MONIL, the Metadata and Object Integration Language

M. LARRE, J. TORRES, E. MORALES, S. TORRES
Department of Computer Science
Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Cuernavaca
Paseo de la Reforma No. 182-A Lomas de Cuernavaca
MEXICO

Abstract: - Data integration is the process of extracting and mergind data from multiple heterogeneous sources to be loaded into an integrated information resource. Developing tools for effectively exploiting source data has became a challenging issue with the appereance of the Data Warehouse Technology. This paper presents the MONIL Language as an alternative to solve data integration problems. The MONIL main features are described using a simple case of study. These features are: an algorith to automatically suggest integration correspondences between source and target data, a metamodel to store integration information, and a set of built-in conversion functions. MONIL Language is embedded in a semiautomatic framework with a set of tools to develop, store and execute integration programs following a simple integration process. A MONIL program execution generates Java language statements and JDBC commands. Integration cases have been successfully solved using MONIL language.

Key-Words: - data integration, databases, metadata, programming language.

1 Introduction

Data Integration is defined as the process of extracting, transforming and merging data from heterogeneous sources into an integrated information resource.

An integration system architecture ([1], [2]) can be defined in terms of wrappers and mediators. Wrappers extract data from sources, and mediators transform, merge and load data into a specific target.

Data integration is a complex problem since semantic and structural heterogeneity's between data exist. Structural differences come from the implementation details, e.g., data models and programming languages. Semantic heterogeneity occurs for example, when different names are used to represent the same object, or when same names represent different objects [3].

Two basic approaches have been proposed in the literature to solve the data integration problem: structural ([4], [5]) which uses a query-by-query approach, and the semantic approach ([7], [8]) which considers the conceptual descriptions of the involved sources.

In recent data integration systems, the user models each data source as a virtual relation called mediated schema. The virtual relation is built using integrated virtual views ([9], [10]) or materialized views ([11], [12], [13], [14]) and the user poses queries in terms of the mediated schema. In addition to the mediated schema, the model has a set of source descriptions that specify the mapping between the mediated schema and each source schema. Two basic approaches for specifying source descriptions are: the Global-As-View (*GAV*) approach ([16], [17], [18]) which uses rules to define the relations in terms of specific source data, and the Local-As-View (*LAV*) approach ([19], [20], [21], [23], [24]) where the source relations are defined as expressions over the relations in the mediate schema.

With the appearance of Data Warehouse technology ([25], [26]), where inconsistency and incompatibility problems need to be solved when data passes from Legacy Systems to Data Warehouses, the developing of efficient tools for solving data integration problems has become a challenging issue for the computer scientists.

This paper describes a programming language called MONIL[1] as an alternative for solving integration problems. MONIL is an expressive language based on metadata and embedded in a dedicated framework which follows a 3-step process to solve data integration problems. MONIL formal definition [27] has 263 basic production rules and it is based on a context free grammar [29].

MONIL Language is a hybrid approach that merges relevant concepts from the semantic and structural approaches, e.g., *(a)* the integration metamodel concept (semantic approach) to manage the integration process information, *(b)* direct

[1] **M**etadata and **O**bject i**N**tegrat**I**on **L**anguage

references to sources and targets (structural approach) to easily extract and load data, and *(c)* stored descriptions for sources and targets units (materialized views) to store the required metadata from each participant element.

The paper is organized as follows. Section 1 describes the process and solution of an integration problem using the MONIL approach. This section also describes MONIL features. Section 2 contains some conclusions and future work.

