Using a Contextual Logic Programming Language to Acess Data in Warehousing Systems

Valéria Pequeno, Salvador Abreu, and João Carlos Moura Pires

CENTRIA, Departamento de Informática,
Faculdade de Ciências e Tecnologia, FCT, Universidade Nova de Lisboa
2829-516, Caparica, Portugal
http://www.fct.unl.pt

Abstract. Data Warehouses (DWs) are repositories containing the unified history of an enterprise used by decision makers for performance measurement and decision support. The data must be Extracted from heterogeneous information sources, Transformed and integrated to be Loaded (ETL) into the DW, using ETL tools, which are mainly procedural. This means that the knowledge of the procedures and adopted policies are hidden in several programs (written in different languages and paradigms). We propose a development of a framework to declaratively model the ETL process, including the semantic correspondence between schema's components, to provide a better understanding of the semantic associated with the ETL process. A prototype, based on contextual logic programming with persistence, is presented.

1 Introduction

A Data Warehouse (DW) is an integrated data repository that represents the unified history of an enterprise at a suitable level of detail to be useful for analysis [1]. The data must be Extracted from different information sources, Transformed and integrated to be Loaded (ETL) into the DW. DW data is then delivered to Data Marts (DM), probably with some more changes. DMs are subsets of DW data designed to serve specific demands of a particular group of users. Moreover, a DW or a DM should be created and accessed through metadata that provides detailed documentation for data in the DW system, such as applied transformations and origin of data.

The design and population of DW through ETL processes is a difficult and time-consuming task involving considerable cost in human and financial resources. Data models complexity expands both in sources and in DW, which gives rise to the difficulty of managing and understanding these models [2, 3]; and data volumes growing at a significant pace. Although there are specialised tools with graphical interface to do the mapping between the source information and the DW system, they are mostly procedural. This means that the knowledge of procedures and policies are hidden in diverse codes, which are totally dependent on experts and technicians. These tools focus strongly on data movement, as the models are only used as a means to this aim.

An ETL process, for building a DW system, is not concerned just with mapping between schemata, but also with an effective data integration that expresses a unified view of the enterprise. In this context, it is crucial to have a conceptual reference model [4–6]. A Reference Model (RM) is an abstract framework that provides a common semantic that can be used to guide the development of other models and help with data consistency [5].

It is proposed in [7] to take a declarative approach, which is based on making clear the relationship between data sources and DW using correspondence assertions and taking into account the Reference Model, independently of the ETL process involved. Furthermore, the ETL process itself can use this information.

A proof-of-concept prototype, which is the basis of this paper, has been implemented using a standard programming language Prolog, as well as a logic programming framework called Information Systems COnstruction language (ISCO). ISCO is based on a Contextual and Constraint Logic Programing [8]. It allows the construction of information systems, which can transparently access data from various heterogeneous sources in a uniform way, like a mediator system [9]. This paper describes as ISCO can be used to access data in a reference model-based warehousing environment.

The remainder of this paper is structured as follows. Section 2 shows the workings of conceptual design of the ETL processes as well as the motivation to this research. Section 3 describes our reference model-based data warehouse architecture. Contextual Logic Programming and ISCO tool are briefly approached in Section 4. Section 5 deals with the issue of modelling and representation in the data warehouse and ISCO framework. Section 6 illustrates some details of implementation. The paper ends with Section 7, which points out the new features of the approach presented here and planned future work on this topic.

2 Research motivation

Nowadays, there is a plethora of commercial ETL tools in the market place, but very few of them are from academic work. Most of the tools suggest reduced support at the conceptual level.

In the academic area, ETL research for DW environment is focused mainly on the process modelling concepts, data extraction and transformation, and cleaning frameworks [10–12]. So far, the authors of this paper are not aware of any research that precisely deals with both mappings (structural and instance) between the sources and the DW, and with the problem of semantic heterogeneity in a whole conceptual level. There are few prototypes, which usually are implemented to perform technical demonstration and validation of the developed research work [13, 11]. As example of works that developed some implementation, we quote [4, 14, 15]. Reference in [4] presents a methodology that was applied in the TELECOM ITALIA framework. Similar to our work, their proposal include a reference model (cited as "enterprise model") designed using an Enriched Entity-Relationship (EER) model. Their prototype focused on logical schemata and on data movement, any transformation (e.g. restructuring of schema and

values) or mapping of instances were deferred for the logical level. In our approach everything stays at the conceptual level. References [14, 15] use ontologies as a common data model to deal with the data integration problem. Skoutas and Simitsis in [14] use a graph-based representation to define the schemata (source and DW) and an ontology, described in OWL-DL. Based on this ontology and the annotated graphs, automated reasoning techniques are used to infer correspondences and conflicts between schemata. Salguero et. al. in [15] extended OWL with temporal and spatial elements, and used the annotation properties of OWL to store metadata about the temporal features of information sources. References [14, 15] mentioned nothing about the mapping of instances, neither they use a reference model in their architecture.

