ISSN 0121–1129

LATIN Interpreter, a Language for Systems Implementation

LATIN Interprete, un leguaje para la implantación de sistemas

Fecha de recepción: 8 de julio de 2010
Fecha de aprobación: 20 de octubre de 2010

Emiro Muñoz Jerez*
Jaime Octavio Albarracín Ferreira**

Abstract

It would be wrong to see reality simply as constituted by isolated objects. If one observes with care, some of such objects are mutually affected, and make up interrelations, which constitute systems. Though systems are the way reality works, and objects are the systems basic constituents. But how can be systems described?

At present do not exist languages to describe systems, only structures such as (3GL, DDL or SQL), or objects (ODL or OQL). Therefore it has been designed and called LATIN, (Language Toward Integration) the language adapted to describe the above mentioned systems. Here there are specific details of the interpreter LATIN's implementation.

Key words: Syrdam, Databases, Compilers and Interpreters, Objects, Systems, Grammars.

Resumen

Sería equivocado ver la realidad como simplemente constituida por objetos aislados. Si se observa con detenimiento, ciertas clases de tales objetos, al afectarse mutuamente, dan lugar a interrelaciones, que conforman los sistemas. De ahí que los sistemas son la forma en que opera la realidad, y los objetos son los constituyentes básicos de los sistemas. Pero ¿cómo describir los sistemas? En el presente no existen lenguajes para describir sistemas, sino solo para estructuras (3GL, DDL o SQL) u objetos (ODL or OQL); por consiguiente, el lenguaje adaptado para describir los sistemas ha sido designado y llamado LATIN (Lenguaje hacia la integración). En este artículo se especifican los detalles de la implantación del LATIN Interpreter.

Palabras clave: Syrdam, Bases de datos, Compiladores e intérpretes, Objetos, Sistemas, Gramáticas.

* Systems Engineer. Specialist in University Teaching. Universidad Industrial de Santander, Bucaramanga, Santander, Colombia. emiro270273@gmail.com
** Systems Engineer. Ph.D. in Computer Science, University of Oviedo, Spain. Bucaramanga, Santander, Colombia. jaimealb@uis.edu.co

I. Introduction

The foundations of object-oriented databases are designed to work with programming languages like Java object oriented, C #, Visual Basic. NET and C + +. For these languages a class can be a variable, a set of instructions or an object, but not a system as a group of classes sharing a common method. In the Teacher thesis Albarracin, this model is called MODRO. But for this article, MODRO will be called MODRES in Spanish or SYRDAM (SYSTEM REORIENTED DATA MODEL), because the MODRO actually is a Systems Reoriented Data Model. Now the SYRDAM is a model for integrating both the functional and structural of organization. To describe the functional there exist languages like Java, C #, C + +, Visual Basic. In addition, to describe the structural there exist SQL and OQL. But there is not a Language to described systems.

For the above languages and even the same OQL, a class is like an individual entity with its own attributes and its own method. The SYRDAM goes beyond focusing systems such as set of interrelated classes, sharing a common method, which is the behavior of the system (See Fig. 1).

II. Specification of Latin

A system described in Language Latin, is interpreted as classes of Db4o[1] . By means of the following steps.

