performing transformations and semantic equivalence operations, and matching them to others belonging to a predefined global ontology.

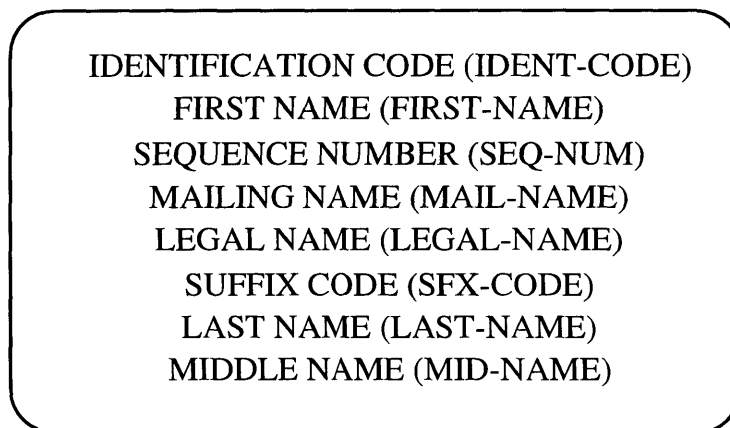
In general, we have seen that the system is missing explicit knowledge declarations about the organization, the meaning and the representation of its

data. This paper proposes to include all that information, named context, as an active part of the system.

#### 4.2.2 Sub-domains

Now we analyze and represent selected fields (Fig. 21) from the ALF-MASTER File. The analysis of each field is broken into:

- **Meaning:** contains the meaning of the information under the field.
- **Representation:** contains the definition of the semantic and syntactic constraints imposed by the environment.
- **Comments:** contains further details of the field's characteristics and its relationship to other database fields.



)

Figure 21: List of Fields Selected for Analysis

### **Identification Code (IDENT-CODE, IC)<sup>34</sup>**

This is a letter that identifies the category of the record. The list of allowable categories, that we called *ClassName(s)*, is located in the File, MIT AFFILIATION CODE Table (Fig. 22). For example, the letter 'A' will identify the data in the record as corresponding to an MIT alumnus. In the database, it is represented as a one (1) alphanumeric character.

This field is multivalued, i. e., the same record can be classified as a member of up to five (5) different classes. In the database they are specified as IDENT-CODE(1), IDENT-CODE(2), IDENT-CODE(3), IDENT-CODE(4) and IDENT-CODE(5). Therefore, the domain definition of this field is a set of values that can be atomic and non atomic data values. The field can have a Null value; in this case, other fields, defined later in this document, should be queried to identify the class of the record.

The MIT Affiliation Code Table (Fig. 22) declares the classes of the ALF-MASTER File. Yet this table constitutes a non-exhaustive explicit declaration of the domain primitives stored in this file. Notice that its items do not fit into any standard form of structured category description, such as a type-hierarchy; the set is neither homogeneous nor complete but meant to fulfill the requirements of the Alumni Database role. It is also important to notice the overlap of types, e.g., alumnus and postdocs.

---

<sup>34</sup> The names of the field in the database are shown in parenthesis.

In order to categorize the records into classes other than the ones listed in the table, we need of additional knowledge, generalization, inference and like mechanisms.

**Sequence Number (SEQ-NUM, SQ)**

It is a unique name number identifier of every record in the File. In the database is represented with ten (10) characters numeric values.

<b>Identification</b>	<b>ClassName</b>
A	alumnus
D	postdocs
E	estate of alumnus
G	grandparents
H	honorary
J	joint
M	in mem or honor
N	non-alumnus
O	alum owned entity
P	parents
R	relations
S	spouse
T	estate of trust
U	alum anonymous
W	widow
X	other

**Figure 22: MIT Affiliation Code Table**

This is one of the fields that the alumni database staff uses to identify entities as follows:



- If the person is an MIT alumni, the first four digits represent the year of graduation so that number<sup>35</sup> is always 19XX, e. g., 1956, the other 6 digits have no semantic meaning.
- If the entity is a company, then the first four digits are 1000.
- If any other entity they query the IDENT-CODE field to find out , for example, that it is a living-group (Fig. 22).
- To identify a company, the IDENT-CODE (1) is an N and the SEQ-NUM must start with 1000.
- There exist a table, part of the ALF-TABLES File, where special SEQ-NUM series are kept; that is, entity-objects can be identify, in some special cases, by the first four digits of their SEQ-NUMs. The table is 999-SEQUENCE-NUMBERS.

The reason why they use this field in addition to IDENT-CODE to identify entities, is because information from both fields is necessary to query entities not explicitly declared in the IDENT-CODE table (either for confidential or for practical reasons).

#### **Last Name** (LAST-NAME, LN)

The content of this field depends on the type of record:

- If record is for a person, then LAST-NAME stores a person's surname.

---

<sup>35</sup> In this and the following examples, the letter 'X' stands for any alphanumeric character.

- If record is for a corporation, association, fraternity, etc. it can be:
  - the name, if its number of characters is less than the field's length; or
  - the first part of the name. In this event, the second part of the name is stored in the FIRST-NAME field;

It is represented as a twenty (20) characters alphanumeric value.

In order to define the meaning of this field we need to know the IDENT-CODE of the record. IDENT-CODE will tell us whether "Smith" is an alumni, widow, etc. That is, only after identifying the type of record, one can tell what information the LAST-NAME of that record contains. For example, if a record has IDENT-CODE(1) = A, then LAST-NAME is the surname of a person, in this particular example, the surname of an MIT alumni.

Yet the IDENT-CODE field is not enough to define LAST-NAME's contents. For example, there are entities such as "companies" recorded in the database without an specific *IDENT-CODE*; indeed, been the second most important type of entity-objects in the Alumni/Gift, companies are not explicitly declared as a category anywhere in the system. To solve this particular problem we need two pieces of information, IDENT-CODE(1) = N and SEQ-NUM = 1000XXXXXX to infer that the LAST-NAME field stores a company name. The field can have a Null value.