In a data integration scenario, other than DW, there are several works available, mainly to establish the structural mapping between the sources and the global schema (see [16–18] for a survey.); and some works involving the problem to map instances that represent the same entity in the real-world, the instance matching problem (see [18, 19] for a survey.). Approaches for structural matching of schemata focus on schema matching, i.e., on (semi-) automatically identifying semantic correspondences between schema components. In order to do this, the proposed techniques ([20–22]) exploit several kinds of information, including schema characteristics, background knowledge from dictionaries and thesauri, and characteristics of data instances. Approaches for dealing with the instance matching problem cover several kinds of data, such as object, tuples, web data, etc., and many different strategies, including look-up tables, heuristics, etc.. These topics were not considered in our research, as we focused on making explicit the relationship between schemas (in terms of both structure and instance).

In the ETL market, some approaches focus on code generation from specifications of mapping and data movement, which are designed by Information Technology (IT) specialists using graphical interfaces [23]. It is the case, e.g., of Pentaho Kettle, an open-source ETL tool, which has an easy-to-use graphical interface and a rich transformation library, but the designer only works with pieces of structures. Others ETL approaches focus on representation of ETL processes [23, 24]. Orchid [24], for instance, is a system part of IBM Information Server that facilitates the conversion from schema mappings to ETL processes and vice-versa. Some Database Management Systems (DBMS) vendors have embedded ETL capabilities in their products, using the database as "engine" and Structured Query Language (SQL) as supporting language.

Also in market ETL tools can be found that do not depend on any particular database technology, allowing easy integration with Business Intelligence (BI) projects deployment (e.g. Oracle Data Integration). Further, there are ETL tools metadata-driven, which are becoming the current trend with ETL data processing. This approach addresses complexity, meets performance needs, and also enables re-use. Informatica PowerCenter was the pioneer. Many of these tools have integrated metadata repositories that can synchronise metadata from

(source) systems, databases and other BI tools. The metadata is represented by proprietary scripting languages, which run within a centralised ETL server [25].

Essentially, the ETL tools are procedural. This means that the knowledge of the procedures and adopted policies are hidden in several programs (written in different languages and paradigms), as well as it is strongly dependent on experts and technicians. An other feature in the ETL process is that the data and business rules evolve requiring the ETL code to be modified and maintained properly. An additional difficulty occurs when ETL tools are used in a data integration context, since each one manages metadata differently. Furthermore, there is not a standard to draw models or to describe data.

This work intends to address the problems stated above. The research focuses on a declarative approach, since a logic-based formalism allow us to deal with the complexity of managing data warehouses and the associated ETL processes in a concise and very perceptible way. Moreover, the semantic integration between the different data sources is accomplished using correspondence assertions to relate concepts from various sources.

3 Data Warehouse Architecture

Our proposal for DW organisation, presented in [7], offers a way to express the existing data models (source, DW, DM, RM) and the relationship between them. The approach is based on *Schema language* (L_S) and *Perspective schema language* (L_{PS}).

Schema language (L_S) is used to describe the actual data models (source, DW, DM, RM). The formal framework focuses on an object-relational paradigm, which includes definitions adopted by the main concepts of object and relational models as they are widely accepted in literature – cf. [26, 27].

Perspective schema language (L_{PS}) is used to describe *perspective schemata*. A perspective schema is a special kind of schema that describes a data model (part or whole) (*target schema*) in terms of other data models (*base schemata*). In Fig. 1, $\mathbf{P}_{S|DW}$ is a perspective schema whose base schema is the source schema **S** and the target schema is the data warehouse schema **DW**.

A simple sales scenario is used as a running example through out this paper. The example consists of two schemata: **S** and $\mathbf{P}_{S|DW}$, shown in Fig. 1. **S** and $\mathbf{P}_{S|DW}$ include information about sales of products, being that $\mathbf{P}_{S|DW}$ contains summarised information regarding sales.