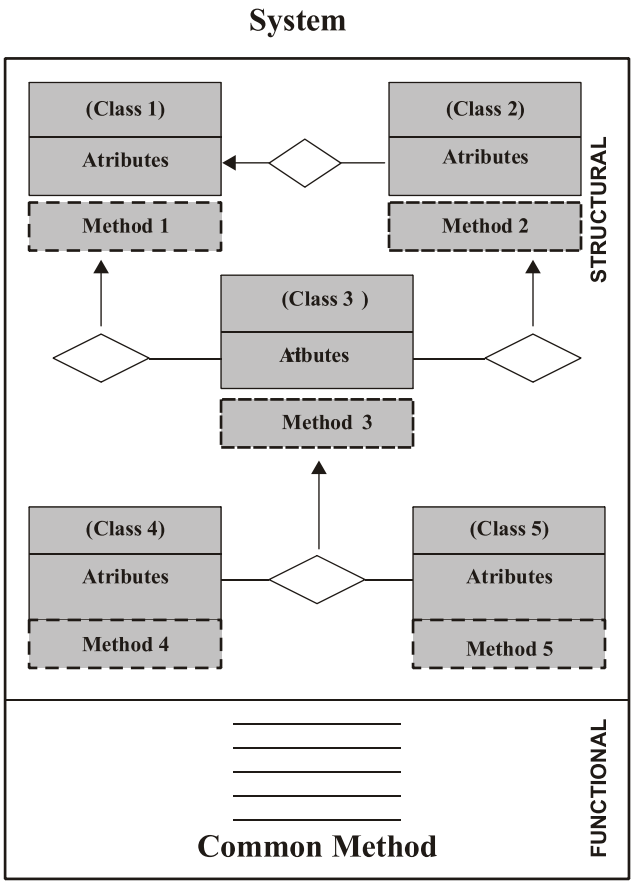

Fig. 1. Approach of the Models of System (MODRES)

[1] Db4o: Object to Orientated of database Engine.

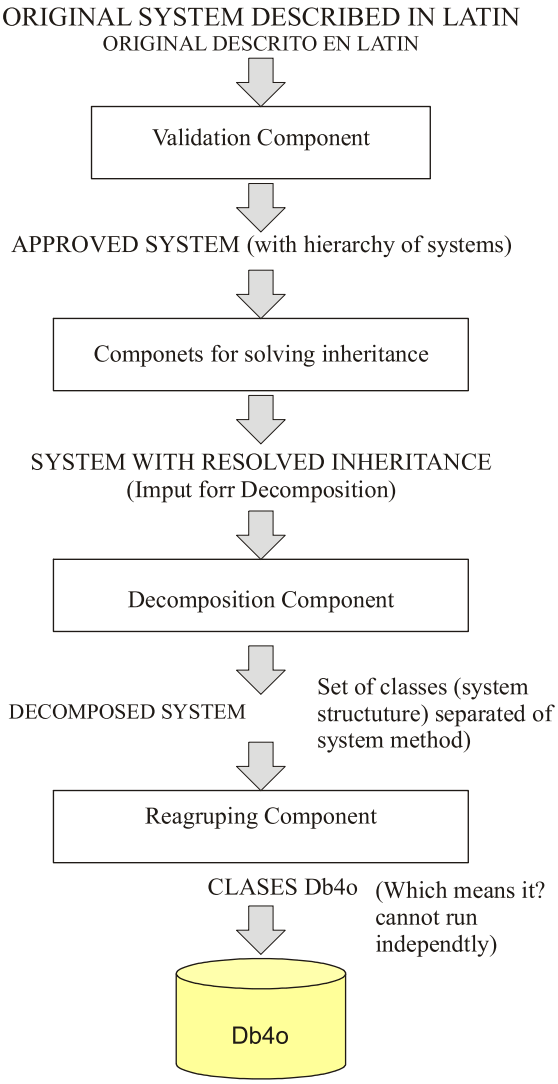

Fig. 2. Steps in the System interpretation for the LATIN

A. Validation Component

In this first step, LATIN recognize and validate by means of its grammar, the definition of the system, which consists of classes and a common method to those classes. Such validation will be to acknowledge the definition of the attributes with data types. It should also recognize the performance of integrity rules concerning primary and foreign keys, and other restrictions inherent in the SYRDAM **[1].** And in the same way to validate the method belonging to that system.

B. Component for solving inheritance

In this second step, the LATIN resolve the problem of inheritance that occurs when a system is defined within a hierarchy of systems (subsystems, systems, Supersystems).

C. Decomposition Component

Here it separates or descomposes common method classes, i.e classes and structures are separated from the common method. This separation is done to build

an appropriate entry for the OODBM's in the present time, unable to recognize systems.

D. Re-grupping Component

The next step is to rebuild again each of the classes in the system with his of slashes method. That is to say at this time each class will have its own method, its own functionality, but such classes can not be independent among since its should be execute all in simultaneity. To implement this concurrency, Db4o has a set of sentences, which lets it define classes with their attributes and method. Although these classes are defined first in the Interpreter Db4o through a flat file that contains such sentences.

E. Deliver classes to Db4o

In this step, LATIN delivers to Db4o through a connection, each of the classes extracted from the system. This seeks the implementation of oriented database systems (SYRDAM), lowering them to the level for object-oriented databases (Db4o).

