MONIL Language, an Alternative for Data Integration

Monica Larre, Jose Torres-Jimenez, Eduardo Morales, Juan Frausto-Solis, Socrates Torres

ITESM Campus Cuernavaca

Av. Paseo de la Reforma 182-A Col. Lomas de Cuernavaca

{monica.larre, juan.frausto, socrates}@itesm.mx

Instituto Nacional de Astrofisica Optica y Electronica

Luis Enrique Erro 1, Sta. Ma. Tonantzintla, 72840 Puebla

emorales@inaoep.mx

Abstract

Data integration is a process of retrieving, merging and storing of data originated in heterogeneous sources of data. The main problem facing the data integration is the structural and semantic heterogeneity of participating data.

A concern of research communities in computer sciences is the development of semi-automatic tools to assist the user in an effective way in the data integration processes.

This paper introduces a programming language called MONIL, as an alternative to integrate data by means of design, storage and program execution. MONIL is based on the use of meta-data, conversion functions, a meta-model of integration and a scheme of integration suggestions. MONIL offers to the user a dedicated work environment with built-in semi-automatic tools supporting the integration process in three stages.

Key words: data integration, integration language, databases, metadata.

1. Introduction: Data integration

Globalization establishes technologies and new parameters for the required information to make decisions. To satisfy the requirements of the users, with the current information that arises from heterogeneous sources, has led the researchers in computer sciences to develop effective tools to get and manage the information they need. The handling of information originated at heterogeneous sources presents a complex task: data integration.

Data integration (DI) can be defined as "***retrieval and merging of data originated at heterogeneous sources to be stored at a selected target***".

Figure 1.1 shows a data integration system ([1], [2], [3]), and it is defined in terms of two elements: *wrappers* and *mediators. Wrappers*, or the extractors, are the elements in charge of retrieving the required information from each one of the sources that participate in the integration process. *Mediators* are the elements in charge of manipulating the source data to convert them into target data. In several projects of data integration (e.g., [4], [5] and [6]), mediators are considered the main elements of the integration process.

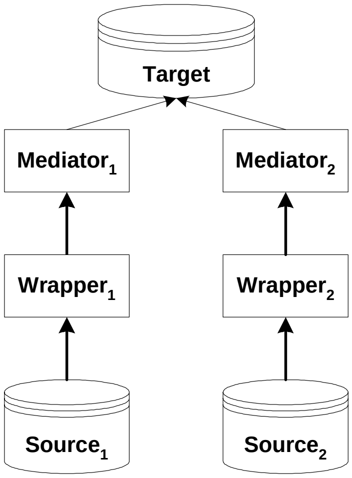


Figure 1.1 Generic architecture of a data integration system.

DI is complex due mainly to the semantic and structural heterogeneity of the participating data. Structural differences come from the implementation details, including differences in the hardware platforms, database models and programming languages. Semantic heterogeneity occurs, for instance, when different names are employed to represent the same information or when the same names represent different pieces of information.

Figure 1.2 shows an example of data integration where the target attribute *Sales* is integrated from three attributes originated at different sources: *nameProd* from Source$_1$, *descripP* from Source$_2$ and *Sales* from Source$_3$. All three sources attributes have different formats (structural heterogeneity) and should be converted into the format of the target attribute. On the other hand, reference attributes, used to equal both registers source and target (indicated as α_1, α_2, α_3 and ς in Figure 1.2) have different names but make reference to the same information (semantic heterogeneity). Conflicts arising from the heterogeneity between the source and target data should be solved as part of the data integration process.

Because data warehouses have been increasing, semi-automatic tools for effective exploitation of multiple data sources to fill up the data warehouses, has become increasingly relevant ([7], [8], [9], [10], [11]).

This paper presents the MONIL[1] language as an alternative to solve DI problems. The first antecedent of the MONIL language appears in [12], where integration problems are solved on the conceptual level. Later on, integration has been sought through the application of genetic programming, as it is shown in [13], but from [14], this language acquires most of its current features, as it appears on [15] and [16]. The structure of this paper is the following: Section 2 presents proposals existing in the literature about the data integration problem. Section 3 contains the features and formalisms of the MONIL language. Sections 4 and 5 describe the work environment developed for MONIL and the proposed integration process. Section 6 presents examples of integration problems and MONIL programs that solve them. Section 7 presents main conclusions of the work and the future work.

[1] Metadata and Object Integration Language

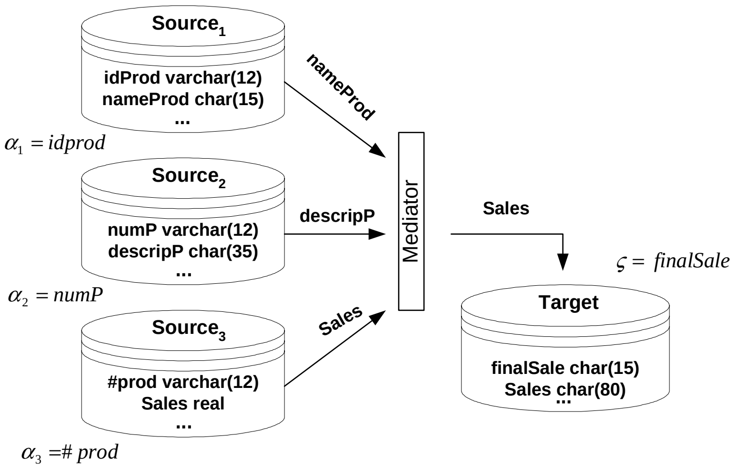


Figure 1. 2. An example of DI problems.

2. Related Work

The DI problem is not new. It has been studied by different communities of researchers, especially by the database communities ([17], [18], [19], [20]) and artificial intelligence ([21], [22], [23], [24], [25]).

The computer science community has proposed solutions to the data integration problems from different perspectives. Most of those proposals are relevant and interesting due to their contributions. Two research trends are the most important because they support one of two approaches: a) the semantic or declarative and b) the structural or procedural. The semantic approach (e.g., [26], [27], [28], [29], [30], [31], [32], [33], [34]) considers the data integration problem through a common data model built up from the conceptual schemes of each one of the participating data sources. In addition, the semantic trend employs a data language for the information exchange between the common data model and the sources that participate in the integration process.

On the other hand, the structural approach (e.g., [35], [36] and [37]) considers only individual, specific and pre-established needs of the different data sources, without taking into account the notion of a global integration scheme. For required information a specific consultation must be specified in terms of the source data.

More recently, new approaches appear that present similar concepts to the structural and semantic trends, but their efforts are mainly oriented to solve the data integration of big Databases, Data Warehouses, Distributed Databases and Web Databases.

These approaches are: GAV (*Global as View*) and LAV (*Local as View*). To carry out the integration, GAV and LAV propose the following process. Each participating data source is manually modeled by the users as a set of relationships called the mediator scheme. Users carry out the search of information they need in terms of the mediator scheme and its attributes and not in terms of the data sources. The relationships of the mediator scheme are virtual, which means that they are not stored anywhere. To complement the information of the mediator scheme, GAV and LAV add descriptions of the participating sources to permit the access to each one of them, although each trend makes this description in a different way.

GAV defines the relations of the mediator scheme in terms of sources and, in a similar way to the structural trend; the consultations to obtain required information from each data source are specified in terms of the source and not of the mediator scheme. Among the most relevant work of the GAV trend, there are: [38], [4], [24], [39], [40].

On the other size, the LAV proposal establishes that source descriptions should be defined starting from the unified global representation of the mediator scheme. Some of the proposals of the LAV trend are: [41], [42], [22], [23], [43], [8].

The mediator scheme used by GAV and LAV is built in virtual view. The traditional ones (e.g., [44], [45], [46], [47], [48], [49]), propose to build the mediator scheme in integrated virtual views. Consultations carried out within the mediator scheme are formulated again to work in a specific way with each one of the sources they are composed of. More recently, the use of materialized views (e.g., [50], [51], [5], [52], [53], [54]) to build up a mediator scheme has called more attention, especially due to situations where materialization is preferable to the use of virtual views: e.g., when the connectivity of the networks is unpredictable, when the consultation response time is critical or when it is cheaper to build up materialized views and keep them than to calculate the consultation every time it is necessary.

3. MONIL Language Proposal

MONIL has been exclusively designed to solve data integration problems. This paper shows instances of integration where data sources and targets are structured, however, MONIL is not restricted to solve DI problems exclusive of structured data. MONIL is an expressive programming language, built on context free grammars and based on the use of metadata, a set of conversion functions, an integration model and an automatic scheme of integration suggestions. To achieve functionality, MONIL is immersed in a work environment formed by a set of automatic and semi-automatic tools that assist the user in the design, development, storage and execution of integration programs solving the data integration problems. The elements of the environment are related among them through an integration process proposed by MONIL.