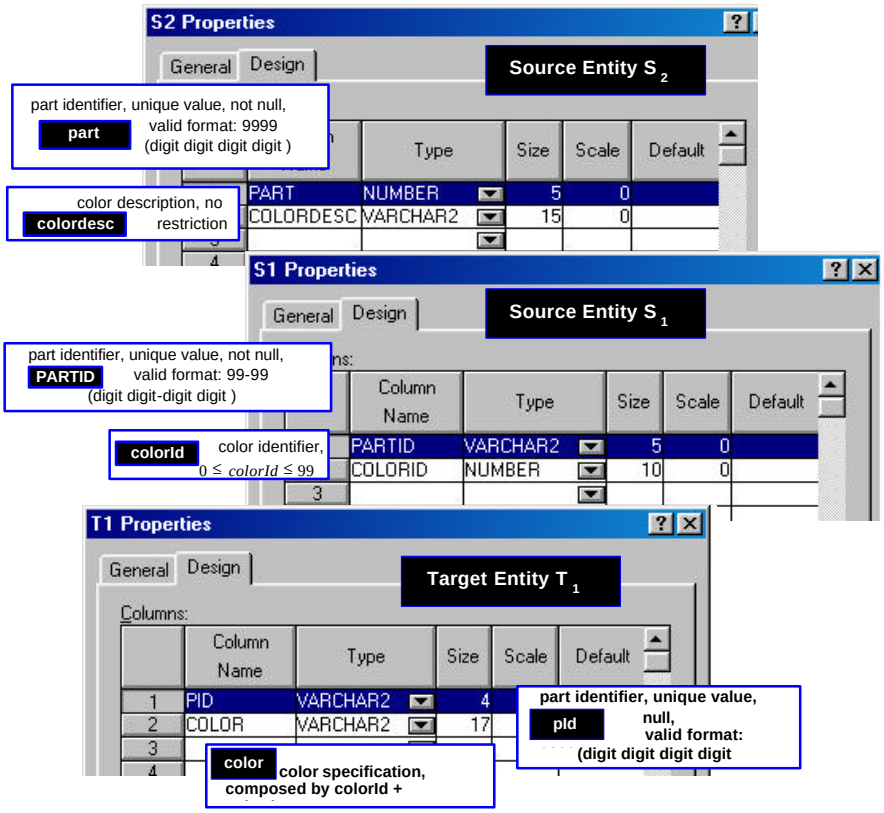

Figure 1. Sources and target conceptual descriptions.

2 A Motivating Example

This section gives an overview of the MONIL approach using a representative integration problem.
The integration problem consists in solving heterogeneities between three entities (two sources and one target) from one relational database [28]. The problem will be called "the part problem" since it involves information (identification and color) from "parts" of a warehouse.

As an introduction to the integration case, Figure 1 gives the conceptual description of the elements: *(a)* source entity S_1 has attributes *partId* and *colorId*, *(b)* source entity S_2 has attributes *part* and *colordesc*, and *(c)* target entity T_1 has attributes *pId* and *color*.

The MONIL approach provides a solution for "the part problem" using its framework [27] and its 3-step integration process:

1. The Integration Correspondence Schema Definition.
2. The Programs Generation.
3. The Programs Execution.

2.1 The Integration Correspondence Schema Definition

The correspondence schema establishes the relationship between source data and target data and it must be defined by the user. Figure Figure **2** explains graphically a correspondence schema: *partId* (attribute from S_1) and *part* (attribute from S_2) are sources for the target attribute *pId*, and the concatenation of *colorId* from S_1 and *colordesc* from S_2 provides source data for target attribute *color*.
The MONIL framework offers easy-to-use graphic interfaces to assist the user during the correspondence schema definition.

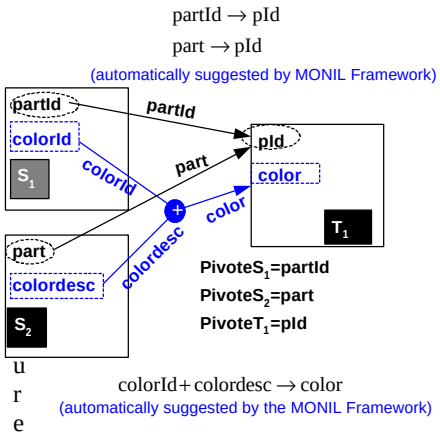

Figure 2. The Integration Correspondence Schema Definition

Figure 3 shows part of the process of definition of the integration correspondences. The functions editor, for example, assists the user to define operations to transform the source attributes and store these as part of the correspondence schema.