L_{PS} mainly extends L_S with two components: Correspondence Assertions (CAs) and Matching Functions (MFs). Correspondence Assertions formally specify the relationship between schema components in a declarative fashion. CAs are classified in four groups: Property Correspondence Assertion (PCA), Extension Correspondence Assertion (ECA), Summation Correspondence Assertion (SCA), and Aggregation Correspondence Assertion (ACA). Property CAs relate properties of a target schema to the properties of base schemata. The Extension CAs are used to describe which objects/tuples of a base schema should have a corresponding semantically equivalent object/tuple in the target schema. The

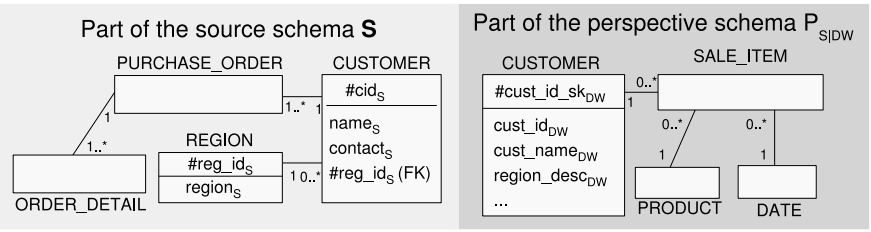


Fig. 1. Simple sales example - partial representation.

Summation CAs are used to describe the summary of a class/relation whose instances are related to the instances of another class/relation by breaking them into logical groups that belong together. They are used to indicate that the relationship between the classes/relations involve some type of aggregate functions or a normalisation process. The Aggregation CAs link properties of the target schema to the properties of the base schema when a SCA is used. Examples of CAs are shown in Fig. 2.

Property Correspondence Assertions (PCAs)
ψ_1: $\mathbf{P_{S\|DW}}$ [CUSTOMER] • **cust_id**$_{\mathrm{DW}}$ → **S** [CUSTOMER] • **cid**$_{\mathrm{S}}$
ψ_2: $\mathbf{P_{S\|DW}}$ [CUSTOMER] • **cust_name**$_{\mathrm{DW}}$ → **S** [CUSTOMER] • **name**$_{\mathrm{S}}$
ψ_3: $\mathbf{P_{S\|DW}}$ [CUSTOMER] • **region_desc**$_{\mathrm{DW}}$ → **S** [CUSTOMER] • FK_2 • region$_{\mathrm{S}}$
Extension Correspondence Assertion (ECA)
ψ_4: $\mathbf{P_{S\|DW}}$ [CUSTOMER] → **S** [CUSTOMER]

Fig. 2. Examples of correspondence assertion.

In Fig 2, the CA ψ_4 defines that the relation CUSTOMER of perspective schema $\mathbf{P_{S|DW}}$ is semantically equivalent to relation CUSTOMER of schema **S**. It means that for each instance **o** in **S**.CUSTOMER, there is a correspondent instance **o′** in $\mathbf{P_{S|DW}}$.CUSTOMER such that **o** and **o′** represent the same entity in the real world. The CAs ψ_1, ψ_2, and ψ_3 define the relationship between the properties of relations $\mathbf{P_{S|DW}}$.CUSTOMER and **S**.CUSTOMER. The property **region_desc**$_{DW}$ is not directly related to a property of $\mathbf{P_{S|DW}}$.CUSTOMER, but to the property **region**$_S$ of relation $\mathbf{P_{S|DW}}$.REGION through the path expression FK_2 • region$_{\mathrm{S}}$ (FK_2 is the name of the foreign key in $\mathbf{P_{S|DW}}$.CUSTOMER that refers to $\mathbf{P_{S|DW}}$.REGION).

Matching functions indicate when two data entities represent the same instance of the real world. These functions, as occur in [28], define a 1:1 correspondence between the objects/tuples in families of corresponding classes/relations. In particular, the work in [7] based on matching function signatures, being that their implementation shall be externally provided, since their code is very close to the application domain.