**Mailing name (MAIL-NAME, AN)**

This is a complete name in mailing format. It may include prefix and suffix particles to address the person(s). In the case of a company, it is the name to be used for mailing purposes. It is represented by thirty (30) characters alphanumeric values.

The field free of any semantics constrains. The owner(s) of the record choose the format and content of the mailing name; for example, whether or not to include a prefix with the name, whether or not to include more than one person's name are decisions left to the owner of the record (alumni, company, joint, etc.) Semantically this field can be thought of as a set of attributes - attributes-group in the IRDS terminology.

The field is likely to be the only existing name in the case of a non-person record. At the same time, it may not contain the name of the owner of the record but the name of other entity to which mail should be sent. For example in the case of a company, the MAIL-NAME can be a particular person within the company to whom mail should be sent.

**Middle Name (MID-NAME, MN)**

It is a person's middle name. It can have up to 15 alphanumeric characters. The field has a Null value for non-person records. The semantics of this field regarding its EntityObject identification are similar to those presented in the LAST-NAME field.

**First Name (FIRST-NAME, FN)**

If the record pertains to a person, this field stores his/her first name. When the record belongs to a non-person, this field is either Null or it contains the second part of the entity's name. In the later case, the first part of the record's name is under the LAST-NAME field. This scheme allows for names longer than the LAST-NAME field length to be included in the database. The field's length is 30 alphanumeric characters.

**Legal Name (LEGAL-NAME, LE)**

This is the legal name of the record. They may request to be addressed one way but the legal name is the formal name and it is stored in this field. For example, a non-US born may choose to change or alter his/her name to make it easier to pronounce, a woman who gets married may choose to alter her name in a variety of ways. In all cases, the LEGAL-NAME is the original name and the modified name is stored in the other name fields present in the database. No prefix or suffix particles added to it. It has 60 alphanumeric characters. The field can have a Null value.

**Prefix Code (PFX-CODE, PX)**

It is a particle(s) added at the end of a name to formally address a person, organization, etc. It has up to 6 alphanumeric characters. There is a list of predefined prefixes; the prefix particles are listed in the PREFIXES Table of the ALF-TABLES File<sup>36</sup>:

---

<sup>36</sup> This list contains the translation of the 6 alphanumeric characters. They were taken from the above mentioned PREFIXES Table.

Admiral	Airman	Attorney	
Brother	Brigadier General	In Honor of	
Judge	Lieutenant	Private First Class	
The Very Reverend	1st Lieutenant	2nd Lieutenant	.....

**Suffix Code (SFX-CODE, SF)**

It is a particle(s) added at the end of a name to formally address a person, organization, etc. It has up to 6 alphanumeric characters. Some examples are listed bellow<sup>37</sup>:

Esquire	Estate of	Friends of
Junior	Trust of	U S Navy
Medical Doctor	PHD	Doctor of Science
Senior	S. J. Superior	U S Army
U S Air Force	U S Coast Guard	U S Marine Corp

**ALUMNI/GIFT SOURCE DOMAIN DESCRIPTION**

**Context\_Name:** MIT-Alumni/Gift

**Context\_Domain:** Person-Alumnus WHERE [Alumnus-->school=MIT]  
*[school-domain-property-list,  
administrative-domain-property-list]*

**Domain\_Constraints:** Time *[1970 - present]*  
Location *[Cambridge, Massachusetts, USA]*  
Size *[500 Megabyte]*

**Context\_Access:** Network Datatabase,  
Natural Language

**Context\_Structure:** Schema Num. 1  
*[ALF-MASTER-VIEW (SQ (char 10), ..., PFX (char 6))]*

**Figure 23: Alumni/Gift Database Source Context**

Nonetheless, the field can have a suffix value other than the ones present in this list.

---

<sup>37</sup> The list is a translation of the 6 alphanumeric character SFX-CODEs found in the ALF-TABLES file of Alumni/Gift.

### 4.2.3 Context interchange view

The addition of context to this database goes as follows:

- The source context was presented in Chapter 3 and we included here again only for the purpose of completeness (Fig. 23)
- The description of the fields is presented in the following pages.

#### Identification Code CDL

##### Identification

- *Access\_Name* [IDENT-CODE, ID]
- *Is\_In* [ALUMNI/GIFT. ALF-MASTER. IDENT-CODE]

##### Links

- *Is\_Kind\_Of* [ EntityObject --> ClassName ]
- *Is\_Part\_Of*<sup>38</sup> [ Null ]
- *Has\_Parts*<sup>39</sup>[EntityObject-->ClassName(1), EntityObject-->ClassName(2),  
EntityObject-->ClassName(3), EntityObject-->ClassName(4),  
EntityObject-->ClassName(5)]
- *Relates\_To*<sup>40</sup> [ EntityObject -->Name,  
EntityObject-->IdentificationNumberName ]

##### Representation

- *Value\_Type* [ (format: alphanumeric-plain), (units: Null),

---

<sup>38</sup> This is the only field in the database containing class name information.

<sup>39</sup> List of the five ClassName(s) properties that a given record can have.

<sup>40</sup> List (not complete) of the other EntityObject names present in the database.