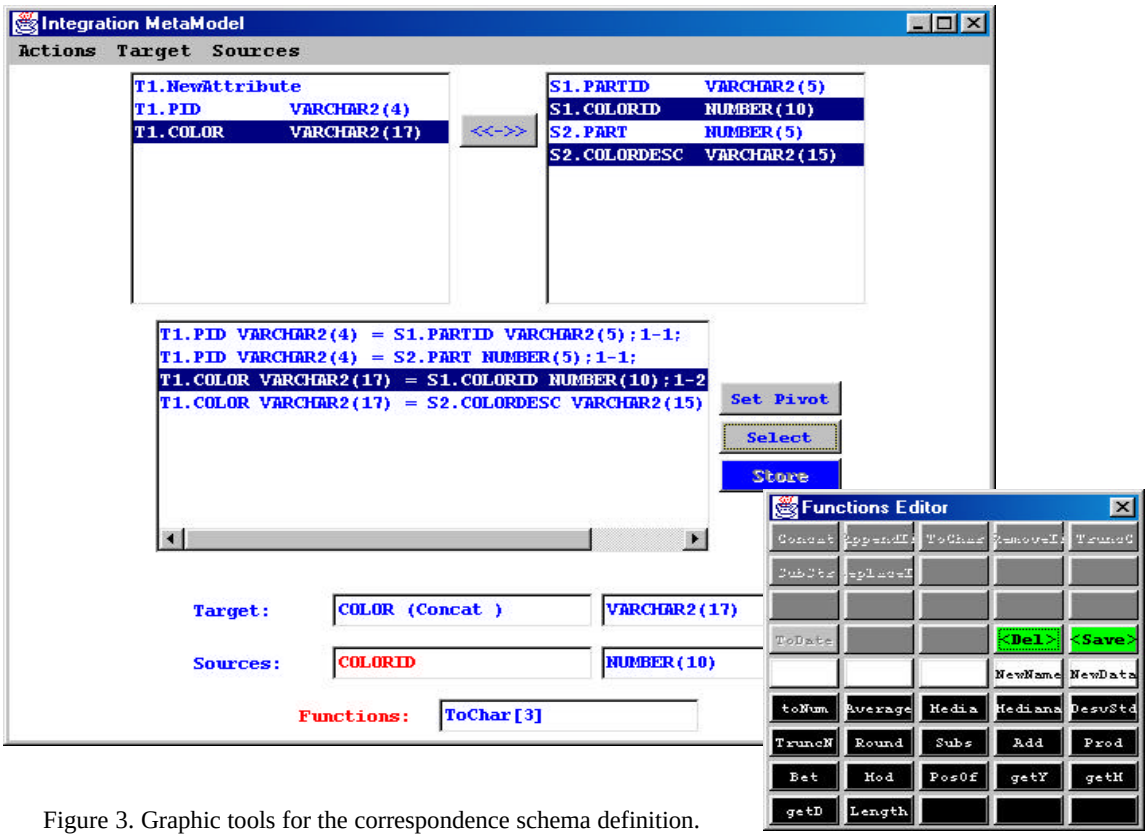

Figure 3. Graphic tools for the correspondence schema definition.

The MONIL framework also includes an algorithm called Integration Correspondence Suggestion or *ICS* to automatically suggest to the user some integration correspondences between source and target data. A previous version of *ICS* was defined in [30] and is based on metadata and the Semantic Proximity Taxonomy[31]. This taxonomy classifies the semantic proximity level between data. *ICS* searches for source and target elements which are semantically related. For "the parts problem", *ICS* automatically suggests all the correspondences needed to solve this integration case, then the user only has to accept these integration suggestions. This MONIL feature minimizes the user effort during the definition of the integration correspondences. To preserve data integrity during the integration process, the integration correspondence schema requires the definition of representative attributes, called **pivots**, for each source-target relationship. Pivots are automatically suggested by the *ICS* for every integration case and represent the relations between sources and targets. Finally, when the user finishes with the correspondence schema definition, it should be stored into the Integration Metamodel [27] or *IM* which is the MONIL general repository that stores all the information used for the integration process.