Fig. 3 illustrates the basic components of the proposed architecture and their relationships. The schemata **RM**, **DW**, **DM**, $\mathbf{S}_1$,...,$\mathbf{S}_n$ are defined using the lan-

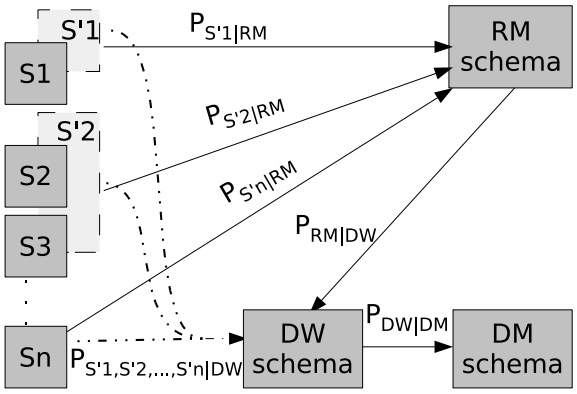


Fig. 3. Proposed architecture.

guage L_S and represent, respectively, the reference model, the data warehouse, a data mart, the source schemata $\mathbf{S}_1$,...,$\mathbf{S}_n$. The schemata $\mathbf{S'}_1$ and $\mathbf{S'}_2$ are defined using the language L_{PS}. They are special kinds of perspective schemata (called *view schema*), since the target schema is described in the scope of a perspective schema, instead of just referring an existing schema. $\mathbf{S'}_1$ and $\mathbf{S'}_2$ represent, respectively, the view schemata $\mathbf{S'}_1$ (a viewpoint of schema $\mathbf{S}_1$), and $\mathbf{S'}_2$ (an integrated viewpoint of schemata $\mathbf{S}_2$ and $\mathbf{S}_3$). The relationships between the target schema and the base schemata are shown through the perspective schemata $\mathbf{P}_{s'1|RM}$,..., $\mathbf{P}_{s'n|RM}$,$\mathbf{P}_{RM|DW}$, and $\mathbf{P}_{s'1,s'2,...,s'n|DW}$ (denoted by arrows).

In [7] is proposed an inference mechanism that, given a set of both schemata and perspective schemata as base, and a perspective schema as target, can deduce a new perspective schema. In context of the Fig. 3, the perspective schema $\mathbf{P}_{s'1,s'2,...,s'n|DW}$ can be automatically deduced by the inference mechanism, having as base the schemata $\mathbf{S}_1$,...,$\mathbf{S}_n$ and the perspective schemata $\mathbf{P}_{s'1|RM}$,..., $\mathbf{P}_{s'n|RM}$, and as target the perspective schema $\mathbf{P}_{RM|DW}$. For a more detailed and formal description of L_S and L_{PS} languages, the reader is referred to [29, 30, 7].

4 Contextual Logic Programming and ISCO

Logic Programming languages are akin to relational databases but provide a significantly higher expressive power, due to their two fundamental mechanisms of nondeterminism and unification, both of which form the basis of the Prolog language. However, it can be argued that standard Prolog is lacking in several areas, which include program structuring facilities and data persistence management. The ISCO programming system addresses both of these issues.

Contexts: the purpose of Contextual Logic Programming (CxLP) was initially to deal with Prolog's traditionally flat predicate namespace, which seriously hindered its usability in larger scale projects. A more recent proposal [31] rehabilitates the ideas of CxLP by viewing contexts not only as shorthands for

a modular theory but also as the means of providing dynamic attributes which affect that theory: we are referring to unit arguments, as described in Abreu and Diaz's work [32]. It is particularly relevant for our purposes to stress the *context-as-an-implicit-computation* aspect of CxLP, which views a context as a first-class Prolog entity – a term, behaving similarly to an object in what it carry state (the unit argument terms) and respond to messages (goals evaluated in context).

Persistence: having persistence in a Logic Programming language is a required feature if one is to use it to construct actual information systems; this could conceivably be provided by Prolog's internal database but is best accounted for by software designed to handle large quantities of factual information efficiently, for instance relational database management systems. The semantic proximity between relational database query languages and logic programming makes the former a privileged candidate to provide Prolog with persistence.

ISCO [8] is a proposal for Prolog persistence which includes support for multiple heterogeneous databases and which extends access to technologies other than relational databases, such as LDAP directory services or, more significantly, the semantic web in the form of SPARQL queries over OWL ontologies [33]. ISCO has been successfully used in a variety of real-world situations, ranging from the development of a university information system to text retrieval or business intelligence analysis tools [8].

ISCO's approach for interfacing to DBMSs involves providing Prolog declarations for the database relations, which are equivalent to defining a corresponding predicate, which is then used as if it were originally defined as a set of Prolog facts. While this approach is convenient, its main weakness resides in its present inability to relate distinct database goals, effectively performing joins at the Prolog level. While this may be perceived as a performance-impairing feature, in practice it's not the show-stopper it would seem to be because the instantiations made by the early database goals turn out as restrictions on subsequent goals, thereby avoiding the filter-over-cartesian-product syndrome.