(scale: Null ), (length: char(1) ]

- *Allowable\_Values*<sup>41</sup> for [IDENT-CODE(1)]  
[ A, P, E, G, H, J, M, N O, P, R, S, T, U, W, X, Null ]  
for [IDENT-CODE(1) & IDENT-CODE(2)]  
[ A&P, A&E, .....];  
.....  
for [IDENT-CODE(1)& ... & IDENT-CODE(5)]  
[A&P&G&H&S, ..... ]
- *Cardinality* [ 1, 2, 3, 4, 5 ]
- *MetaPropperties* [ Null ]

**Constraints**<sup>42</sup>[ALF-MASTER, ALF-MASTER-VIEW ]

**Missing\_Value**<sup>43</sup> if [ IDENT-CODE(1) = Null ] then [ look\_up (SEQ-NUM) ]

**Transformations**<sup>44</sup>

if [(IDENT-CODE(1)=J) or (IDENT-CODE(2)=J) .... or (IDENT-CODE(5)=J)]<sup>45</sup>  
then [ EntityObject-->CARDINALITY = 1 ) ]

---

<sup>41</sup> The domain of this field includes 1-values-set (listed in full), and from 2-values up to 5-values sets (not listed in full).

<sup>42</sup> Declares the database views from which the object can be accessed.

<sup>43</sup> If the value of this field is Null, we fetch a query to look up the SEQ-NUM of the record, and from that value we can infer the identification code (ClassName) of the record.

<sup>44</sup> Global entity-object names are inferred from the local names and some other information, i. e., the cardinality.

<sup>45</sup>If none of the IDENT-CODEs of the record is J (join), then the record is a single entity-object record; otherwise the record corresponds to a joint entity-object and, therefore with cardinality greater than 1. Notice that we are declaring here the EntityObject's cardinality not the EntityObject-->ClassName's cardinality. The later was done a few lines above.

```

else [EntityObject-->CARDINALITY > 1]
if [EntityObject-->ClassName-->CARDINALITY= 1]46
  if [IDENT-CODE(1) = A]
  then [EntityObject --> ClassName = Alumnus,
        EntityObject --> ClassName = MITAlumnus,
        EntityObject --> ClassName = UniversityAlumnus]
  if [IDENT-CODE(1) = D]
  then [EntityObject --> ClassName = Postdoctor,
        EntityObject --> ClassName = MITPostdoctor,
        EntityObject --> ClassName = Researcher]
.....
if [EntityObject-->ClassName-->CARDINALITY = 2]47
  if [IDENT-CODE(1) = A and IDENT-CODE(1) = D]
  then [EntityObject --> ClassName = Postdoctor,
        EntityObject --> ClassName = MITAlumnus]

```

## Last Name CDL

### Identification

- *Access\_Name* [LAST-NAME, LN]
- *Is\_In* [ALUMNI/GIFT. ALF-MASTER. LAST-NAME]

---

<sup>46</sup> In this case, the record has only one identification code, that is, only IDENT-CODE(1) has a non-null value. The scheme presented defines the ALF-MASTER records as members of global entity-object classes. Only two cases are presented (A and D) the rest of the IDENT-CODE are omitted.

<sup>47</sup> When the cardinality of IDENT-CODE is 2 or greater than 2, new ClassNames can be inferred by mapping sets of IDENT-CODEs to one single new ClassName. In this case, the record has two identification codes; that is, IDENT-CODE(1) and IDENT-CODE(2) have non-null values. For example, one can retrieve all MIT alumni who are postdocs. Notice also that we include only one example of cardinality 2 and no examples for cardinalities 3, 4 and 5.



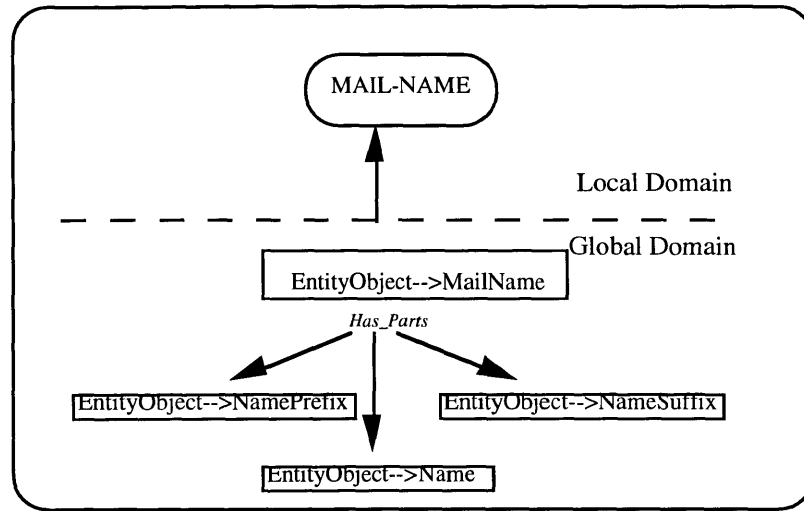


Figure 24: Local and Global Links

### Links

- *Is\_Kind\_Of* [ EntityObject --> Name ]
- *Is\_Part\_Of*: [ EntityObject --> FullName]
- *Has\_Parts* [ Null ]
- *Relates\_To* [ Null ]

### Representation

- *Value\_Type* [ (format: alphanumeric-plain), (units: Null),  
(scale: Null ), (length: char(20) ]
- *Allowable\_Value* [ XXXXXXXXXXXXXXXXXXXXXXXXXX ]
- *Cardinality* [ 1 ]
- *MetaProperties* [ALF-MASTER.IDENT-CODE,  
ALF-MASTER.SEQ-NUM ]<sup>48</sup>

---

<sup>48</sup>Both MetaProperties are ALF-MASTER fields, therefore we can infer their values querying the file.

**Constraints** [ALF-MASTER, ALF-MASTER-VIEW]

**Missing\_Value** [LAST-NAME = Null]<sup>49</sup>

then [EntityObject -->LegalName]

**Transformations**

if [IDENT-CODE = (A, D, ..., W) ]<sup>50</sup>

then [LAST-NAME = Person -->LastName]

if [ IDENT-CODE(1) = N and SEQ-NUM = 1000XXXXXX ]

then

if [FIRST-NAME = Null]

then [LAST-NAME = Company -->FullName]

else

[ALF-MASTER.LAST-NAME & ALF-MASTER. FIRST-NAME)=

Company --> FullName]

if [IDENT-CODE (1) = J ]<sup>51</sup>

then [LAST-NAME = Null ]

---

<sup>49</sup> If LAST-NAME is Null, we chose to give the more general property EntityObject-->LegalName as alternative.