III. Design of Latin Grammar

The creation of a model of data, implies the use of a language that allows to describe this model. But as such a language it doesn't still exist by reason of the biggest granularidad in the MODRES regarding the previous models, and also by reason of the best approach toward a paradigm of systems in the conception of the MODRES, it is necessary to create it, being this language LATIN outlined by the Doctor Albarracín. LATIN like such it requires of the definition components, manipulation and consultation of objects (ODL AND OML). Similar to their previous SQL of the pattern compound relational also of a DDL and a DML. For such a reason, it becomes necessary to design a Grammar free of context for this model that is able to generate the ODL and OML characteristic of LATIN.

For this grammar's design, the formal system will be used "BNF" or form of Backus-Naur **[2].** Keeping in mind the syntax SQL, the standard ODMG and the syntax used by the programming language guided to objects, like it is the case of the languages JAVA and C #.

But although it is certain that the first step to build LATIN, is to define the grammar, it becomes necessary also to describe the structure of the language that implements the component ODL and OML.

The structure that will have the INTERPRETER in question will contain the following articles **[3].**

- **Value literal.** They are the variables, numbers, it dates, hour and the values type NULL.
- **Allowed characters.** These are those special characters that all programming language allows to the moment to define an instruction, being those but common the following ones: (, *,), ', <, +, /, {, &,>, -, (,}, =).
- **Reserved words.** They are those that are only and characteristic of the language, not being possible to define a variable, constant, or I object with the name of these reserved words.
- **Comments.** These are accepted from a sequence '/ * ' until the maximum sequence '* / '.
- **Initial syntax.** Alone they accept a command of having entered per time and it will always finish in "; ".
- **Syntax of sentences ODL.** These are those that allow the definition of class hierarchies and their structural elements, the entities.
- **Syntax of sentences OML.** These are those that allow to define and to use a method, as well as to consult class hierarchies and their structural elements, the entities, and their functional elements, the operations of the methods.

IV. Lexical and Syntactic Implementation of Analyzers

The following one passes the grammar once it has been designed, it is to implement the lexical and syntactic analyzer. These allow to control the entrances typed for the user and they open the way to the generation of the structure of the files of hierarchy of classes. A lexical analyzer is that that

he takes the entrances typed and it divides them in lexical components, "that are sequences of characters that have an atomic meaning" **[4].** Once made this classification they are correspondents to the syntactic analyzer as TOKENS. The following figure shows like it is carried out this process.

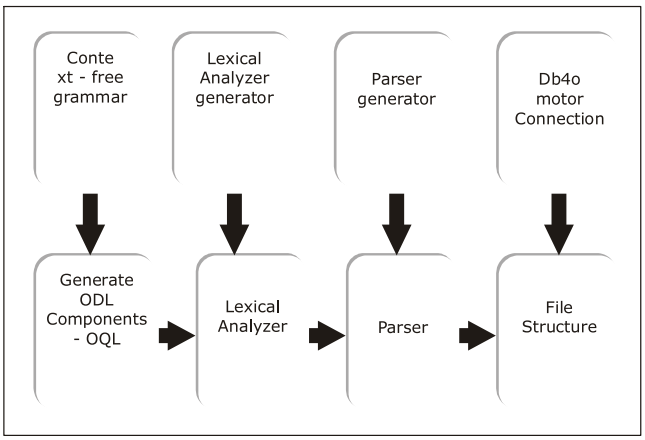

Fig. 3. Design of Interpreter "LATIN"

The chosen language for the INTERPRETER'S development is Visual C #, because this it is a language that offers many advantages, among which are had **[5]:**

- Offers the power of the design guided to objects with a simple syntax, easy to manage and a robust and pleasant environment for the user.
- It has in their structure a group of potent and flexible classes.
- Interoperability among languages, also called programming of mixed language.
- Excellent acting in work on the net.
- Excellent levels of security.

A syntactic analyzer is: "a program that he/she receives as entrance the individual elements or lexical components (tokens) of the program source and it determines the structure of the sentences or instructions that conform it" **[4].** It is for this reason that the analyzer's main functions are to detect and to inform of the syntactic errors and the generation of a syntactic tree for each one of the expressions typed for the user. The following figure shows the general outline of the syntactic analysis.