5 Prototype architecture

The present proposal has implemented a proof-of-concept prototype using a Prolog language. The prototype comprises six cooperating modules, namely the *schema manager*, the *inference mechanism*, the *schema repository*, the *ISCO translator*, the *ISCO-generated applications*, and the *ISCO repository*. The architecture of the prototype is depicted in Fig. 4.

The *schema manager* module was written using native-prolog. It is used by the designer to manage the schemata (in language L_S) as well as the perspective schemata (in language L_{PS}). The designer, using the *schema manager*, should define all source schemata, the data warehouse schema, the reference model schema, and perspective schemata that he/she needs.

The *inference mechanism* has been written using native-prolog. It is a rule-based rewriting system that automatically generates new perspective schemata

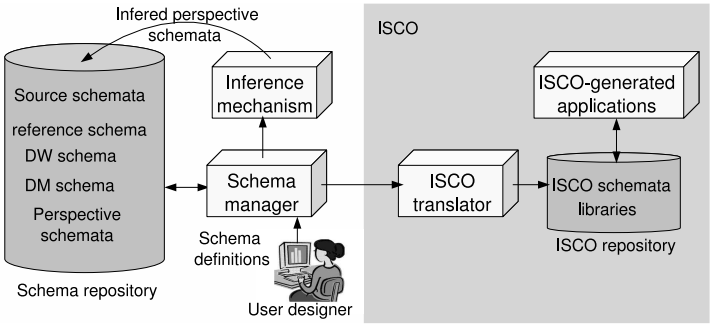


Fig. 4. Prototype architecture.

based on previous ones. It is formed by rules for rewriting CAs, rules for rewriting match function signatures, and rules for rewriting components that are presents in CAs or in matching function signatures, being a total of 39 rules.

The *ISCO translator* performs the mapping between schemata written in L_S or L_{PS} languages to ISCO schemata. Classes or relations of schemata written in language L_S are directly mapped classes in ISCO, which are also called " classes" and which compile to regular Prolog access predicates. Classes or relations of perspective schemata (written in language L_{PS}) are also mapped to ISCO classes based on CAs and match function signatures. In the current implementation, when perspective schemata are translated to ISCO schemata, it is assumed that their base schemata were already mapped to ISCO. *ISCO translator* has been written in native prolog. All functions, inclusive of the match functions, declared in the original perspective schemata are defined in a library called *v_utils*, which is common to all ISCO schemata created (more details in the next section).

The *ISCO-generated applications* includes all files that are necessary to access data from information sources. So, data in any perspective schema mapped to ISCO, specifically any inferred perspective schema between the DW and its sources, can be queried in a transparent way, just as in a mediator approach (more details in the next section).

The *schema repository* stores both the schemata (in language L_S) and perspective schemata (in language L_{PS}), including any inferred perspective schema created by the inference mechanism, while the *ISCO repository* is used to store ISCO schemata and ISCO files (libraries, units, etc.).

The next Section present implementation details using the running example to describes the process of ISCO-generation applications in warehousing environments using the implemented prototype.

6 Implementation issues

Each class or relation in the (perspective) schemata are mapped to ISCO classes using the *ISCO translator*. The ISCO classes may reflect inheritance, keys, indexes, foreign keys and sequences to name a few. The process involved differ

enough, depending on whether the original schema was written in L_S language or is a perspective schema.

In the case of schemata defined in L_S language, a declaration is added to ISCO schema in order to provide ISCO with the necessary information to access an external data source, such as an ODBC-accessed database. In the context of the running example (see Fig. 1), the ISCO schema that is mapped from schema **S** shall contain the clausule:

external(**S**, postgres(S)).

This clausule means that **S** is an outside database hosted in PostgreSQL.

All classes or relations in the original schema are simply mapped to ISCO classes, which should be declared as *external* and *mutable*. External means that the class has been created in an independent database and mutable means that its instances can change. For instance, the relation **S**.CUSTOMER is mapped to ISCO as illustrated in Fig 5.