<sup>50</sup> That is, if any of the record's IDENT-CODEs belongs to the list of the person categories.

<sup>51</sup>The (only) name of a joint should be under the JOINT-NAME field.

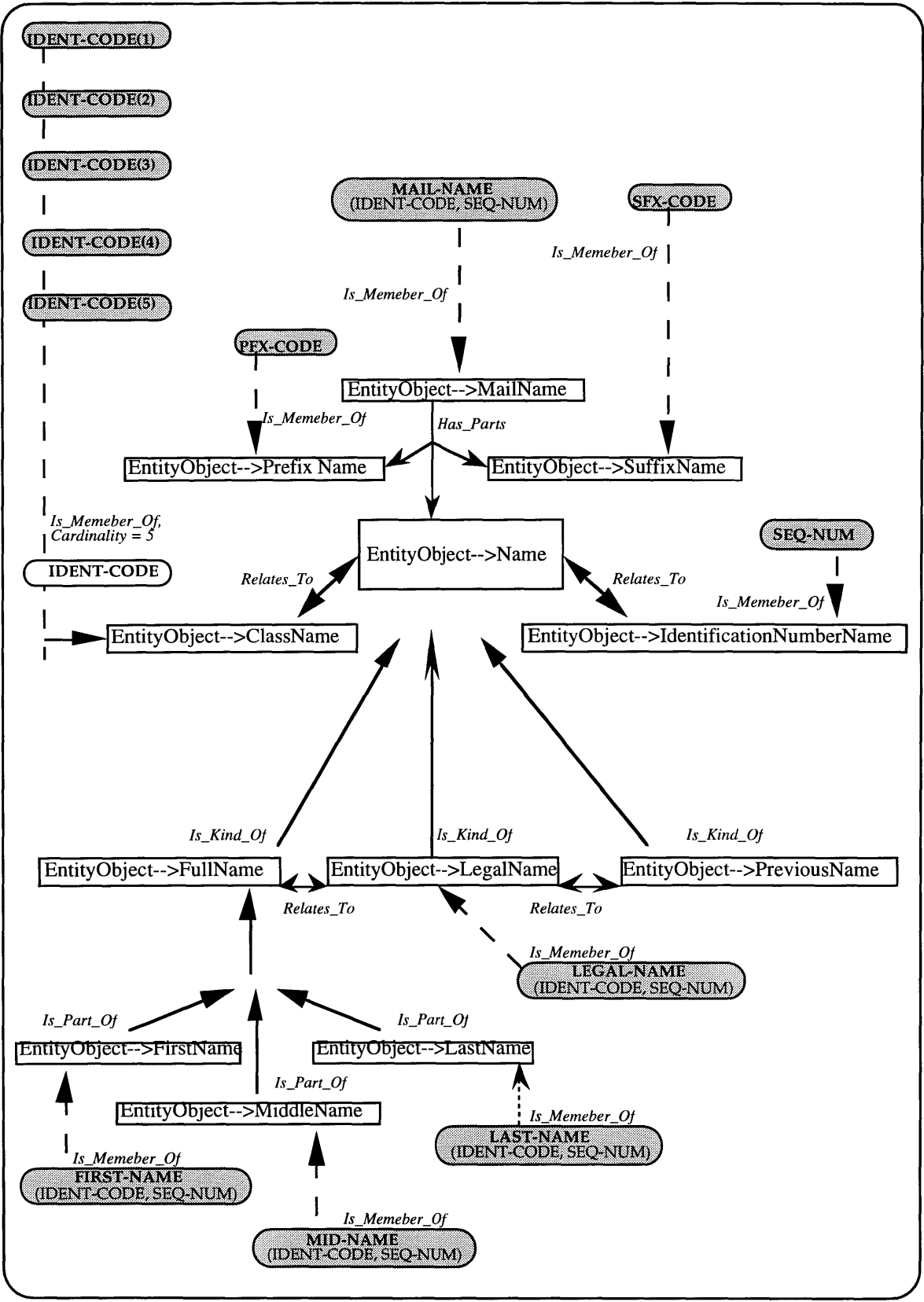


Figure 25: Subset of the Ontology

## 4.3 The Topologically Integrated Geographic Encoding and Referencing (TIGER) Files

### 4.3.1 Domain

The TIGER files [TIGER 91] are network files for the whole USA and its territories designed as basic mapping tool for the Census by the Bureau of Census. It integrates all the information necessary to administer the Census enumeration process, i. e., these files contain a digital description of the nation's political and statistical geographic areas. They are an extract of selected geographic and cartographic information from the Census Bureau's TIGER Database<sup>52</sup>. They can be thought of a digital street maps overlaid with complete 1990 Census geography (e. g. tracts, blocks, etc.) and postal information (e. g., street names. address ranges and ZIP codes). The normal geographic coverage for each file is a county.

### 4.3.2 Sub-domains

Overall the files contain the following information:

- **Census Geography:** tracts, blocks, voting precincts, etc.
- **Postal Geography:** street name, address ranges and ZIP codes.
- **Coordinate measurements:** latitude, longitude, state plane or any other world-wide reference scheme.

---

<sup>52</sup> All statistical maps and reference maps for the Census are computer plotted from the TIGER database. This database is used to perform statistical analysis. It is only a graphic representation of ground truth.