The proposed integration process points out three basic stages:

1. Definition of the corresponding integration scheme.

2. Generation of integration programs in MONIL

3. Program execution.

Solving the problem of DI by stages maintains flexibility, and allows the DI process to interfere as least as possible with the day-to-day operation of the participating information systems.

MONIL includes the main GAV advantages:

- Incorporates or eliminates dynamically DI process elements.

- Excludes descriptions of elements that do not participate in a direct way in the integration.

MONIL incorporates from LAV the concept of a corporate model through the so called integration meta-model. MONIL may be considered among the structural and semantic data integration problems solution tools, since it solves cases in both types of conflicts.

In addition to its advantages, MONIL introduces some new concepts to solve DI problems:

- The **integration units** are used to describe the elements participating in the process.

- The **pivots** to relate the source and target elements being integrated and to keep up the integrity of the data during integration.

- The **conversion functions** to solve conflicts issuing from the structural and semantic heterogeneity among the data.

- The use of **metadata** as the information supply sources for the operation of the three phases of the MONIL integration process.

- The **automatic scheme of the integration correspondence** to reduce the user effort, based on the semantic proximity concepts [55].

To make easier the MONIL language operation, a special semi-automatic work environment has been designed to develop, store and execute integration programs in the MONIL language.

The integration process is supported by the work environment defined by MONIL to systematize DI problems solution. However, and in spite of its features, not all tools in the MONIL work environment carry their tasks in a fully automatic way. User's participation is especially required during the problem definition, an action that is performed at the first stage of the integration process.

3.1. MONIL language concepts

As a fundamental part of its conceptual definition, MONIL includes four elements:

- Integration units
- Pivots
- Integration conditions
- Conversion functions

3.1.1. Integration units

Integration units represent all the source and target elements that participate in the integration process. To manipulate them, the integration units are represented by a set of symbols that form part of the formal language specification.

MONIL distinguishes two categories: HLIU and SIU.

The HLIU (high-level integration units) represented in the DI process the elements at a higher hierarchy or abstraction level, for example, databases, tables, files and classes. The SIU (specific integration units) are indivisible elements that participate in the DI process, for example, attributes and objects.

MONIL permits the use of several abstraction levels in the high level units. The abstraction level is represented by a super index associated with each unit, for instance, for the case of the *entity-relation* model [53], a database that participates in the integration process will have abstraction level zero and each one of the tables forming it will have level one.

MONIL uses two kinds of high abstraction level integration units: HLSIU and HLTIU.

The HLSIU (high level source integration units) are the integration data suppliers. The HLSIU are represented by the symbol S_i^ω, where i represents the *i*-th source participating in the integration, and ω, represents the source abstraction level. Lower ω values represent higher abstraction levels. Thus, for example, S_0^0 represents an entire database and S_0^1 represents only a database table.

The HLTIU (high level target integration units) denoted by T^ω, are the receiving integrated data units, where ω represents (as for the sources), the abstraction level of the HLTIU. Again, T^0 will be the receiving database and T^1 a table of the database.

Moreover, there is another classification for integration units. The specific integration units or SIU, which are the attributes directly involved in the integration process, and they are also classified in source and target.

The Source Specific integration Units (SSIU) or σ are the source elements participating in the DI process

The Target Specific Integration Units (TSIU) τ are the target elements received by the integrated data.

By definition, the SIU must belong to a HLIU. The relationship between the SSIU and the HLSIU is expressed by means of:

$$S_i^{\omega} = \{\sigma_{i,1}, \sigma_{i,2}, ..., \sigma_{i,s}\}$$ **(1)**

Where S_i^{ω} is the *i*-th source that participates in the integration process with an abstraction level given by ω, and s is the total number of $\sigma_{i,j}$ that belongs to S_i^{ω}. The value of s does not necessarily represent the total number of attributes of S_i^{ω}, i.e.: $s \leq attributesOf(S_i)$. This is a GAV feature of MONIL that permits to model only the source participating elements.

As for the sources units, the relationship between the TSIU and HLTIU is expressed by:

$$T^{\omega} = \{\tau_1, \tau_2, ..., \tau_p, t_1, t_2, ..., t_q\}$$ **(2)**

And due to the MONIL target orientation, T^{ω} represents the target integration unit of ω level, p represents the number of TSIU participating in the integration (there should be at least one), q is the number of target elements (that are only auxiliary elements, not SIU) but which are used in the DI process. The $p+q$ value does not necessarily represent the total of attributes that form each T^{ω}.

3.1.2. The pivots

A pivot is a set of one or more attributes, established by the user, that represents the HLIU that participate in the integration process. An example may be the primary keys or candidate keys of the tables of a relational database model. Pivots may be or may be not SIU.

For the MONIL definition, the objectives of the pivots are:

- To establish a relationship among the participating HLIU and the HLTIU.
- To maintain information integrity during the complete integration process.

MONIL assumes that the elements designated as pivots are representative and with unique values and without missing values. However, in most cases, if the pivots do not meet these requirements, the integration process of MONIL is not interrupted. This occurs with data integration cases that do not have an established structure.

MONIL establishes a rating for its pivots by the number of SIU that form them as:

- Simple, with only one SIU
- Compounded, formed by more than one SIU.

By their origin, pivots are classified as:

- Source (α)
- Target (ξ)

The value of a source pivot α_i represents the source unit S_i^{ω}, and is expressed as:

$$\alpha_i = \bigoplus_{j=1}^{A} (\sigma_{i,j})$$ **(3)**

Where *A* is the number of $\sigma_{i,j}$ elements which in concatenating with the operator $\oplus$ form the source pivot α_i

The target pivot value ξ identifies the target unit T^{ω}, and is expressed as:

$$\xi = \bigoplus_{j=1}^{Z} \left(x_{i,j} \right)$$

(4)

Where the items $x_{i,j}$ are the elements which form the pivot. They may be an integration unit τ or an element not TSIU t, and, finally, Z will be the number of elements that forms the pivot ξ.

The relationship between ξ and α is a $1-to-N$ for any $N \geq 1$, i.e., each target pivot ξ may be associated with one or more source pivots α_i.

3.1.3. Integration conditions

These are the integration process rules, represented by ϑ, and their goal is to select source data susceptible to be used in each integration case.

The structure of an integration condition is based on the pivots values established by the user for DI. For each relation HLIU *source-target*, it is necessary to define a condition to regulate the specific integration process. For example, for an integration case where *N* data sources are participating, there should be *N* integrating conditions.

For data without structure, the integration conditions perform an even more important role, the one of becoming the guide of the integration process.

Integration conditions are built establishing either the correspondence between the source-*target* pivots or defining the correspondence between the *source-target* pivots. The structure of ϑ, is represented by:

$$\vartheta : \xi = \alpha_i$$

(5)

Where it is established that the value of the source attribute, represented by α_i participates in the DI process if and only if there is any value of the target pivot ξ equals to the source pivot value α_i.

The ϑ's are automatically defined by the MONIL environment at the beginning of the integration process, and, by default, they have the format defined by (5). The automatic generation of ϑ is performed on the integration correspondence basis between the pivots elements defined by the user

If data without structure is used, the user will be responsible for the definition of the integration conditions using the editing tools offered by the MONIL environment.

3.1.4. Conversion functions

Conversion functions (represented by F) are the MONIL transformation operators. Their purpose is to solve structural and/or semantic conflicts between the source and target data. Conversion functions may be selected by the user once the integration relations are established. Some of these functions are suggested automatically by the MONIL environment, however, the user may eliminate or add functions deemed necessary to solve the particular integration problem. The scope of the conversion functions is wide, some of them are even capable of modifying and creating physical structures of HLTIU.

MONIL offers a taxonomy from which the user may build more complex functions permitting to solve different integration cases, due to the recursive nature of most functions.

According to its scope, this taxonomy classifies its functions in three groups:

1. Converting functions (Fx) work exclusively with source elements.
2. Basic functions (Fec) work with source and target elements.
3. Construction or structural functions (Fs) alter or create HLTIU structures.

A converting function Fx description is as follows:

$$Fx = \{Fx([p_1],[p_2]..[p_f](\sigma_{i,j}))\} \text{ for } \tau \neq \sigma$$

(6)

Where Fx is any converting function that uses $f - number$ of parameters p to solve an existing conflict between $\sigma_{i,j}$ and τ. For those cases when the mapping of the target source does not require any transformation $\sigma_{i,j} = \tau$, (6) is reduced to:

$$Fx = \sigma_{i,j}$$

(7)

The definition of the parameter p is recursive in most cases, i.e., the parameters of a function may be defined by another function and this, in turn, can have as a parameter a function, and so on:

$$p_f = \begin{Bmatrix} data \\ Fx \end{Bmatrix}$$

(8)

This feature offers built-in possibilities to obtain very complex functions to solve particular (rare) integration problems.

The Fx group is the most numerous, it is evaluated from left to right and its functions are classified considering the type of data on which they operate. The classification is: *i*) Functions for text or chains, *ii*) Functions for numbers, *iii*) Functions for dates and time, and *iv*) Functions for equivalence.