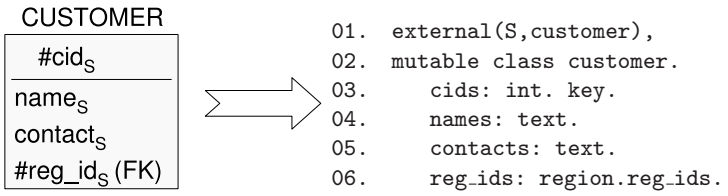
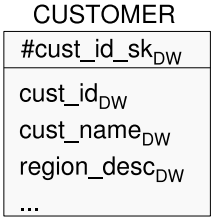


Fig. 5. Example of the mapping of a relation (in a schema) to a ISCO class.

In Fig. 5, lines 1 and 2 mean that the instances of ISCO class CUSTOMER are in database **S**, class CUSTOMER, line 3 means that the property $\mathbf{cid}_S$ is an integer number and a primary key, and line 6 means that the property $\mathbf{reg\_id}_S$ is a foreign key that refers to class REGION through property $\mathbf{reg\_id}_S$. Note that, in this example, the ISCO class and the class in the database have the same name, but could be different ones. The ISCO class CUSTOMER defines the predicate **customer/4**, which behaves as a database predicate but relies on an external system (in this case the PostgreSQL ORDBMS) to provide the actual facts. In the current implementation, it is assumed that all instances of ISCO classes have an object identity (OID), which is a integer number automatically generated by the system.

In the case of perspective schemata, the classes or relations are usually mapped to *computed* classes. It means that the class instances will be generated each time that a query is made to the class, similar to the concept of view in SQL. Computed classes are expected to contain one or more rules. These rules define how the computed class instances are obtained and always come after the computed class definition. In the body of each rule, the variables, which represent the computed class arguments, must have the same name of the respective

argument that they represent and they must be all in uppercase. This is necessary in order for Prolog to link each variable with its respective computed class argument correctly. Classes or relations in the perspective schema are mapped to computed classes in ISCO when they are related to classes or relations in a base schema through some extension correspondence assertion (ECA). A example is illustrated in Fig. 6.




```
01.  computed class customer.
02.     cust_id_skdw: int. key.
03.     cust_iddw: int.
04.     cust_namedw: text.
05.     region_descdw: text.

06.  rule:- s:> customer@(oid=Osource,cids=CUST_IDDW,
        names=CUST_NAMEDW,reg_ids=Var1),
07.     s:> region@(reg_ids=Var1,regions=Var2),
08.     REGION_DESCDW=Var2,
09.     CUST_ID_SKDW=⟨⟨createSKey⟩⟩,
10.     OID=⟨⟨createNewOID⟩⟩.
```

Fig. 6. Example of the mapping of a relation (in a perspective schema) to a ISCO class.

In Fig 6, line 6 defines a query to the ISCO class CUSTOMER in source schema **S**. This query is obtained using the CAs $\psi_1 - \psi_4$ (see Fig. 2), being that the values of **cust_iddw** and **cust_namedw** are acquired directly from this query while the value of **region_descdw** requires an additional query to ISCO class REGION in schema **S** (lines 7 and 8). Lines 9 and 10 define the necessary steps to generate, respectively, surrogate keys and oids for the computed class. The surrogate key is an integer number automatically generated based on the object identity in variable "Osource". The oid for the computed class (OID) is a compound object identity with the following structure:

$$\text{oid(schema,class,oids)},$$

being that *class* is the name of the computed class which OID belongs, *schema* is the name of the schema which *class* belongs, and *oids* is a list of compound object identities in the form of (S′,C′,**o′**), with C′ being the name of a class or relation in a schema S′ containing the object identity **o'** from what the OID is derived. The object identity **o′**, in turn, can be a compound object identity or a simple object identity (an integer number). For example, OIDs for computed classes may look like:

$$\text{oid}(\mathbf{P}_{S|DW},\text{CUSTOMER},\text{oid}(\mathbf{S},\text{CUSTOMER},667789))$$

which means the object in the computed class CUSTOMER is derived from object in **S**.CUSTOMER whose OID is 667789.