## CHAPTER 5

### PROTOTYPE<sup>54</sup>

#### 5.1 Scope of the prototype

The prototype consist of a global ontology organized as a global network (database) directory. The directory is the place where the systems and their high level descriptions are stored. This part intends to mimic the services currently offered by the network directory part of the OSI open system standards. It is more powerful than the OSI's current version in the sense that the domains stored in the real directory express only an address with no meaning content that con support intelligent query of the network.

For a given domain, the directory will support the (hierarchical) storage of their ontos and other possible relations among them. These ontos point to the sources where information about then is contained. For example, we have designed 'Educational institutions' and "Territories" Domains (Fig. 7). and place them as part of the above mentioned directory. These two domains are related by the fact that they have some common ontos, i. e., Alumnus: this information is also included as part of our directories

---

<sup>54</sup> The ideas exposed in this chapter as well as the code presented in Appendix Num. 1 are a preliminary design of the prototype that is currently being designed as part of the Context Interchange effort.

- **Topological information:** Expression of how streets intersect and how lines connect to form block or tract boundaries.
- **Other** features such as hydrography<sup>53</sup>.

Particularly, all the above mentioned information is stored by census geographic area codes, names for basic map features and address ranges in flat file format. Each file contains 12 record types that collectively contain geographic information (attributes) such as address ranges and ZIP codes for street segments, name and codes of the feature type, codes for legal and statistical entities, etc. The 12 record types come in the same tape as 12 separate files. Record types 1 and 2 are enough to develop maps for a given area.

#### **4.3.3 Context interchange view**

The record types 1 and 2 of the TIGER files store mainly the topological information. The information contained in the line segments encoded in these types is used to construct the maps. In this case, the semiotics of character string data are interpreted by programs intelligent enough to know, for example, how to build polygons (census blocks, parcels, ZIP codes, etc.) from the line segment semantics whose description is available in the TIGER files data dictionary [TIGER 91].

---

<sup>53</sup> In some cases, these additional information is provided by the Bureau of Census in additional files compatible with the base-files, i. e., TIGER.

From a pragmatic point of view, the user of the system can ask for information on the available domains. He/she will be given the option to go to either go to the sources or to navigate through the network directory for more specific domains.

## **5.2 Design issues**

The prototype that we have implemented consist of the following modules:

### **knowledge representation mechanisms**

Interconnected frames implemented as C++ classes. In them we store the information about the system in form of the defined export context query-import context.

### **knowledge base**

To allow automatic storage and access to the information representing the nodes of the network, we organize them in the network directory. All the system containing information on the same domain are grouped under the same class and listed under the file that implements that "domain-class"

## CHAPTER 6

### CONCLUSION

Nowadays, data integration strategies in large-scale computing environments must address the design and implementation of technologies and tools to manage the heterogenieties originated from disparate component data environments. Aiming to facilitate to the user to locate, to access and to merge the information she/he wants, this thesis presented a methodology for data environment encoding and description. Our work was built upon the context interchange data integration approach. The rest of this chapter describes our findings and future research.

#### **6.1 What is missing today?**

First, we reviewed (Chapter 2) the data definition and data access schemes in current systems. This survey of current data environment representation techniques revealed that operational systems have explicitly incorporated limited information of the systems themselves and their data. In general, this information is stored, using a very limited number of constructs provided by the data manager:

- Types.



- DDLs and DMLs.
- Schemas, E-R diagrams and other logical modeling tools.
- System-wide data repositories, warehouses and dictionaries.

These data manipulation tools are kept segregated from the data itself, and accessed by entirely different means. All of them can be qualified as static metadata declaration and manipulation tools. None of them contemplates the idea of adding meaning to its data definition scheme, and sharing this meaning with other systems for data exchange purposes.

In addition, these data representation and manipulation schemes (from the simple flat file data model to more sophisticated semantic models) and their associated data "structures" (data type structure, entity structure, relational table structure, etc.), do not contemplate representation and declaration of themselves at global network level to facilitate global queries and source identification to potential users. We suffered these problems while performing this study, we queried the systems that we were analyzing and observed that their absence resulted in poor accessibility, misuses of data, etc.

### **Problems caused by what is missing in our systems**

Continuing with our study, we showed the collection of data disparities (conflicts) whose representation is not covered by current data models. Conflicts arise when more than one value<sup>55</sup> is possible for a given concept. For us, conflict

---

<sup>55</sup> Here, value is used in a generalized way.

mediation and resolution is to choose a particular value that is acceptable in the receiver environment (context). We classified them as

- 1-dimensional conflicts.
- N-dimensional conflicts.
- Naming conflicts

All these issues mentioned above - static and incomplete representation tools, no network query facilities, etc. are systematically preventing us from achieving data integration over the network. To correct this situation, we presented a novel integration paradigm: the context interchange theory.

## **6.2 Context interchange**

In Chapter 3, we began our work of building open data environments based on the context interchange paradigm. We defined the context of a system, i. e., data environment as the representation of its Domain, its data semantics, its definition and access tools (DDL and DML), its schema (how data is organized and how to access it), and the representation of the meaning of its data in terms of a preexisting global ontology.

### **The client/server context application**

The tools we envision to develop the context application, especially systems' data environments, include a combination of database technology, knowledge-base

technology and artificial intelligence techniques. In particular databases and knowledge-bases bear the responsibility to store and search the shared information. We believe that artificial intelligence programming techniques ought to be used to manipulate that context (data) knowledge. For example, we saw in this thesis that rule-based representation technologies capture guesses that are not necessarily sound or true in any “reasonable” model and we found this knowledge representation technique to be very appropriate to describe the specialized and idiosyncratic data environments.

Context, as a repository of knowledge for data structure, data semantics and data meaning, closes the gap between data and information. We described Context Interchange as an application, in a client/server environment, embedded in the application layer of the OSI model.

### **Context model**

We described the system as a Domain containing EntityObjects, Properties and MetaProperties. Set theory and transformation theory were used to model domains and relations and transformations among them.