The second MONIL group of functions is the group of basic functions or Fec. These functions are two:

- The ε function: extracts data from sources.
- The ι function: loads the integrated information to the targets.

Fec syntax does not use parameters, thus, the minimum expression for the function ε is:

$$\varepsilon = \left\{ \bigoplus_{i=1}^{K} Fx_i \right\}$$

(9)

Where it is shown, that the body of the function is formed by the concatenation of K Fx functions.

ι Function is expressed as:

$$\iota = \tau \left\{ \bigoplus_{i=1}^{N} \varepsilon \right\}$$

(10)

Where τ is the receiver of integrated data and the operator $\oplus$ represents the concatenation operation of N instances of ε. For each τ, N (at least one) supplying sources (SSIU) may participate.

The third group of conversion functions is the construction or structure functions (Fs). These functions permit to modify the structure and even to create new HLTIU structures. Fs do not use parameters. The Fs solve complex integration cases where exist conflicts of schematic discrepancy. There are four Fs: *NewEntity* (ψ) and *NewAttribute* (ζ) called primary functions used to modify the structure of HLTIU; and *NewData* (γ) and *NewName* (δ) called secondary structures used to select the source data to populate the new structures created.

Currently, MONIL definition relies on 32 basic conversion functions, however, and due to the flexibility and expressive power of MONIL, a number of function combinations can be created to solve different data integration problems.

In section 6, examples of conversion functions are shown as part of program codes in MONIL language.

3.2. MONIL compared with other tools

Figure 3.1 shows comparison parameters between MONIL language and other existing proposals in the literature. The compared aspects are:

1. Research trend between the proposals GAV and LAV analyzed in section 2.
2. The possibility to suggest some of the integration correspondences among the participating elements.
3. Consideration of the full integration process, or whether to solve only the data integration in a partial way.
4. The proposal as a language of commands without associated tools or, if it is a formal programming language, with the entire environment to edit, compile and store programs.
5. Implementation of the proposal.
6. Specific (S) or general area (G) of the tool with respect to integration problems.

As it can be appreciated, MONIL features cover the standards of all aspects pointed out in the comparison, showing a better coverage in the integration suggestions, implementation, full integration process and general applications.

Param	TSIMMIS	MOMIS	DWH	NPDI	COM	MONIL
1	GAV	LAV	LAV	LAV+	GAV	GAV+

				GAV		LAV
2	NO	NO	NO	NO	NO	YES
3	PARTIAL	PARTIAL	PARTIAL	PARTIAL	PARTIA, FULL	FULL
4	C	C	C	C	-	L
5	YES	YES	NO	NO	YES	YES
6	G	S	S (DW)	S (Web)	S/G	G

Figure 3.1 MONIL vs. other data integration proposals.

3.3. Formal definition of MONIL language

MONIL language is represented as a context free grammar[56]:

$$\mathrm{G}_{\mathrm{M}} = (V, F, P, I) \qquad \textbf{(11)}$$

Where:

$$V = \{v, E, C, S, id, T, Fx, F_{EC}, F_S, \vartheta, \alpha, ID, ...\}$$

$$F = \{\{,\},=,;,:, PivotT, PivotS, SourceEntities, ...\}$$

$$P = \{112\_productions\}$$

$$I = P_{MONIL}$$

A subset of productions and 58 terminal symbols describing MONIL language are shown in the production list BNF[56], some of these are presented in the next list:

$$< P_{MONIL} ::=< v > \{< E >< C >\} < v >::=< id >$$

$$< E >::=< T >;< \xi >;< S >$$

$$< T >::= TARGET \ < t >< idU >$$

$$< \xi >::= PivotT :< dPIVOT >$$

$$< dPIV >::=< enPiv >< idU >:< tDAT >< dPIV'> < dPIVOT'>::= \theta \,|,< dPIVOT >$$

$$< S >::= SE :< idU > [< KEY >;< \alpha >< \vartheta >]S' < S'>::= \theta \,|,< S >$$

$$< \alpha >::= PivotS :< dPivote > < \vartheta >::= Conditions :< dato\,\vartheta >$$

$$< dato\,\vartheta >::=< idU >=< idU >< dato\,\vartheta'> < dato\,\vartheta'>::= \theta \,|< and >< dato\,\vartheta >$$

$$< C >::= t\arg et : attribute\{< \tau >< \sigma >< \rho >\} ...$$

From $< P_{MONIL} ::=< v > \{< E >< C >\}$ it is clear that the structure of a MONIL program has three main elements:

$$P_{MONIL} = vEC$$

(12)

Where v is a unique valid identifier that represents the program, E the heading, and C the body.[2]

The heading of a program is given by:

$$E = T\xi \bigoplus_{i=1}^{N} [S_i \alpha_i \vartheta_i]$$

(13)

Where T is the HLTIU that will receive the integrated data by means of the target pivot ξ, S_i is the *i*-th of the N participating HLSIU, α_i is the pivot representing the source S_i, and ϑ_i is the *i*-th integration condition that controls the information between T and S_i, $\oplus$ is the concatenation operator.

The body of a MONIL program combines the declarative information of source and target SIU with the procedural information of the so-called integration procedures.

The body of a program describes the process that will be carried out to enable the data from S_i to be integrated in T. This body can be represented by:

$$C = \tau \bigoplus_{i=1}^{N} \left[\bigoplus_{j=1}^{M} \sigma_{i,j} \right] \rho$$

(14)

Where τ is the TSIU objective, NxM is the total of participating SSIU (there are M events of $\sigma_{i,j}$ for each S_i), and ρ is the integration procedure.

An integration procedure is a set of conversion functions that will be applied to $\sigma_{i,j}$, to be converted into τ.

MONIL distinguishes two types of integration procedures:

- Converting procedures ρc
- Generating procedures ρg

The ρc are those that do not cause modification to the target structure, they simply transform the values of the source values before leaving them at their target. They are based on Fec functions.

The generating procedures ρg are based on structural functions and their actions cause transformations in the HLTIU. Specially, these procedures are applied when the integration case under solution involves different levels of abstraction between sources and targets.

Finally, substituting (13) and (14) in (12), we get the full expression of the MONIL program:

$$P_{MONIL} = vT\xi \bigoplus_{i=1}^{N} [S_i \alpha_i \vartheta_i] \tau \bigoplus_{i=1}^{N} \left[\bigoplus_{j=1}^{M} \sigma_{i,j} \right] \rho$$

(15)

An example of a MONIL program is shown in Figure 3.2.

4. MONIL Framework

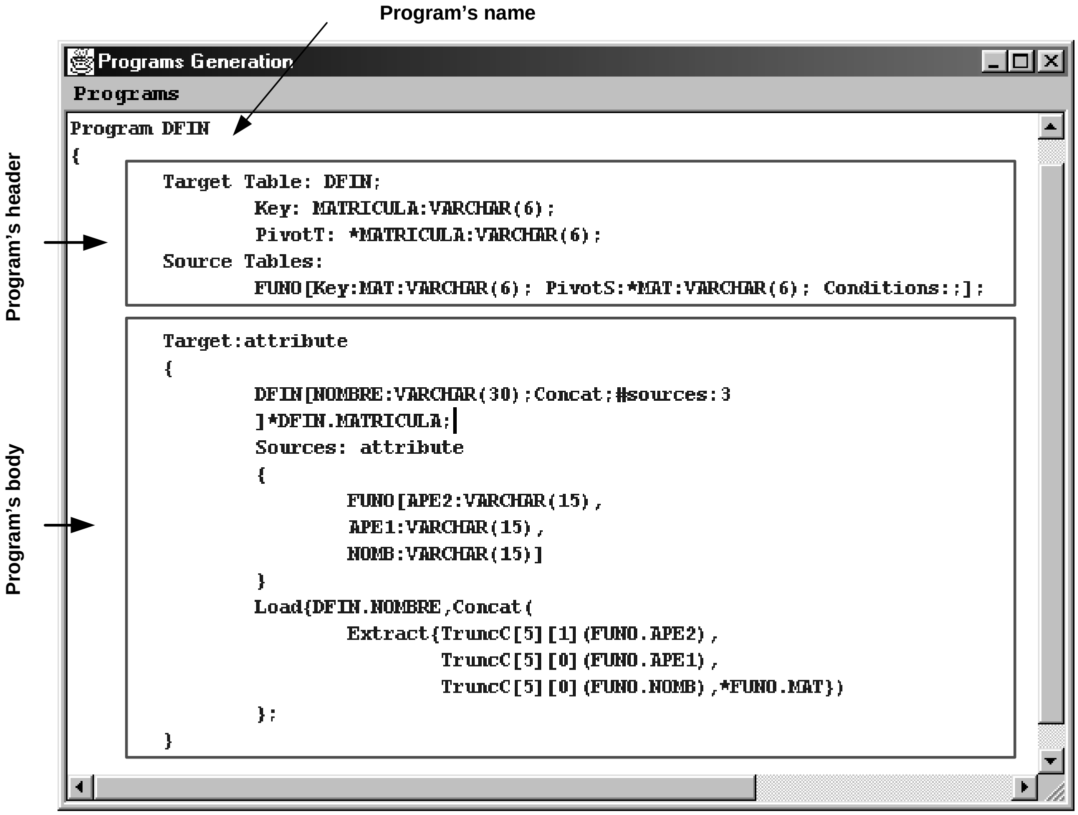
[2] To reduce the complexity of the expressions, language reserved words are omitted.