Once having the ISCO schemata, the following phase is to generate a GNU Prolog/CX executable containing the native-code executable version of all ISCO predicates. GNU Prolog/CX compiles Prolog (and ISCO) programs to native executables. Each schema and perspective schema described in ISCO, as well the library *v_utils*, will correspond to units whose terms can be instantiated and collected into a list to form a context. A set of operations and operators are available in GNU Prolog/CX to construct contexts, being the o more usual in our application the *context extension* operation given by the operator `:>`. The goal **U** `:>` **G** extends the current context with the unit **U** and resolves **G** in the new context, as if it were regular Prolog. For instance, to make a interrogation to the computed class CUSTOMER, we can use the following syntax:

$$\texttt{v\_utils :>}\ P_{S|DW}\ \texttt{:> customer(A,B,C,D).}$$

In this goal, we start by extending the initially empty context with unit v_utils. After, this new context is again extended with the unit $P_{S|DW}$, and it in the latter context that goal `customer(A,B,C,D)` is derived.

7 Conclusions and Future Work

In this article we have discussed an implementation that permits the generation of applications which transparently access source information in a reference model-based warehousing system based on a logic-based formalism. We had access to a system which already provides a Logic-based programming layer while being able to transparently access facts stored in existing RDBMs. Having a logic programming appearance lay down a setting where programs are first-class objects on which meta-reasoning may be performed, including proofs of correctness or explanations for results.

The prototype has been developed using a contextual logic programming with persistence, called ISCO [8]. ISCO provides: a) high levels of performance, by virtue of being derived from GNU-Prolog; b) expressiveness, due to the use of constraint logic programming; c) simplicity; d) persistence; and e) structured code, as a result of using Contextual Logic Programming constructs. Constraints (over finite domains) are used to generate more efficient SQL codes for database predicates, besides being a form of search-space pruning which is complementary to backtracking in that propagation works as an a-priori filter on the search.

ISCO allows access to heterogeneous data sources and to perform arbitrary computations. User-queries can be done in ISCO, in a transparent way to access the information sources, even the data of the DW schema. This feature can be useful in some situations, although a mediator strategy in a DW context, in general, may not be appropriate, due to particular nature of the DW. Specifically, this work can be used in applications that hide from their users the complexity involved in accessing multiple data sources, since these sources are in databases that can be queried via remote access in real-time.

Some improvements of our prototype will be done, namely: to use foreign keys in ISCO classes with aggregate functions, or derived by a normalisation process; improvement the generation of surrogate keys. The final prototype will be applied to various situations, some synthetic and others real, as a means of providing experimental validation of the usefulness of the chosen approaches.

For future work, we are presently working on how the perspective schemata can be used to automate the materialisation of the ETL process. Various other on-going research is related to this work, such as: a) management of schema versioning using contexts in ISCO; and b) incorporating aspects from temporal contextual logic programming [34] to deal with evolving data and schemata, and so to further reduce the complexity of the system, by allowing the temporal component to be abstracted. Another important direction for future work is to develop a graphical user-friendly interface to declare the schemata in our language, and thus, hide some syntax details.

References

1. Inmon, W.H.: Building the data warehouse. 4th edn. Wiley Publishing (2005)
2. Pérez, J.M., Berlanga, R., Aramburu, M.J., Pedersen, T.B.: A relevance-extended multi-dimensional model for a data warehouse contextualized with documents. In: DOLAP'05: Proc. of the 8th ACM Intl. Workshop on Data Warehousing and OLAP, USA, ACM (2005) 19–28
3. Matias, R., Moura-Pires, J.: Revisiting the olap interaction to cope with spatial data and spatial data analysis. In: ICEIS'07 - Proc. of the 9th Intl. Conf. on Enterprise Information Systems. Volume DISI. (2007) 157–163
4. Calvanese, D., Dragone, L., Nardi, D., Rosati, R., Trisolini, S.M.: Enterprise modeling and data warehousing in TELECOM ITALIA. Inf. Syst. **31**(1) (2006) 1–32
5. Imhoff, C., Galemmo, N., Geiger, J.G.: Mastering Data Warehouse Design - Relational and Dimensional Techniques. Wiley Publishing (2003)
6. Knackstedt, R., Klose, K.: Configurative reference model-based development of data warehouse systems. Idea group publishing (2005) 32–39
7. Pequeno, V.M., Pires, J.C.G.M.: Using perspective schemata to model the ETL process. In: Intl. Conf. on Management Information Systems, France (June 2009)
8. Abreu, S., Nogueira, V.: Using a logic programming language with persistence and contexts. In: INAP'05: 16th Intl. Conf. on applications of declarative programming and knowledge management. Volume 4369 of Lecture Notes in Computer Science., Springer (2006) 38–47 (Revised Selected Papers).
9. Wiederhold, G.: Mediators in the architecture of future information systems. In: IEEE Computer. Volume 25(3). (1992) 38–49
10. Lujn-mora, S., Vassiliadis, P., Trujillo, J.: Data mapping diagrams for data warehouse design with UML. In: ER'04: 23rd Intl. Conf. on Conceptual Modeling, Springer (2004) 191–204
11. Raman, V., Hellerstein, J.: Potter's wheel: An interactive data cleaning system. In: 27th VLDB Conference, Italy (2001)
12. Albrecht, A., Naumann, F.: Managing etl processes. In: NTII'08: Intl. Workshop on New Trends in Information Integration, New Zealand (2008) 12–15
13. Vassiliadis, P., Vagena, Z., Skiadopoulos, S., Karayannidis, N., Sellis, T.K.: ARKTOS: towards the modeling, design, control and execution. Information Systems **26(8)** (2001) 537–561