Our next step was to define a context definition language. CDL structures contain declarative and procedural knowledge (table look-ups, heuristics, rules, database queries, and others) stating meaning, semantics, logical location, etc. of the system and its data.

### **High level domain representation**

We design a high level CDL representation structure for context with the purpose of helping to locate the data sources belonging to a particular Domain.

This tool is similar to the OSF DCE directory services. The components of this high level description of the system are:

- **Context\_Name**
- **Context\_Domain**
- **Context\_Constraints**
- **Context\_Access**
- **Context\_Structure**

Nonetheless, we observed that in the absence of a fully-featured language, ad hoc query languages (for example, one supporting queries to the fields listed above) at best can only be used for simple and limited read-only access. Even then, considerable sophistication is likely to be required for the user if the data extracted from the database is not to be a meaningless mess. If more powerful ways of accessing a system are required, then we need to refine our CDL structure and add to it more local domain representation capabilities. This is our next theme.

### **Local domain representation**

We defined CDL structures, local to the system, where we embedded the characteristics of the EntityObjects, Properties, MetaProperties and Domain of the system. Properties were presented as belonging to a single EntityObjects. Although we could have presented them as belonging to other entity-objects - using the context knowledge captured within the CDL structures, we chose not to. At global level, it would had forced us to create a larger global ontology, defining additional global entities, global properties, etc. which are tasks out of the scope of this paper. At local level, promoting the ALF-MASTER EntityObject

to, for example, 'Person' involves changes and redefinition of some of its Properties that are hard to capture given the limited global ontology at hand. This database field level CDL was implemented with the following components:

- **Definition**

It captures the name and logical location of the data in the system's schema;

- **Links**

They express the data domain relative to a predefined global ontology.

- **Representation**

Data semantics and syntax. It includes the MetaProperties, i. e., the domain primitives needed to complete data description.

- **Constrains**

Logical data access constrains and connections.

- **Missing\_Value**

Error handling scheme.

- **Transformations**

Knowledge representation and manipulation schemes needed to transform local terms into global terms.

## 6.3 Context acquisition: definition and views

Our analysis in Chapter 4 demonstrated that even in very disparate environments (a hierarchical database system and a collection of flat files) the problems involved in data environment description, i. e., context representation are very similar. They relate to two sets of different issues:

- **Views definition**

If the goal is to create a mechanism allowing for access to all the data structures, the representation of context is very difficult. As opposed to that, we chose to represent views that, showing a subset of the existing data, can support a number of key queries to the system.

- **Data analysis**

On the other hand, our experience has shown that these “front-end” products to existing data environments are not straight forward to design and involve a lot of data analysis (updated documentation, if exit, ...). Therefore, to build a context of a mature system is not a straight forward task and requires cooperation from both the context knowledge engineer and the developers of the data environment (in most of the cases they are the database administrators and designers),

### **Varieties of context**

Related to the above mentioned context views is the issue of varieties of context that we found to be a very crucial context design issue. For example, the ALF-MASTER File has the following fields containing name information:

LEGAL NAME	FIRST NAME	MIDDLE NAME
LAST NAME	IDENTIFICATION CODE	NAME GPR
MAIL NAME	NICK NAME	SHORT NAME
JOINT NAME	SEQUENCE NUMBER	PREVIOUS NAME

Depending on the receiver, these fields can be interpreted as different, equivalent, meaningful, etc. This problem is due to the different levels of granularity, degrees of specificity, abstraction, etc., under which to encode a concept<sup>56</sup>. Therefore, the real issue is how many of the above names should be exposed or, in other words, how much information is to be exposed, or can be exposed in the system context.

For example, in the case of Alumni/Gift, we can choose to give strictly alumni administrative information and eliminate from its context all references to the existence of companies, associations, etc. and their associated information (Fig. 25). We could have chosen to give only the FULL-NAME field and hide the rest of the names present in the system, etc. These scheme creates the notion of possible different contexts in a single system, also called varieties of context.

## 6.4 Network interoperability through context

Incorporating context knowledge to the system provides a complete and consistent view of the data residing in the local system<sup>57</sup>. Once this

---

<sup>56</sup> Recall that we modeled those disparate materialization of the same concept linked by Relates\_To Link; at the same time, we had all of them as Is\_Part\_Of the property concept common to all of them. The scheme is presented in Figure 25 for a subset of the above names.

<sup>57</sup>For example recall from the Alumni?Gift example that we declared that the LAST-NAME field is besides a person surname, a company name, part of a company name, etc.

contextualized view of the data has been provided, the system can be understood and interpreted by any network user accessing the resources kept in the system, without the harms of misuses and misinterpretations of its data.

Context as new interface to the system provides:

- **Transparency:** the underlying meaning, organization and semantics of the system are reflected in the interface.
- **Consistency:** different parts of the system do similar things in a similar way.
- **Simplicity:** number of elements, basic EntityObjects and Properties of the system declared globally, should be kept low.
- **Complexity:** number and range of ad hoc verification conditions that have to be added to the schema in order to ensure that the data in the database remains consistent and that context is reflecting the underlying data environment is very important.

In order to design system interfaces with these properties, we need to address the following issues in the near future:

### **How much local?, how much global ?**

In the data integration strategy, we should define loosely-coupled versus tightly couple approaches to integration (related to the previous point). Why? Because It is a critical decision regarding how much knowledge should be shared.



Along these lines, we need to decide how much of the reasoning should be embedded in CDL frames: should they just be self-describing declarative structures? should they perform reasoning when attending user requests?

The last global vs. local point deals with the ontology. The global ontology and the implications (mainly at local system global level) of promoting local EntityObjects to global EntityObject, promoting Properties to MetaProperties, etc., are issues that need to be addressed more in-depth by future research.