MONIL framework was designed with the purpose of assisting the user in the generation, storage and execution of the integration programs to solve DI problems. The environment is formed by software components that support and implement the integration process consisting of the three phases, proposed by MONIL.

MONIL architecture is shown in Figure 4.1. The elements formed by the MONIL framework communicate with the user through the interfaces of the elements EC and AgI. The elements communicate between them through different activities defined as part of the MONIL integration process. Each one of these elements is described in more detail in subsequent paragraphs.

4.1. Integration Metamodel (IM)

The integration metamodel is the general repository of information of the MONIL environment and its main function is to store all the data generated and required over the three stages of DI process. The existence of the IM concentrates the global view of the integration process and places MONIL language and its integration process within the LAV paradigm of data integration. The IM is considered as an element of liaison and service of the integration process because it facilitates communication among the work environment modules.



```
Program DFIN
{
    Target Table: DFIN;
            Key: MATRICULA:VARCHAR(6);
            PivotT: *MATRICULA:VARCHAR(6);
    Source Tables:
            FUNO[Key:MAT:VARCHAR(6); PivotS:*MAT:VARCHAR(6); Conditions:;];

    Target:attribute
    {
            DFIN[NOMBRE:VARCHAR(30);Concat;#sources:3
            ]*DFIN.MATRICULA;
            Sources: attribute
            {
                    FUNO[APE2:VARCHAR(15),
                    APE1:VARCHAR(15),
                    NOMB:VARCHAR(15)]
            }
            Load{DFIN.NOMBRE,Concat(
                    Extract{TruncC[5][1](FUNO.APE2),
                            TruncC[5][0](FUNO.APE1),
                            TruncC[5][0](FUNO.NOMB),*FUNO.MAT})
            };
    }
```

Figure 3.2 A MONIL program structure

In the metamodel data reading and writing operations are performed, depending on the process integration time and of the component of the environment that uses it. Maintenance processes of IM are performed by an independent software module (administrator), available to act at any time.

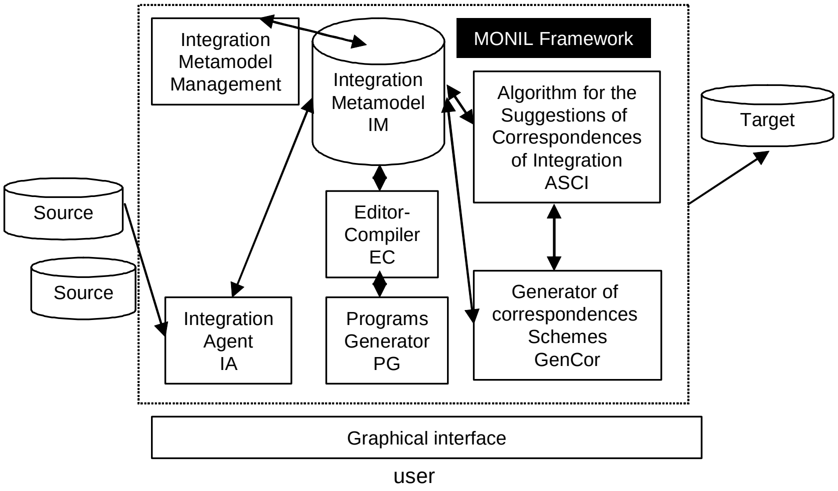


Figure 4.1. The MONIL framework architecture.

IM is built on a relational data model since it is required to be solid, scaling and easy to operate. Specifically, it was built using ORACLE [57], [58] as the database administrator system.

The conceptual architecture of the metamodel appears in Figure 6 showing the most important entities of the model.

ICOS. It stores all the integration correspondence schemes generated either automatically or semi-automatically during the stage 1 of the DI process.

MP. It stores all the programs generated either automatically or manually during the integration process.

CF. It stores the name, code and description of all the conversion functions defined as part of MONIL language.

$S_N, S^{N+1}, T_N, T^{N+1}, \sigma, \tau$ are the tables that store the HLIU and SIU.

4.2. Algorithm for suggestions of the integration correspondences (ASCI)

ASCI is an algorithm that analyzes data, and searches for pairs *source-target* susceptible of being integrated based on structural and/or semantic similarities. ASCI is a tool of MONIL environment of optional use that automatically generates integration correspondences. (See Figure 4.2).

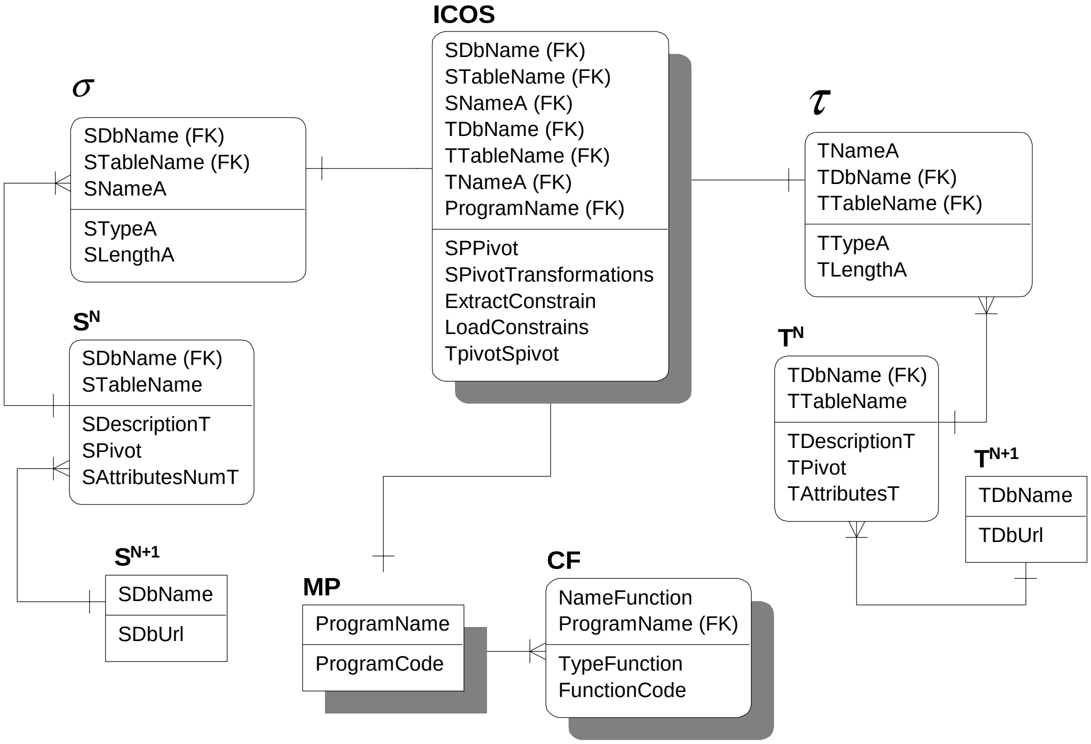


Figure 4.2. The main elements of the integration metamodel.

ASCI work requires reading privileges with respect to HLIU source and target to obtain metadata and/or samples of data to allow it to perform its work. Its use is limited to integration cases that handle data with structure and metadata. ASCI automatic actions use concepts of semantic proximity and produce as a result an initial scheme of integration correspondence which may be discarded, used as such or complemented by the user. ASCI looks for correspondences $1-to-1$ among all the *source-target* SIU participating in the integration process, analyzing their semantic meanings.

The search for correspondence is carried out using the following algorithm:

```
While SIU target
  While SIU source
    $If\,(isSub(\tau,\sigma_{i,j})\,\|\,isSynonym(\tau,\sigma_{i,j}) \quad suggestCorrespondence(\tau,\sigma_{i,j})$
```

The algorithm works on the basis of two methods: *isSub* and *isSynomyn*.

The *isSub* method analyzes the pairs of names of the SSIU and the STIU, and it searches if the name of one unit is a substring of the name of the other unit, then its effectiveness depends on the original designs and on the obtained metadata of the HLSIU.