2.2 Programs Generation

Figure 4 shows the MONIL program called "POLITET1" that specifies the required integration operations to solve the "the parts problem". The program was generated automatically using the stored correspondence schema definition of Figure Figure **2**.

Besides the automatic process to generate MONIL programs, the framework offers text edition tools for users that prefer typing (manually) their programs. For those programs that were not automatically generated, the framework includes a compiler to make a complete code analysis to find and correct syntactic and semantic errors Every MONIL program has a 2-section structure: **heading** and **body**. The program's heading describes every element that participate in the integration process. These elements are called **Integration Units** or *IU*. MONIL distinguishes two *IU* categories: High Level Units and Specific Units and uses two different high level and specific units: **source units** which are data providers and **target units** which are data receivers.

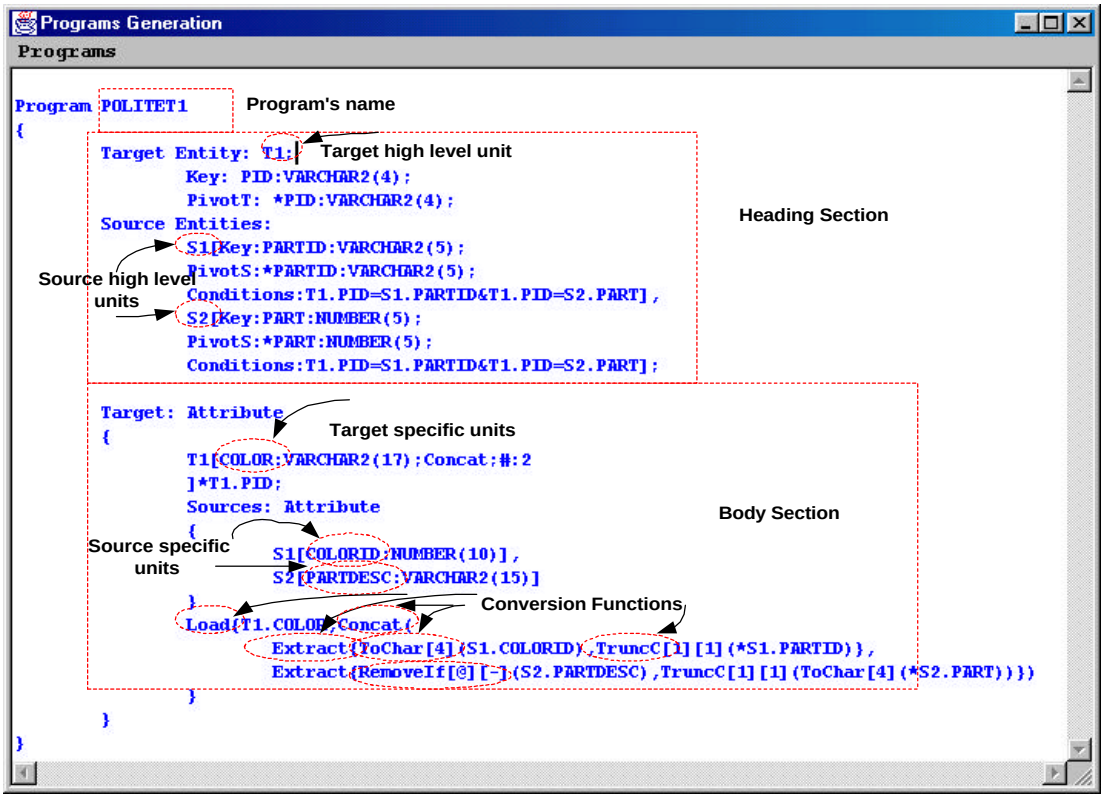

Figure 4. The MONIL program POLITE1 automatically generated using a correspondence schema definition

In the "POLITET1" program code (Figure 4), S_1 and S_2 are the high level source units; T_1 is the high level target unit, *partId*, *colorId*, *part* and *colordesc* are the specific source units; and *pId* and *color* are the specific target units. The program's body describes the required process or the Integration Procedure to transform source data into target data.