### **The parts of context interchange that we did not explicitly address**

The thesis targeted the definition of context and context knowledge and the tools to assist the development of context definitions and the maintenance of this knowledge, i. e. data environments. Nonetheless, we lack a target context to perform context comparisons and transformations across the network<sup>58</sup> and we did not address:

- context mediation

It will be targeted to facilitate context mediation, that is, the ability to query context knowledge from disparate systems.

- Context resolution

Detects and tries to resolve (manipulate) a data conflict

---

<sup>58</sup>The reader may remember that we presented some examples of absolute conversions (from local to hypothetical global properties), included in the transformations layer of our CDL frames. However, we did not formally address the issues of context comparison and context conversion.

We hope to continue our work on the development of context interchange, specially regarding the development of the global ontology and the development of mapping functions from concepts in the ontology to the (local) data definition schemes that we presented in this thesis.

-- FIN --

## References

[**ACM 90**] ACM Computing Survey. Special Issue on Heterogeneous Databases. Vol. 22, Num. 3. 1990

[**Andrews & Leventhal 93**] Andrews, D. and Leventhal, N. Fusion: Integrating IE, CASE and JAD: a Handbook for Reengineering the System Organization. Prentice -Hall, 1993.

[**Atkinson & others 90**] Atkinson, M., Bancilhon, F., Dewitt, D., Dittrich, K., Maier, D., Zdonik, S. The Object Oriented Database System Manifesto. Proceedings of the ACM SIGMOD Conference. Atlantic City, 1990.

[**Atre 92**] Atre, S. Distributed Databases, Cooperative Processing and Networking. MacGraw Hill Database Expert's Series, 1992.

[**Backus 78**] Backus, R. Can Programming Be Liberated from the Von Newman Style? A Functional Style and its Algebra of Programs. Communications ACM, 21, 1978.

[**Benyon 90**] Benyon, D. Information and Data Modeling. Blackwell Scientific Publications, 1990.

[**Brachman & Levesque 85**] Brachman, R. and Levesque, editors. Readings in Knowledge Representation. Morgan Kaufman Publishers, 1985.

[**Bright & others 92**] Bright, M., Hurson, A. and Pakzad, S. A Taxonomy and Current Issues in Multidatabase Systems. IEEE Computer, March 1992.

[**Chen 76**] Chen, P. The Entity-Relationship Model: Towards a Unified View of Data. ACM TODS, 1:1, March 1976.

[**Codd 70**] Codd, E. The Relational Model of Data for Shared Data Banks. Communications ACM, 13:6, June 1970.

[**Collet & others 91**] Collet, C. and others. Resource Integration Using a Large Knowledge Base in Carnot. IEEE Computer. December 1991.

[**Date 85**] Date, J. An Introduction to Database Systems. Addison-Wesley, 1985.

[**Davis & others 93**] Davis, R., Howard, S., Szolovits, P. What is a Knowledge Representation? AAI, Spring 1993.

[**Devanbu & others 91**] Devanbu, P., Brachman, P., Selfridge, P., Ballard, B. LaSSIE: a Knowledge-based Software Information System. Communications of the ACM, Vol. 34, No. 5, May 1991.

[**Eckel 93**] Eckel, B. Inside and Out C++. Osborne MacGraw Hill, 1993.

[**Elmarsri & Wiederhold 81**] Elmarsri, R. and Wiederhold, G. GORDAS: a Formal High-Level Query Language For the Entity-Relationship Model. Proceedings of the Second International Conference on Entity-Relationship Approach, Washington, D.C., October 1981.

[**Giordano 93**] Giordano, R. The Information "Where?" House. Database Programming and Design. September, 1993.

[**Goh & others, 94**] Goh, C., Madnick, S. and Siegel, M. Context Interchange: Overcoming the Challenges of Large-Scale Interoperable Database Systems in a Dynamic Environment CISL WP # 94-01, Massachusetts Institute of Technology, 1994.

[**Goldfine & Koning 88**] Goldfine A. and Koning P. A Technical Overview of the Information Resource Dictionary System (Second Edition). NBSIR 88-3700, National Bureau of Standards, 1988.

[**Gupta 89**] Gupta, A. Integration of Information Systems: Bridging Heterogeneous Databases. IEEE Press 1989.

[**Hammer & Mcleod 81**] Hammer, M. and Mcleod Database Description with SDM: A Semantic Database Model. Communications of the ACM, Transaction on Database Systems, 6(3), 1981.

[**Hu 89**] Hu, D. C/C++ for Expert Systems. MIS Press, 1989.

[**Jones 92**] Jones, M. Unveiling Repository Technology. Database Programming and Design, April 1992.

[**Kent 78**] Kent, W. Data and Reality: Basic Assumptions in Data Processing Reconsidered. 1978.

[**Kent 79a**] Kent, W. Limitations of Record-Based Information Models. ACM TODS, 4:1, March 1979.

[**Kent 79b**] Kent, W. Data and Reality. Basic Assumptions in Data Processing Reconsidered. North-Holland Publishing Company, 1979.

[**Kent 91**] Kent, W. The Breakdown of the Information Model in Multi-Database Systems. SIGMOD Vol. 26, No. 4. December 1991.

[**Kernigham & Ritchie 86**] Kernigham, B. and Ritchie, D. The C Programming Language. Prentice-Hall.

[**Kim 92**] Kim, W. Object Oriented Databases. MIT Press, 1992.

[**Lenat & others 90**] Lenat, D., Guha, R., Pittmank, K., Pratt, D., Shepherd, M., CYC: toward Programs with Common Sense. Communications of the ACM, 33(8), 1990.

[**Lisov & Guttag 86**] Lisov, B. and Guttag, J. Abstraction and Specification in Program Development. MIT Press / McGrawHill, 1986.