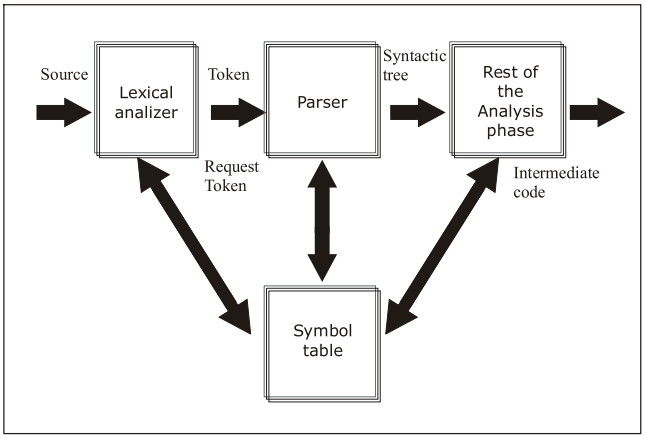

Fig. 4. General scheme of the syntactic analysis

V. Connecting the Latin With Db4o

The following code shows how LATIN, will connect to Db4o through a series of instructions specific of LATIN, which allow data to flow properly from LATIN to Db4o. This code shows, that first itmust make sure that if initially there is already a database with the same name, this should be eliminated. Following this step creates the database using the ObjectContainerdb owned by Db4o. We also observe that it uses the exception handling to insert, update and delete data from the database. Finally, we must close the database with the keyword Close () and the name of the connection string variable, in this case "bd". Consider the following code.

Connection String for LATIN with Db4o

```
using System;
using System.IO;
using com.db4o;
Using db4objects.db4o.Database;
public static void Main (String [] args)
{
File.Delete("C:/database_name.yap"); //reset database
ObjectContainerdb= b4o.OpenFile("C:/database_name.yap");
 try {
 {
 //Instructions to insert, update and delete
 }
 finally {
 {
 db.Close();
 } } } }
```

VI. Design of the Gui

The interface that will be created for the Latin, it will be following a traditional style and design. In the construction of this interface is used the development environment of Visual C #, see figure below.

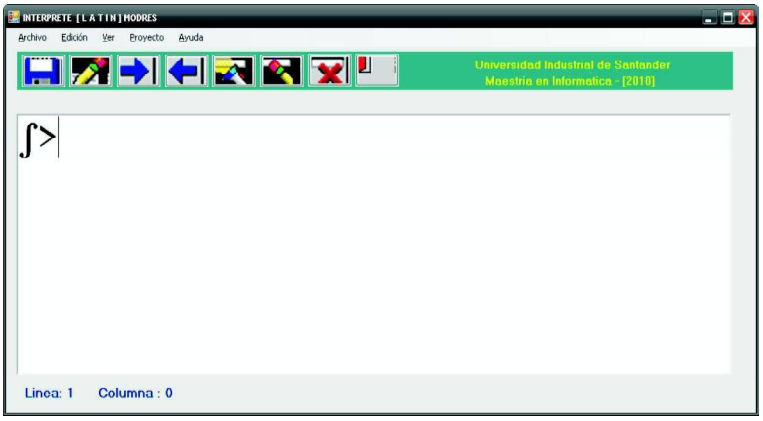

Fig. 5. Interface graph of the Interpreter LATIN

VII. Tests of Latin

One of the most important stages in the development software is undoubtedly the stage tests, in which the effective acting of the tool is verified. LATIN like such it is not unaware to this stage, because in her each one of their components will be proven (lexical and syntactic Analyzer). The procedure to continue to carry out this type of tests is the following one:

- To execute the Interpreter.
- To analyze the acting of the Analyzers (Lexicon and Syntactic).
- To observe and to register the obtained results.

Next the process of tests is explained that will be made the Interpreter's lexical and syntactic analyzers LATIN. As well as the variables to keep in mind to execute this test. For I finish the respective connection of LATIN he/she settles down with Db4o, for the shipment of the classes.

A. Process of test of the Lexical Analyzer

They will be carried out three types of tests:

- The first one will consist on entering a series of chains of characters been worth by the grammar.
- For the second test chains of characters will not be entered allowed inside the grammar.
- For the finish chains of worth characters and invalids they will be entered.