For "the parts problem", two different integration processes must be done to solve the problem:

1. The integration process for the correspondences *partId->pId* and *part->pId* is: *(a)* Extract source attributes *partId* and *part*, *(b)* Modify the structure (format) of the source attributes, and *(c)* Load extracted data into the target attribute *pId*.
2. The integration process for the correspondences *colorId+colordesc->color* is: *(a)* Extract separately source attributes, *(b)* Modify the source attributes formats (e.g., character "-" is removed from attribute *colordesc*), *(c)* Concat attributes *colorId* and *colordesc*, and *(d)* Load resultant data into target attribute *color*.

The operators proposed by the MONIL Language to transform and manipulate source and target data are the Conversion Functions or *CF*. The CF format is given by:

FuncName$[parameter_1]...[parameter_N](data)$

where N represents the total number of parameters that each conversion function requires, and data, is the specific integration unit that is modified by *FuncName*.

In Figure 4, some of the used *CF* are: *Load*, *Extract*, Removeif, *TruncC*, *ToChar*, etc. Currently, MONIL Language definition has 50 conversion functions available classified into 5 categories: *(a)* Text/String, *(b)* Extract/Load, *(c)* Structure, *d)* Numeric, and *(d)* Date/Time.

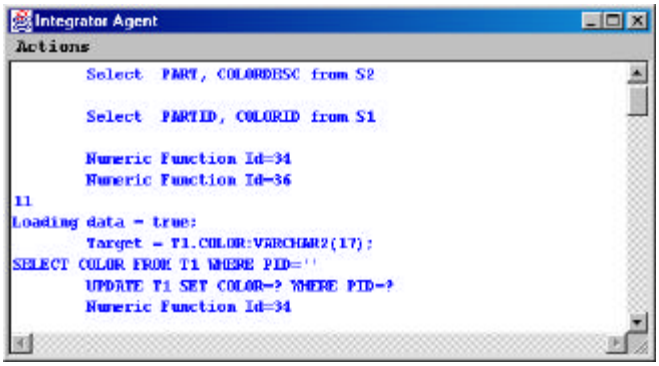

Figure 5. Java Language code and JDBC commands generated during the execution of the POLITE1 program.

The program generation phase concludes when the new program is stored into the *IM* to be available for the execution phase. A stored program could be executed every time that is required by the user.

2.3 Programs Execution

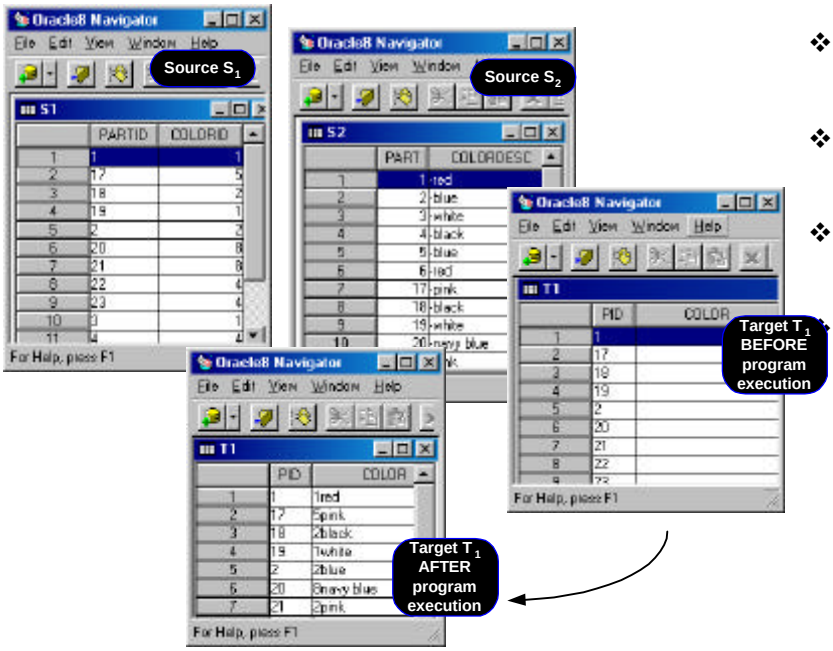

Figure 6. Elements of the "parts problem" before and after MONIL Program execution.