The *isSynonym* method uses the entity Synonym of the IM to compare the source and target names of the correspondence scheme, with pairs stored in Synonym. If any pair *source-target* appears on the registers of Synonym, the correspondence is suggested. Synonym stores pairs of data names with some degree of semantic proximity. If the user wishes it, using the IM administrator, he/she can initialize the Synonym with those values he/she considers representative of the cases of integration he/she intends to solve. However, it is not necessary since ASCI will automatically increase the contents of the Synonym with each correspondence selected during the first stage of the DI process that is still not included in the Synonym selection. This action increases the number of available synonyms and, therefore, the performance of ASCI

4.3. Editor-Compiler (EC)

EC is a software module that provides the user with tools to design (an Editor) and validate (a Compiler) new integration programs.

Editor includes a text window where the user may enter the programs. When a new program construction starts, the Editor includes in the text window displayed for the user the minimum structure of an integration program, as reference.

Operationally, the Editor offers 4 options:

1. To create a new program.
2. To edit existing program.
3. To compile a new or modified program.
4. To save a new or modified program.

It is possible to create a new program or to edit and existing program. The compilation phase reviews the codes of the new or modified program to verify whether it complies with all the syntactic and semantic rules of the MONIL language. The option to save a program will be activated only after the compilation option is used and when it is successful in order to guarantee that the new or modified programs are valid.

4.4. Generator of correspondence schemes (GenCor)

It is a component of the software that provides functionality to the generation of integration correspondence schemes. Functions offered by GenPro are:

- Parameters definition. It permits to select the HLIU that participate in the DI problem (databases, tables, etc.).
- Automatic generation of correspondence suggestions. It calls for the execution of ASCI to look for and suggest integration correspondences.
- Manual definition of integration correspondence. It permits the user to select integration correspondences.
- Validation of the initial scheme of correspondence. It permits the user to validate the conversion functions necessary to solve conflicts between the source and target data, i.e., when $\tau \neq \sigma$.
- Storage of the final scheme of correspondence, once it is validated.

4.5. Generator of Integration Programs (GenPro)

It is a software component that automatically generates IP in MONIL language, using stored correspondence schemes. The automatic action of GenPro offers two possibilities:

- To generate only one IP based on a specific correspondence scheme.
- To generate all the IP using all the correspondence schemes stored in the integration metamodel.

Construction of an IP is a progressive process of symbols concatenation ruled by the MONIL syntax. The IP contains reserved words and specific data of the integration case, defined in the scheme of the base correspondence. This construction option produces IP syntactically correct. However, since the data to build an IP derive from the correspondence schemes defined by the user, any error in their specifications will be incorporated to the generated IP.

In addition to the automatic option, and if the user decides to design a program manually, GenPro transfers the control over to the Editor-Compiler to edit, compile and store the user´s program.

4.6. Integrating Agent (AgI)

The AgI is a semi-automatic module that executes IP stored in IM. AgI actions require a connection with the source and target HLIU of the data. It uses reading privileges to retrieve data from the HLIU, writing privileges to load integrated data to the target HLIU. Its action starts when the user selects a MONIL program. The program can add integrated data ($Append\_F$) to the existing data at the target, if in the program code appear the procedure ρc, and, on the contrary, it can replace the target data with the new data ($\mathrm{Re}\,place\_F$) if in the code there are ρg procedure.

AgI action by default is ρc. The execution of an IP is shown by the following algorithm:

```
connect(HLSIU(read-only); HLID (write))
extractData(HLSIU) //gets data from sources
If $Fx$
    convertData(SSIU→TSIU) // transforms them
If $Fs$
    createStructures(HLID) //builds structures
If $Fx$
    convertData(SSIU→TSIU)
    loadData(HLID) //converts and load data
```

The execution of IP is the last stage of the DI process. It is considered that a program has been executed successfully when the source data have been placed at their respective targets.

The flexibility of the MONIL environment permits at the end of the execution of an IP another one can be executed immediately, even the same one.

5. MONIL integration process

Proposed DI process:

1. Generation of integration correspondence scheme.
2. Integration programs generation.
3. Execution of integration programs.

Functionally, the MONIL framework provides a software component for each one of the three stages detailed in section 4.

5.1. Stage 1: Generation of the integration correspondence schemes.

It is the first stage of the DI process and its purpose is to build a scheme of integration correspondence to describe the DI problem we wish to solve. During this first stage, user’s participation is required, being desirable for the user to be familiarized with the integration problem characteristics.

The integration correspondence scheme sets up the bases of the DI process in describing in detail the elements that participate in the integration and all the actions that must be carried out as the part of the process.

A scheme of integration correspondence (represented by Γ_k) is the not empty set of *tuples* of attributes source-*target* called integration correspondences and represented by c_i .

A c_i establishes the existence of an integration relation between a source and a target units. A scheme of correspondences is represented by:

$$\Gamma_k = \{c_1, c_2, ...c_N\} \qquad \textbf{(16)}$$

Where the *k*-th DI problem is described, stating that it is formed by N correspondences of integration c_i .

A graphical example describing the correspondence scheme concept between three_data sources *Source1*, *Source2* and *Source3*, and a target *Target* is shown in Figure 5.1. The correspondence scheme in Figure 5.1 describes the integration between: *Matricula* of *Target* and *Matricula* of Source$_1$, *Id* of Source$_2$, and *Numero* of Source$_3$. The SIUD *Nombre* is integrated from the concatenation of *Nombres*, *ApellidoP* and *ApellidoM* from *Source$_1$*, *Nombre* from *Source$_2$*, and *DatosP* from *Source$_3$*. The TSIU *Carrera* is formed by: *Carrera, Especilidad* and *Carrera* of Source1, Source2 and Source3 respectively. The last correspondence shown by the example is, for TSIU, *edad* which is integrated from *Edad* of Source1; Source2 and Source3

Due to the GAV feature of MONIL, there are attributes of some of the sources that do not participate in the integration.

The construction of a Γ_k has 3 steps:

1. The construction of an initial scheme of correspondence.
2. Validation of the initial scheme to transform it into the final correspondence scheme.
3. The storage in the integration metamodel of the final correspondence scheme.

An initial Γ_k is formed by N correspondences c_i as follows:

$$c_I : T.\tau = S_i.\sigma_i \qquad \textbf{(17)}$$

where the sub index I is the correspondence number, T describes the TSIU in terms of its HLIU, the symbol $=$ represents the integration relationship between the target τ and the source, $\sigma_{i,j}$ represents the whole of j source units participating on behalf of S_i^{ω} and the abstraction $1-to-N$ establishes the correspondence between N sources and one target.

The component offered by MONIL for this DI process is the GenCor tool.

When an initial correspondence scheme Γ_k has been generated, the validation process begins. The validation process is semi-automatic and consists of defining and/or accepting those transformations (using MONIL conversion functions) that applied to the source SIU can solving the conflicts existing between the source and target data.

This step is considered very important because any mistake produce at this stage will have consequences on the entire integration process. Once the validation process is concluded, a new Γ_k is considered complete and valid, and it is stored in the table Correspon of the IM, ready to be used for the generation of integration programs.

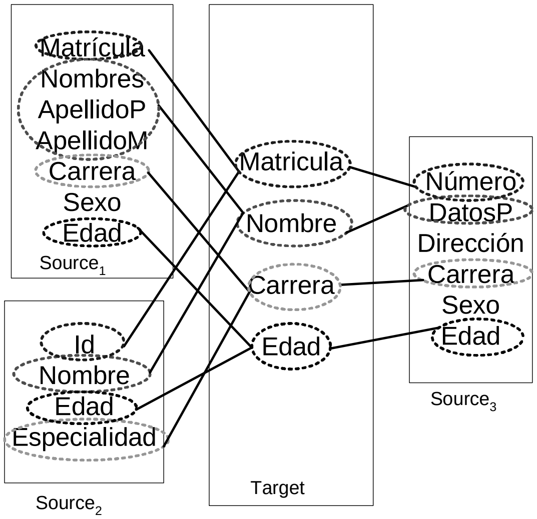


Figure 5.1. A correspondence scheme between three sources and one target.

5.2. Stage 2: Generation of the Integration Programs.

The purpose of this stage is to generate valid integration programs and the functionality is given by the modules of the GenPro and EC environments.

GenPro uses a valid correspondence scheme Γ_k stored in IM, to produce an integration program to model and solve the problem set forth by the correspondence scheme. To use GenCor, it is indispensable that correspondence schemes exist in the integration metamodel performing as the input to the automatic generation of programs.

However, MONIL environment is flexible and permits the users to build their own IP manually using the ED, as it has been already described in the previous section. The actions of IP generation are executed out of the line, that is, no connection is required at any time to any of the integration units. This is because all the information used to generate them is obtained from Γ_k stored in the metamodel, or it is typed by the user at the time of designing them.

5.3. Stage 3: Execution of the Integration Programs

This is the third and last stage of the DI process by MONIL and its purpose is to execute the stored MONIL programs to retrieve and integrate source data and deposit them as target data. The executed programs are selected by the user from all the stored programs. The execution process is automatic, that is, once it has been initiated, if successful, it does not require user's participation until it concludes. The execution is carried out by AgI.