B. Process of test of the Syntactic Analyzer

To compile, to execute and to develop the test to the syntactic analyzer will become necessary to create the following class in C #:

```
public class static void Main (String [] args)
{
 /* Initial analyzer * /
 try {
 {
 parser p = new parser
 (new Lexer
 (new FileReader (args [0])));
 Object result = p.parse ().value;
 }
 catch (exception e)
 {
 e.printStackTrace();
 } } }
```

This class receives as parameter a file of plane text with the chains typed for the user and carries out the connection among the lexical and syntactic analyzer.

For the development of these tests it will be considered the syntax used in the description of the Supersystems, Subsystems and Systems characteristic of the grammar.

If the results generated by Interpreter LATIN are in agreement with the investigation, you will proceed to the connection and shipment of the Supersystems, Subsystems and Systems to Db4o.

C. Process of connection test with Db4o

The connection with the database motor Db4o is carried out by the Interpreter LATIN after verifying the correct definition of the Supersystems, Systems, Subsystems, as well as the manipulation of data by means of the instructions characteristic of LATIN (**ELEGI, MUTA, ADDIDI AND DELEGE**)[2].

For the realization of the connection tests with Db4o the Visual tool C will be used #, since with this the graphic interface will be created. Alone coarse to make click with the right button of the mouse in references (References) to add references, as it shows it the following figure.

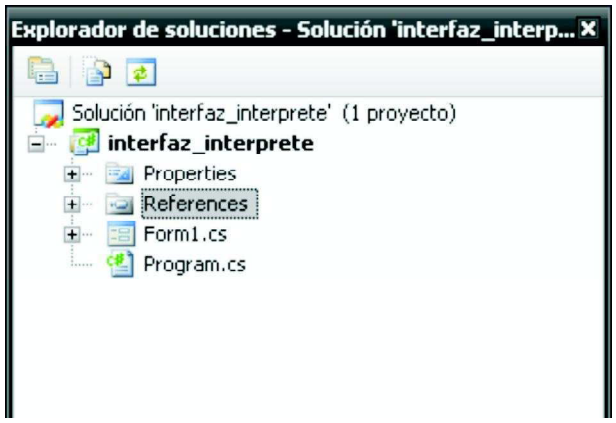

Fig. 6. References of Visual C#

[2] ELEGI, MUTA, ADDIDI Y DELEGE they are the equivalent Latins of Select, Update, Insert y Delete.

The purpose of adding this reference is the one of establishing the connection of C #with Db40, by means of the bookstore "Db4object.Db4o.dll". Since this bookstore is the one used by Db40 to be connected with the programming languages like C #, Java, Visual C++, to the moment to create a database. Later to this is given to examine, to find the portfolio Db4objects, such and as it shows it the following figure.

With the left button it occurs double click on the portfolio db4o-7.4, to locate the portfolio net.2.0 and in her to find the bookstore Db4object.Db4o.dll. To this bookstore is given double click to show the editor, to see following figure.

This way it is clever the connection with Db4o, to begin with the tests, such and as it shows it the following figure.