The MONIL program execution is the last step for the integration process, and it is performed by a framework tool called the Integrator Agent or *IAg*, which searches and executes the MONIL programs selected by the users translating them into both Java language and JDBC technology [32], [33]. Figure 5 shows a small part of the code generated when the program "POLITET1" was executed.

The *IAg* execution ends successfully, when every defined source data is transformed and loaded into its related target data. Figure 6 shows "the parts problem" data participants before and after the program "POLITE" execution: *(a)* the target unit T_1 before the "POLITET1" execution, and *(b)* the target unit T_1 after the integration program execution. Both data sources S_1 and S_2, stay without changes during the integration process.

3. Conclusions and Future Work

A Data Integration Language called MONIL was presented as an alternative for solving data integration problems. MONIL is an expressive programming language embedded in a flexible framework in which data integration operations can be easily expressed. MONIL Language has been successfully tested using multiple sources with different heterogeneity levels.

The main contributions of MONIL Language are:

- A formal Programming Language Definition based on Metadata to express the data integration process.
- An algorithm to automatically suggest integration correspondence between source and target data.
- A MONIL Framework specially designed to develop, store and execute MONIL programs.
- A set of built-in Conversion Functions that supports: (a) manipulation of source data units, and (b) alterations of the structure of target data units.

Currently, our work is focussed on extending the scope of the integration suggestion algorithm to automatize many other integration process activities.

References:

[1] Ullman, J. Information Integration using Local Views. In Proc. of *The 6th Int. Conf. on Database Theory (ICDT'97)* Vol. 1186 of Lecture Notes in Computer Science Springer-Verlang, (1997) pp. XIX-XL.

[2] Wiederhold, G. Mediators in the Architecture of the Future Information Systems. *IEEE Computer*, 25(3), (1992) 38-49.

[3] Kim, W., Choi, I., Gala, S. & Scheevel, M. (1993). On Resolving Schematic Heterogeneity in Multidatabase Systems. Distributed and Parallel Databases.

[4] Dayal, U. & Hwuang, H. (1986). View Definition and Generalization for Database Integration in a Multidatabase Systemas. *Proc. IEEE Workshop on Object-Oriented DBMS*.

[5] Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman J.,Widom J.: The TSIMMIS Project: Integration of Heterogeneous Information Sources. *In Proc of IPSJ Conference*, (1994)7-18.

[7] Bergamaschi, S., Castano, S., Beneventano D. & Vincini, M. (2001). Semantic Integration of Heterogeneous Information Sources. *Special issue on Intelligent Information Integration Data and Knowledge Engineering*, 36(1), 215-249.

[8] Bergamaschi, S., Castano, S., De Capitani di Vimercati S., Montanari S. & Vincini M. (2001). A Semantic Approach to Information Integration:

The MOMIS Project. *Special Issue on Intelligent Information Integration Data and Knowledge Engineering*.36(1),2-15-249.

[9] Litwin, W., Mark, L. & Roussopolos, N. 1990. Interoperatibility of Multiple Autonomous Databases. *ACM Computing Surveys*, 22(3), 267-293.

[10] Arens, Y., Chee, C.Y., Hsu, C.N. & Knoblock C.A. 1993. Retrieving and Integrating Data from Multiple Information Surces. Int. *Journal of Intelligent and Cooperative Information Systems*. 2(2), 127-158.

[11] Lu, J.J., Moerkotte, G., Shue, J. & Subrahmanian, V.S. 1995. Efficient Maintenance of Materizlized Mediated Views. Proc. *ACM Sigmod Symp. on Management of Data*, 340-351.

[12] Zhuge, Y., Garcia-Mollina, H., Widom, J. & Hammer J. 1995. View Maintenance in a Warehousing Environment. *ACM Sigmod Symp. on The Management of Data*, 316-327.