Unlike the stages 1 and 2, the execution of an IP does require a permanent connection with the source and target HLIU: the sources require reading access and the target reading-writing access. The execution process of a stored program may be repeated the number of times the user wants. According to what has been said, the ways to execute a program are: *Append_F* and *Replace_F*.

The execution *Append_F* of a program adds the integrated information from the sources to the information existing at the target. Target information is not lost. This kind of execution is expressed as:

$$\tau_{final} = \tau_{inicial} + \rho$$

(18)

where $\tau_{inicial}$ is the value stored in the TSIU before the execution of the IP, ρ represents the integration process of the participating source SIU and the sign + represents an operation of aggregation between the existing target data and the new integrated data.

By default, the operator + transforms itself in concatenation when the participating data are of the chain type and stays as + for the numerical data. The parameters for the numerical data, however, can be modified at the beginning of the execution in average, product and mean.

On the contrary, the execution *Replace_F* of an integration program acts replacing all the existing information at the target by the source integrated information. This kind of execution is represented by:

$$\tau_{final} = \rho$$

(19)

Where ρ represents the source integrated information which will be deposited at the target. When a target TSIU is empty, *Append_F* and *Replace_F* actions produce the same result.

6. Test Cases

The purpose of this section is to check out the expressivity and scope of MONIL language by means of solving DI cases. The trial cases are presented by means of the classification of the type of structural and semantic conflict existing in the set using the concepts of semantic proximity [55]. The problem is complemented with the solution equation offered by MONIL to solve the conflict, and for the cases 1 and 5, MONIL program is included too.

Case 1. Data Scale Conflict (domain incompatibility)

Problem. To integrate at the target SIU $\tau = CAL$, the information originated at the source SIU $\sigma_{1,1} = CALIFICA$. There is a data scale conflict among the SIU semantically equivalent. The source SIU is expressed in hundreds and the target SIU should be expressed in tens (Figure 6.1).

Solution. Solution of the data scale conflicts is carried out by means of MONIL conversion functions that work with the information stored in pre-established tables. The minimum expression of the solution is:

$$\tau = \varepsilon\left(Equi[10,0],[Div]\left(\sigma_{1,1}\right)\right)$$

(20)

Where ε, is the basic function of data retrieval, *Equi* is the equivalence function (converting function) that solves the data scale conflict; and finally, $\sigma_{1,1}$ is the source SIU providing those data. The execution of the MONIL program produces an effect where the source values are converted into required units by means of:

$$valueOf(\tau) = {}^{valueOf\left(\sigma_{1,1}\right)}\!\big/_{10.0}$$

The MONIL program that solves this data integration case is shown on the Figure 6.1.

Case 2. Data Accuracy Conflict (domain incompatibility)

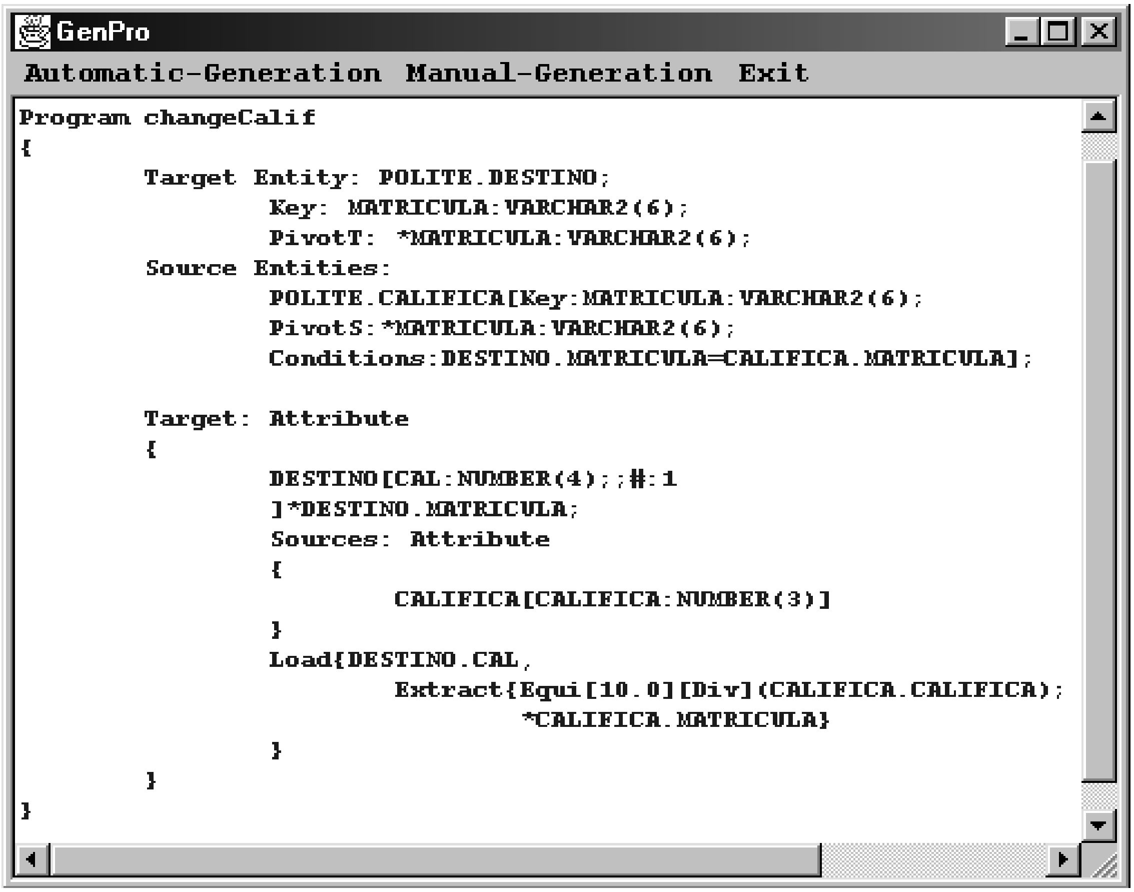
Problem. To integrate the attribute *Cal* (students' grades) from the source *CALIFICA* (which uses numbers registers) to the target *DESTINO* (that uses words or letters to represent the students' grades), it is necessary to replace the numerical values by distinctive words. The grades values from *CALIFICA* should be rated into 5 groups, and each group will be represented by a different pre-established value.

Solution. Data accuracy conflicts between the source and target is solving by the next expression:

$$\tau = \varepsilon\left(Change[X][Y](\sigma_{1,1})\right)$$

(21)

Where X represents the possible source values: SIU $\sigma_{1,1}$, and Y are the values we wish to get for the target TIU.



```
Program changeCalif
{
        Target Entity: POLITE.DESTINO;
                Key: MATRICULA:VARCHAR2(6);
                PivotT: *MATRICULA:VARCHAR2(6);
        Source Entities:
                POLITE.CALIFICA[Key:MATRICULA:VARCHAR2(6);
                PivotS:*MATRICULA:VARCHAR2(6);
                Conditions:DESTINO.MATRICULA=CALIFICA.MATRICULA];

        Target: Attribute
        {
                DESTINO[CAL:NUMBER(4);;#:1
                ]*DESTINO.MATRICULA;
                Sources: Attribute
                {
                        CALIFICA[CALIFICA:NUMBER(3)]
                }
                Load{DESTINO.CAL,
                        Extract{Equi[10.0][Div](CALIFICA.CALIFICA);
                                *CALIFICA.MATRICULA}
                }
        }
}
```

Figure 6.1. MONIL program solving a scale conflict between source and target SIUs.

Case 3. Aggregation Conflict (abstraction levels incompatibility)

Problem. It is necessary to integrate the grades of the students to calculate the averages obtained in all their subjects they have studied during an academic period.

Solution. The conflict appears when an operation of aggregation to the source SIU is necessary in order to represent a target TIU. In this case, there is a one way relation. The other way relation cannot be obtained. The participating SIU maintain a semantic relationship. A simple expression to show MONIL solution to this conflict is:

$$\tau = \varepsilon\left(Average[2](\sigma_{1,1})\right)$$

(22)

Where ε is the basic function of data extraction, $Average$ is the converting function that will transform N source data (in this case 2, indicated by the parameter [2]), into one target datum τ through the operation of the average.

Case 4. Schematic Discrepancy Problem (abstraction level = Attributes)

Problem. To integrate the data from HLSIU $S_1^1 = PARTES$ to add new columns to HLTIU $T^1 = VENTAS$.

Solution. This problem presents a schematic discrepancy conflict because the SSIU correspond to the target metadata. MONIL solves this level of schematic discrepancy by means of its construction functions that act adding new attributes to the HLTIU structure with the data provided by SSIU. It is difficult to solve this kind of conflicts because the semantic similarities among the data are established among the elements at different abstraction levels. The minimum expression of MONIL solution of this case is given by:

$$\tau = \iota(\gamma(\sigma_{1,1}), \delta(\sigma_{1,2}))$$

(23)

Where it creates new columns in the HLTIU using the function $\gamma = NewName$ that will transform the $\sigma_{1,1}$ values into new columns of the HLTIU, and the function $\delta = NewData$, will populate new columns with the values obtained from the $\sigma_{1,2}$ source.