14. Skoutas, D., Simitsis, A.: Ontology-based conceptual design of ETL processes for both structured and semi-structured data. Int. J. Semantic Web Inf. Syst. **3**(4) (2007) 1–24
15. Salguero, A., Araque, F., Delgado, C.: Ontology based framework for data integration. WSEAS Trans. Info. Sci. and App. **5**(6) (2008) 953–962
16. Hai, Do, H.: Schema Matching and Mapping-based Data Integration: Architecture, Approaches and Evaluation. VDM Verlag, Germany (2007)
17. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. The VLDB Journal **10**(4) (2001) 334–350
18. Doan, A., Halevy, A.Y.: Semantic integration research in the database community: A brief survey. AI Magazine **26**(1) (2005) 83–94
19. Bleiholder, J., Naumann, F.: Data fusion. ACM Comput. Surv. **41**(1) (2008) 1–41
20. Nottelmann, H., Straccia, U.: Information retrieval and machine learning for probabilistic schema matching. Inf. Process. Manage. **43**(3) (2007) 552–576
21. Chiticariu, L., Hernández, M.A., Kolaitis, P.G., Popa, L.: Semi-automatic schema integration in clio. In: VLDB '07: Proceedings of the 33rd international conference on Very large data bases, VLDB Endowment (2007) 1326–1329
22. Engmann, D., Maßmann, S.: Instance matching with COMA++. In: BTW Workshops. (2007) 28–37
23. Kimball, R., Caserta, J.: The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data. John Wiley & Sons (2004)
24. Dessloch, S., Hernandez, M., Wisnesky, R., Radwan, A., Zhou, J.: Orchid: Integrating schema mapping and ETL. ICDE'08: IEEE 24th International Conference on Data Engineering (April 2008) 1307–1316
25. Raminhos, R.F.: Extraction and transformation of data from semi-structured text files using a declarative approach. Master's thesis, Faculdade de Ciências e Tecnologia of the Universidade Nova de Lisboa (2007)
26. Codd, E.F.: A relational model of data for large shared data banks. In: Communications of the ACM. (1970) 377–387
27. Cattell, R.G., Barry, D., eds.: The Object Database Standard ODMG 3.0. Morgan Kaufmann Publishers (2000)
28. Zhou, G., Hull, R., King, R.: Generating data integration mediators that use materialization. J. Intell. Inf. Syst. **6(2/3)** (May 1996) 199–221
29. Pequeno, V.M., Pires, J.C.G.M.: A formal object-relational data warehouse model. Technical report, Universidade Nova de Lisboa (November 2007)
30. Pequeno, V.M., Pires, J.C.G.M.: Using perspective schemata to model the ETL process. Technical report, Universidade Nova de Lisboa (2009)
31. Abreu, S., Diaz, D.: Objective: in Minimum Context. In Palamidessi, C., ed.: ICLP'03: 19th Intl. Conf. in Logic Programming. Volume 2916 of Lecture Notes in Computer Science., India, Springer-Verlag (December 2003) 128–147
32. Abreu, S., Diaz, D.: Objective: in minimum context. In Palamidessi, C., ed.: In Proc. of 19th Intl. Conf. in Logic Programming (ICLP 2003), Lecture Notes in Computer Science 2916, Springer-Verlag (2003) 128–147
33. Lopes, N., Fernandes, C., Abreu, S.: Representing and Querying Multiple Ontologies with Contextual Logic Programming. ComSIS: Computer Science and Information Systems **05**(02) (December 2008)
34. Nogueira, V., Abreu, S.: Temporal contextual logic programming. Electron. Notes Theor. Comput. Sci. **177** (2007) 219–233