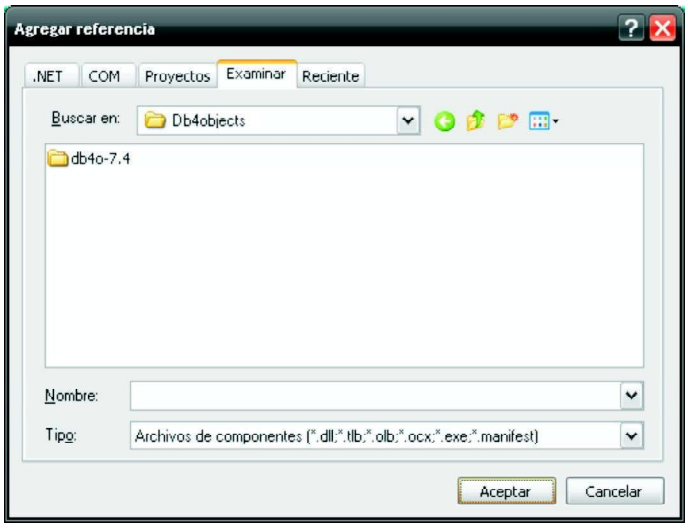

Fig. 7. Add Visual reference C #

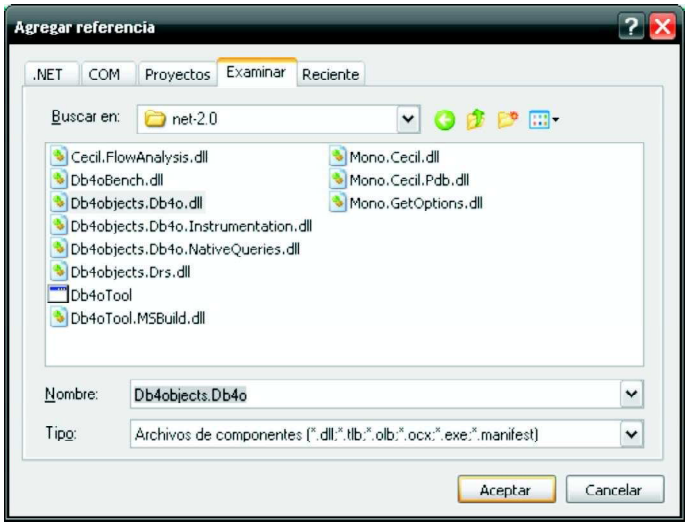

Fig. 8. Add reference (Continuation) Visual C #

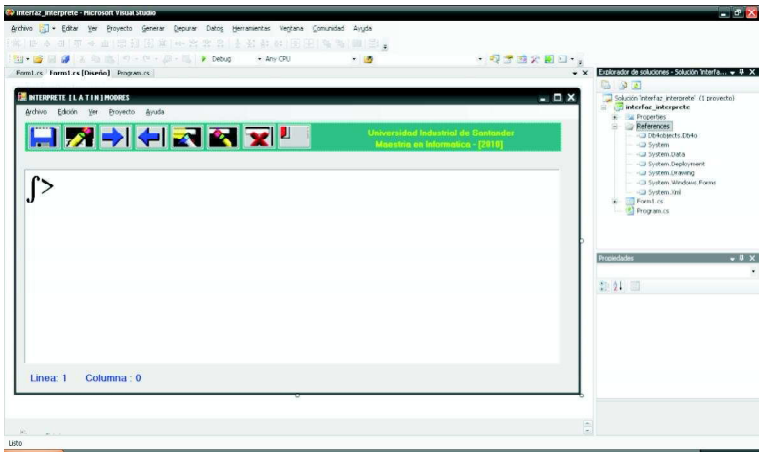

Fig. 9. Screen of connection of Visual C# with Db4o

VIII. Conclusions

1. The specification in the interpreter's detail LATIN will allow to implement computationally the Reoriented Pattern of Data to Systems (MODRES).

2. The interpreter LATIN, will allow to define Supersystems, Subsystems and Systems.

3. The interpreter LATIN, will allow to solve the problem that the languages like Java present, Visual C++, C #, Visual Basic and other that don't allow to define Supersystems, Subsystems and Systems.

4. The interpreter LATIN, will allow to solve the problem of the organizational disintegration in any organization.

5. The interpreter LATIN, offers another programming alternative, for the databases guided to objects.

References

[1] J. O. Abarracín Ferreira, *Integration of Data and processes in the organizations by means of a Objects Reoriented Data Model*. Doctoral Thesis; University of Oviedo. Spain. 2006.

[2] C. Lauden Kenneth. *Compilers of Construction.* Editorial Thomson. ISBN: 970-686-299-4; 2001.

[3] F. Contreras Herazo, A. S. García Payares, J. N. Cala Uribe, *Construction of those component ODL and OQL for a generating system of Reoriented databases to Objects (SGBDRO)* , Thesis of Pregrado. Universidad Industrial de Santander, Bucaramanga –Santander, Colombia–, 2008.

[4] S. Gálvez Rojas, M. Mata, *Java to End: Translators and Compilers*. Universidad de Málaga –España–, 2005.

[5] H. Schildt. *C# Manual of Reference*. MacGraw-Hill. ISBN: 84-481-3712-4; 2003.