[13] Zhou, G., Hull, R., & King, R. 1996 Generating Data Integration Mediators that use Materialization. *Journal of Intelligent Information Systems* 3:2/3, Kluwer Academic Publishers, May, 199-221.

[14] Gupta, H. & Mumick, I.S. (1999). Selection of Views to Materialize under a Maintenance-Time constraint. *International Conference on Database Theory*.

[16] Papakonstantinou, Y., Garcia-Molina, H.,Widom J.: Object Exchange Across Heterogeneous Information Sources, *IEEE International Conference on Data Engineering* (1995) 251-260.

[17] Adali, S., Candan, K., Papakonstantinou, Y.,Subrahmanian, V.: Query Catching and Optimization in Distributed Mediator Systems. In Proc. of *ACM SIGMOD Conf. On Management of Data* (1996).

[18] Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J. & Widom, J. (1997) The TSIMMIS Project: Integration Of Heterogeneous Information Sources. *Jornal of Intelligent Information Systems*, 8(2),117-132.

[19] Jarke, M., Jeusfeld, M.A., Quix, C. & Vassiliadis, P. 1998 Architecture and Quality in Data Warehouses. Proc. of *the 10th conf. On Advanced Information Systems Engineering (CAiSE'98)*, 1413, 93-113.

[20] Bergamaschi, S., Castano, S., Vincini, M., Beneventano D. (1999) Intelligent Techniques for the Extraction and Integration of Heterogeneous Information. *Workshop on Intelligent Information Integration, (IJCAI99)* .

[21] Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., Rosati R.: A Principled Approach to Data Integration and Reconciliation in Data Warehousing, In Proc. of *the International Workshop on Design and Management of Data Warehouses (DMDW'99)* Vol. 19 (1999).

[22] Friedman, M., Mavy, A., & Millstein, T. (1999). Navigational plans for Data Integration. *16th National Conference on Artificial Intelligence (AAAI'99)*.

[23] Jarke, M., Quix, C., Calvanese, D., Lenzerini, M., Franconi, E., Ligoudistiano, S., Vassiliadis, P., Vassiliou Y. Concept based Design of Data Warehouses: The DWQ Demostrators, in Proc. of *the ACM SIGMOD Int. Conf. On Management of Data* (2000) 591.

[24] Beneventano, D., Bergamaschi, S., Guerra, F. & Vincini, M. 2001. The MONIS Project Approach to Information Integration. *IEEE and AAAI International Conference on Enterprise Information Systems (ICEIS01)*.

[25] Kimball, Ralph.: *The Data Warehouse Toolkit*. 2nd. edn. John Wily and Sons, Inc. 1996.

[26] Inmon, W.H., Terdeman, R.H. & Imhoff, Claudia. 2000. *Exploration Warehousing Turing Business into Business Oportunity*. John Wiley and Sons, Inc.

[27] Larre, M., Torres-Jiménez J., & Morales, E.. 2001 Data Integration with MONIL, Metadata and Correspondence Suggestions. *3er. Encuentro Internacional de Ciencias de la Computación (ENC'01)*, (2)623-632.

[28] Date, C. D. *An Introduction to the Database Systems* (Introduction to Database Systems 7th Edition). Addison-Wesley Pub. Co. 1999.

[29] Hopcroft, J., Ullma, J.: *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Pub. Co. 2^{nd} Edition (2000).

[30] Larre, M., Torres, S., Torres, J., Morales, E. (2000). Un Algoritmo para la Integracion de Datos basado en el Descubrimiento de Relaciones. En Procs. del *7° Congreso Internacional de Investigaciones en Ciencias Computacionales CIIC00* 263-274.

[31] Kashyap, V., Sheth, A. Schema Correspondences between Objects with Semantic Proximity. *Technical Report DCS-TR-301*, Department of Computer Science, Rutgers University (1993).

[32] Reese, George. 1997. *Database Programming with JDBC and Java*. 2nd. Edition, O'Reilly Associates.

[33] White, S., Fisher, M., Cattel, R., Hamilton, G., Hapner M.: *JDBC API Tutorial and Reference*, Second Edition. Addison-Wesley, Pub. Co.1999.