[**Madnick & others 90**] Madnick, S. Siegel, M. and Wang, R. The Composite Information Systems Laboratory (CISL) Project at MIT. *Data Engineering*, Vol. 13, No. 2, June 1990.

[**Malone & others 87**] Malone, T., Grant, K., Turbak, F. Brobst, S. and Cohen, M. Intelligent Information-Sharing Systems. *Communications of the ACM* Vol. 30, Num. 5, May 1987.

[**Mark & Roussopoulos 86**] Mark, L. and Roussopoulos, N. Metadata Management. *IEEE Computer*, December 1986.

[**Martin 86**] Martin, J. *Data Types and Data Structures*. Prentice-Hall International, 1986.

[**McCarthy 84**] McCarthy, J. *Scientific Information = Data + Metadata*. Database Management: Proceedings of a Workshop. Navy Postgraduate School, Stanford University, California, 1984.

[**McCarthy 88**] McCarthy, J. *The Automated Data Thesaurus: A New Tool for Scientific Information*. Proceedings of the 11th International Codata Conference , September 1988.

[**Minsky 68**] Minsky, M., editor. *Semantic Information Processing*. MIT Press, Cambridge, MA, 1968.

[**Morgesten 84**] Morgesten, M. *Active Databases as paradigm for Enhanced Computing Environments*. Conference on Very Large Databases, 1984.

[**Navathe 92**] Navathe, S. *Evolution of Data Modeling for Databases*. *Communications of the ACM*. Vol. 35, No. 9, September 1992.

[**Neira & Madnick 93**] Neira, M. and Madnick, S. *Context Interchange Theory: towards the Definition and Representation of Context Knowledge*. CISL WP # 93-08, Massachusetts Institute of Technology, 1993.

[**Palinsar & Gamer 90**] Palinsar, F. and Gamer, M. Mapping the unmappable: plumbing the depths of crossfile and crosssystem navigation. Online, July 1990.

[**Patel-Schneider & others 84**] Patel-Schneider, P., Brachman, R. and Levesque, H. Argon: Knowledge Representation Meets Information Retrieval. Proceedings of the First Conference on Artificial Intelligence Applications, 1984.

[**Ross 81**] Ross, R. Data Dictionaries and Data Administration: Concepts and Practices for Data Resource Management. New York: Amacon, 1981.

[**Rumble & Smith 90**] Rumble, J. and Smith, F. Database Systems in Science and Engineering. Adam Hilger 1990.

[**Shen & others 91**] Shen, W., Huns, W. and Collet, C. Resource Integration without Application Modification. MCC Technical Report Number ATC-OODS-214-91, 1991.

[**Sheth & Larson 90**] Sheth, A. and Larson, J. Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases. ACM Computing Surveys, March 1990.

[**Shipman 79**] Shipman, D. The Functional Data Model and the Data Language DAPLEX. Proc. ACM SIGMOD'79 Conference. Boston, 1979.

[**Siegel & Madnick 89a**] Siegel, M., and Madnick, S. Schema Integration Using Metadata. CISL Working Paper No. 89-06, Massachusetts Institute of Technology, 1989.

[**Siegel & Madnick 89b**] Siegel, M., and Madnick, S. Identification and Reconciliation of Semantic Conflicts Using Metadata. CISL WP # 89-09, Massachusetts Institute of Technology, 1989.

[**Siegel & Madnick 91**] Siegel, M. and Madnick, S. Context Interchange: Sharing the Meaning of Data. ACM SIGMOD Record, April 1991.

[**Siegel & Madnick 92**] Siegel, M. and Madnick, S. A Metadata Approach to Solving Semantic Conflicts. Conference on Very Large Databases, 1991.

[**SIGMOD 91**] SIGMOD Record. Special Issue: Semantic Issues in Multidatabase Systems, December 1991.

[**Shoman 91**] Shoman, Y. Varieties of Context. Artificial Intelligence and Mathematical Theory of Computation: Papers in honor of John McCarthy. Academic Press, 1991.

[**Smith & Smith 77**] Smith., J. and Smith, D. Database Abstraction, Aggregation and Generalization. ACM TODS, 2:2, June 1977.

[**Takagi 92**] Takagi, A. Multivendor Integration Architecture (MIA) Overview. NTT Review, Vol. 3 Num. 5, September 1992.

[**Tanenbaum 88**] Tanenbaum, A. Computer Networks. Prentice-Hall 1988.

[**TIGER 91**] The Bureau of the Census. TIGER/Line Census Files, 1990 Technical Documentation. Prepared by the Bureau of Census. Washington: The Bureau, 1991.

[**Thompson 89**] Thompson, J. Data with Semantics, Data Models and Data Management. Van Nostran Reinhold, New York, 1989.

[**Wang & Madnick 89**] Wang, R. and Madnick, S. Facilitating Connectivity in Composite Information Systems. Data Base, Fall 1989.

[**Wang & Madnick 90**] Wang, R. and Madnick, S. A Poligen Model for Heterogeneous Database Systems: The Source Tagging Perspectives. January 1990.

[**Wiederhold 84**] Wiederhold, G. "Knowledge and Database Management" IEEE Software, Vol. 1, No. 1, 1984.



[**Wiederhold, 86**] Wiederhold, G. Views, Objects and Databases. IEEE Computer. December 1986.

[**Wiederhold 92**] Wiederhold, G. Mediators in the Architecture of Future Information Systems. IEEE Computer, March 1992.

[**Wiederhold & others 92**] Wiederhold, G. Wegner, P. and Ceri, S. Toward Mega Programming. Communications of the ACM Vol. 35, Num. 11, November 1992.

[**Wiederhold 93**] Wiederhold, G. Intelligent Integration of Information. Proceedings of the ACM SIGMOND Conference, May 1993.

[**Winston 94**] Winston, P. Onto C++. Addison-Wesley Publishing Company, 1994.

## **Appendix Num. 1: Source code**