Case 5. Schematic Discrepancy Problem (abstraction level = Entities)

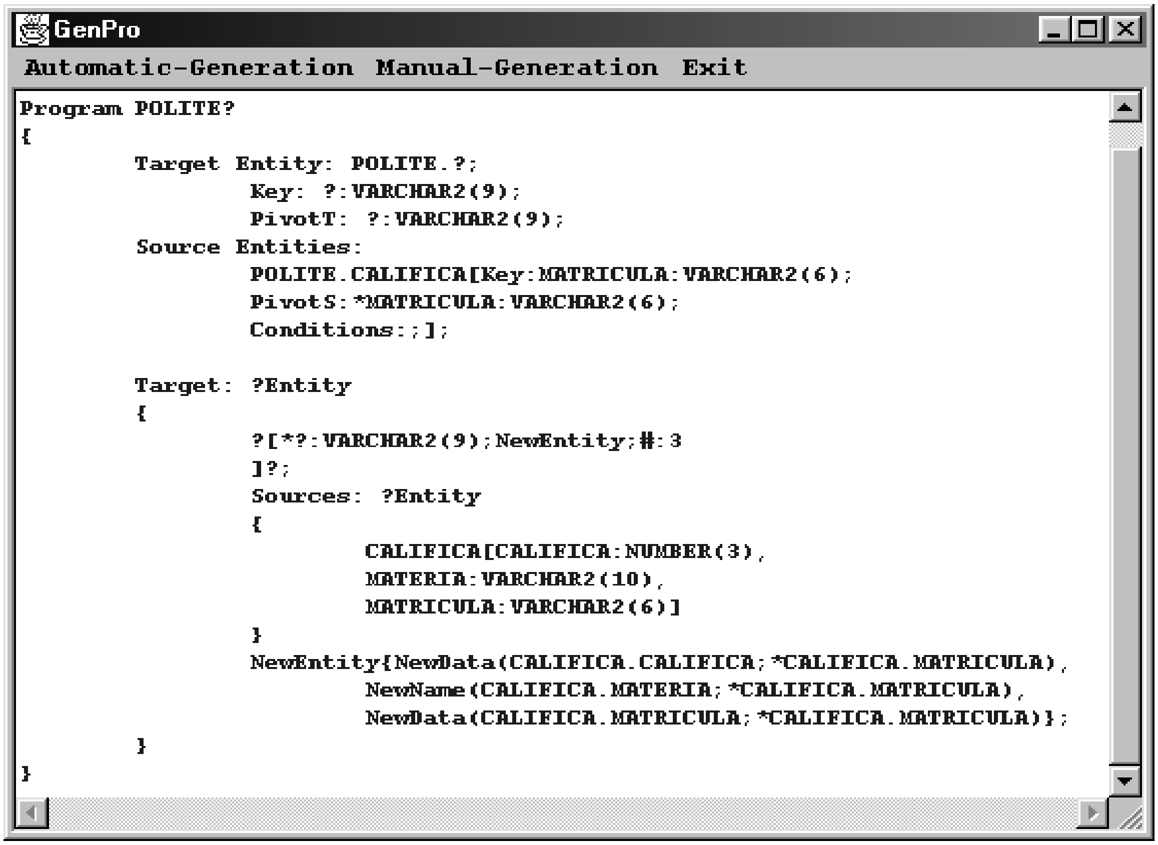
Problem. The integration should build new entities to store the grades of the students by subject. Each new entity will only store grades of the students per each subject.

Solution. MONIL solves this case by means of its construction functions which, unlike the above case, it works on a higher abstraction level and solve a schematic discrepancy conflict on entity level. The minimum expression of the solution of such a problem is:

$$\tau = \gamma(v(\sigma_{1,1}), \delta(\sigma_{1,2}))$$

(24)

Where a structure function is the one that builds new HLTIU structures using the function $\gamma = NewName$ which will transform the $\sigma_{1,1}$ values into new HLTIU, and the function $\delta = NewData$ will populate the columns of the new HLTIU created with the values obtained from the $\sigma_{1,2}$ source. See Figure 6.2.



```
Program POLITE?
{
        Target Entity: POLITE.?;
                Key: ?:VARCHAR2(9);
                PivotT: ?:VARCHAR2(9);
        Source Entities:
                POLITE.CALIFICA[Key:MATRICULA:VARCHAR2(6);
                PivotS:*MATRICULA:VARCHAR2(6);
                Conditions:;];

        Target: ?Entity
        {
                ?[*?:VARCHAR2(9);NewEntity;#:3
                ]?;
                Sources: ?Entity
                {
                        CALIFICA[CALIFICA:NUMBER(3),
                        MATERIA:VARCHAR2(10),
                        MATRICULA:VARCHAR2(6)]
                }
                NewEntity{NewData(CALIFICA.CALIFICA;*CALIFICA.MATRICULA),
                        NewName(CALIFICA.MATERIA;*CALIFICA.MATRICULA),
                        NewData(CALIFICA.MATRICULA;*CALIFICA.MATRICULA)};
        }
}
```

Figure 6.2. Solution of the schematic discrepancy problem on entity level.

7. Conclusions and Future Work

This article presents a proposal to solve the data integration problems by means of the_MONIL language. The main contributions of MONIL are:

- Definition of a formal programming language devoted to solve data integration problems.
- Introduction of concepts of integration units, pivots, conversion functions and integration conditions.
- Use of metadata to build integration metamodel as a fundamental part of the integration process.
- Automatic generation of a correspondence suggestion scheme to assist the user in the definition of the integration metamodel.
- Work environment with integrated semi-automatic tools that support the design, storage and execution of the integration programs.
- Integration process that allows to carry out integration operations in a simple and ordered way and, in many operations, without interfering with the day-to-day operations of the information systems that provide or receive data in the integration process.

Notwithstanding, and due to the complexity of the problem the data integration presents, there is much future work to perform in this area, especially in the search to automate the process integration activities that, for the time being, are still semi-automatic, among them the definition of all the integration correspondence schemes.

References

[1] J. Ullman, .Information integration using logical views. In. Proc. of the *6th International Conference on Database Theory (ICDT.97)*, vol. 1186 of Lecture Notes in Computer Science, pp. 19-40, 1997.

[2] N. Kushmerick, R. Doorenbos, and D. Weld. Wrapper induction for information extraction, 15th International Joint Conference on Artificial Intelligence, 1997.

[3] G. Wiederhold. Mediators in the architecture of future information systems, *IEEE Computer*, vol. 25, no. 3, pp. 38-42, 1992.

[4] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS approach to mediation: Data models and languages, *Journal of Intelligent Information Systems*, 1997.

[5] G. Zhou, R. Hull, and R. King. Generating data integration mediators that use materialization, *Journal of Intelligent Information Systems*, vol. 3:2/3, no. 2/3, pp. 199-221, May 1996.

[6] L. Ling, M. T. Özsu, and L. Liu. Accesing heterogeneous data through homogenization and integration mediators, Second IFCIS *Conference on Cooperative Information Systems* (CoopIS97), 1997.

[7] W. Inmon. *Building the Data Warehouse*, 2nd ed. John Wiley and Sons, 1996.

[8] M. Jarke, C. Quix, D. Calvanese, Maurizio Lenzerini, E. Francosi, S. Ligoudistiano, P. Vassiliadis, and Y. Vassiliou. Concept based design of data warehouses: The DWQ demonstrators, In Proc. of the ACM SIGMOD International *Conference on Management of Data*, p.p. 591-2000.

[9] W. Inmon, R. Terdeman, and C. Imhof. *Exploration Warehousing Turing Business Into Business Opportunity*, John Wiley and Sons, Inc., 2000.

[10] R. Kimbal, L. Reeves, M. Ross, and W. Thornthwaite. *The Data Warehouse Lifecycle Toolkit: Tools and Techniques for Designing, Developing, and Deploying Data Warehouses*, John Wiley and Sons, 1998.

[11] R. Kimball, L. Reeves, M. Ross, and W. Thornthwaite. *The Data Warehouse Lifecycle Toolkit : Expert Methods for Designing, Developing, and Deploying Data Warehouses*, John Wiley and Sons; ISBN: 0471255475, 1998.

[12] M. Larre, S. Torres, J. Torres, and E. Morales. *Un algoritmo para la integración de datos basado en el descubrimiento de relaciones*, In. Proc. of the *7° Congreso Internacional de Investigaciones en Ciencias Computacionales* (CIIC00), pp. 264-273, 2000.

[13] S. Torres, M. Larre, and J. Torres. A string representation methodology to generate syntactically valid genetic programs, WSEAS Transactions on Systems, vol. 1, p.p. 290, 2002.

[14] M. Larre, J. Torres, and E. Morales. Data integration with MONIL, metadata and correspondence suggestions, *3er. Encuentro Internacional de Ciencias de la Computación* (ENC01), vol. 2, pp. 623-632, 2001.

[15] M. Larre, J. Torres, E. Morales, and S. Torres. Data integration using the MONIL language, Proceedings of ICEIS 2002 - *the Fourth Conference on Enterprise Information Systems*, 2002.

[16] M. Larre, J. Torres, and E. Morales. MONIL, the metadata and object integration language, Advances in information science and soft computing (ISBN 960 8052 602), p.p. 114, 2002.

[17] C. Beeri, G. Elber, T. Milo, Y. Sagiv, O. Shmueli, N. Tishby, Y. Kogan, D. Konopnic-

ki, P. Mogilevski, and N. Slonim. Websuite-a tool suite for harnessing web data, In. Proc of the *International Workshop on the Web and Databases*, 1998.

[18] W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity, In. Proc. of ACM SIGMOD *Conference on Management of Data*, 1998.

[19] L. Haas, D. Kossmann, E. Wimmers, and J. Yang. Optimizing queries across diverse data sources, In Proc. of the *International Conference on Very Large Data Bases* (VLDB), 1997.

[20] Z. Ives, D. Florescu, M. Friedman, and A. Levy. An adaptative query execution system for data integration, Proc. of ACM SIGMOD *Conference on Management of Data*, 1999.

[21] S. Bergamaschi, G. Cabri, F. Guerra, L. Leonardi, M. Vincini, and F. Zambonelli. Supporting information integration with autonomous agents, *5th International Workshop CIA-2001 on Cooperative Information Agents*,, 2001.

[22] O. Duschka, M. Genesereth, and A. Levy. Recursive query plans for data integration,. *Journal of Logic Programming, special issue on Logic Based Heterogeneous Information* Systems, 1999.

[23] M. Friedman and D. Weld, .Efficient execution of information gathering plans. In. Proc. of the *International Joint Conference on Artificial Intelligence*, 1997.

[24] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources, *Intelligent Information System*, vol. 8, num. 2, pp. 117-132, 1997.

[25] C. Knoblock, S. Minton, J. Ambite, N. Ashish, P. Modi, I. Muslea, A. Philpot, and S. Tejada. Modeling web sources for information integration,. In Proc. of the *15th National Conference on Artificial Intelligence*, 1998.

[26] D. Beneventano and S. Bergamaschi. Extensional knowledge for semantic query optimization in a mediator based system, International Workshop on Foundations of Models for Information Integration (FMII-2001), 2001.

[27] D. Beneventano, S. Bergamaschi, F. Guerra, and M. Vincini. The MOMIS approach to information integration, *AAAI International Conference on Enterprise Information Systems* (ICEIS01), 2001.

[28] S. Bergamaschi, S. Castano, D. Beneventano, and M. Vincini. Semantic integration of heterogeneous information sources, Special Issue on *Intelligent Information Integration, Data and Knowledge Engineering*, vol. 36, no. 1, pp. 215-249, 2001.

[29] W. Kim, I. Choi, S. Gala, and M. Sheevel. On resolving schematic heterogeneity in multi-databases systems, *Distributed and Parallel Databases*, vol. 1, num. 3, 1993.

[30] S. Madnick. From VLDB to VMLDB (very MANY large data bases):dealing with large-scale semantic heterogeneity, in Proc. if the *21th International Conference on Very Large Databases*, pp. 11-16, 1995.

[31] F. Saltor and E. Rodriguez. On intelligent access to heterogeneous information, In. Proc. of the *4th KRDB Workshop*, 1997.

[32] M. Bright, A. Hurson, and S. Pakzad. Automated resolution of semantic heterogeneity in multi-databases, *ACM Transactions on Database Systems*, vol. 19, no. 2, pp. 212-253, 1994.

[33] R. Hull. Managing semantic heterogeneity in databases: A theorical perspective,*16th ACM SIGACT Symp. on Principles of Database Systems* (PODS.97), 1997.

[34] S. C. Diego Calvanese and, F. Guerra, D. Lembo, M. Melchiori, G. Terracina, D. Ursino, and M. Vincini. Towards a comprehensive methodological framework for semantic integration of heterogeneous data sources, Proc. of the *8th Int. Workshop on Knowledge Representation meets Databases (KRDB 2001)*, 2001.

[35] T. Häder, G. Sauter, and J. Thomas. The intrinsic problems of structural heterogeneity and an approach to their solution, *VLDB Journal*, vol. 8, no. 1, pp. 25-43, 1999.

[36] W. Klas, G. Fisher, and K. Aberer. Integrating relational and object oriented database systems using a metaclass concept, *Journal of Systems Integration*, vol. 4, no. 4, 1994.

[37] M. Roth and P. Scharz. Don.t scrap it, wrap it, *23th Conference on Very large Databases*, 1997.

[38] S. Adali, K. Candan, Y. Papakonstantinou, and V. Subrahmanian. Query caching and optimization in distributed mediator systems, In. Proc. of the ACM SIGMOD *Conference on Management Data*, 1996.

[39] C. H. Goh, S. E. Madnick, and M. Siegel. Context interchange: Overcoming the challenges of large scale interoperable database systems in a dynamic environment, *3rd. International Conference on Information and Knowledge Management* (CIKM.94), pp. 337-346, 1994.

[40] J. Hammer, M. Breunig, H. Garcia-Molina, S. Ñestorov, V. Vassalos, and R. Yerneni. Template based wrappers in the TSIMMIS system, In Proc. of the 26th SIGMOD *International Conference on Management of Data*, 1997.

[41] A. Cali, D. Calvanese, G. D. Giacomo, and M. Lenzerini. Accessing data integration systems through conceptual schemas, Proc. of the *20th International Conference on Conceptual Modeling* (ER2001), pp. 270-284, 2001.

[42] D. Calvanese, G. De-Giacomo, M. Lenzerini, D.Ñardi, and R. Rosati. A principled approach to data integration and reconciliation in data warehousing., Proc. of the *International Workshop on Design and Management of Data Warehouses* (DMDW.99), vol. 19, p. 16, 1999.

[43] A. Levy, A. Rajaraman, and O. J.J. Query an answering algorithms for information agents, *Proceeding of AAAI*, 1996.

[44] Y. Arens, C. Chee, C. Hsu, and C. Knoblock. Retrieving and integrating data for multiple information sources, *International Journal of Intelligent and Cooperative Information Systems*, vol. 2, no. 2, pp. 127-158, 1993.

[45] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. View-based query processing and constraint satisfaction, Proc. of the *15th IEEE Sym. on Logic in Computer Science* (LICS 2000), pp. 361.371, 2000.

[46] W. Litwin, L. Mark, and N. Roussopolos. Interoperatibility of multiple autonomous databases, *ACM Computing Surveys*, vol. 22, no. 3, pp. 267-293, 1990.

[47] R. Hull. Towards the study of performance trade of between materialized and virtual integrated views, *Workshop on Materialized Views*, pp. 91.102, 1996.

[48] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views,.Proceedings of the *14th ACM SIGACT SIGMOD-SIGART Symposium on principles of Database Systems*, 1995.

[49] R. Pottinger and A. Y. Levy. A scalable algorithm for answering queries using views, Proc. of the *Int. Conf. on Very Large Data Bases* (VLDB), 2000.

[50] R. Hull and G. Zhou. A framework for supporting data integration using the materialized and virtual approaches, ACM SIGMOD *International Conference on Management of Data*, pp. 481-492, 1996.

[51] R. Pottinger and A. Y. Halevy. Minicon: A scalable algorithm for answering queries using views, *VLDB Journal*, 2001.

[52] D. Theodoratos, S. Ligoudistianos, and T. Sellis. Designing the global data warehouse with SPJ views, *11th Conference on Advanced Information Systems Engineering* CAiSE.99), 1999.

[53] D. Srivastava, S. Dar, H. V. Jagadish, and A. Y. Levy. Answering SQL queries using materialized views, Proceedings of *VLDB*, 1996.

[54] A. Labrinidis and N. Roussopoulos. Reduction of materialized view staleness using on line updates, Proc. of *Workshop on Materialized Views: Techniques and Applications* (VIEW 1996), pp. 91-102, 1996.

[55] V. Kashyap and A. Sheth. Schema correspondences between objects with semantic proximity, Department of Computer Science Rutgers University, *Tech. Rep. DCS-TR-301*, 1993.

[56] J. Hopfcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation*, 2nd ed. Addison-Wesley Pub. Co., 2000.

[57] T. Kyte. *Expert One on One: Oracle*, 2nd ed. Wrox Press Inc ISBN: 1861004826, 2001.

[58] G. Harrison. *Oracle SQL High-Performance Tuning*, 2nd ed. Prentice Hall PTR ISBN: 0130123811, 2000.