Revista Colombiana de Computación, 2018, Vol 19, No. 1, pp. 39-55
https://doi.org/10.29375/25392115.3228

A Tangible System for Learning Relational Algebra

Un sistema tangible para el aprendizaje de álgebra relacional

Antonio Xohua-Chacón[1], Edgard Benítez-Guerrero[1], Carmen Mezura-Godoy[1]

1 Faculty of Statistics and Informatics, Universidad Veracruzana, Xalapa, Veracruz, México
axohua@uv.mx, edbenitez@uv.mx, cmezura@uv.mx

(Received: 1 December 2017; Accepted: 17 January 2018)

Resumen. Las interfaces tangibles de usuario (TUIs) son aquellas en donde el usuario interactúa con un sistema digital a través de la manipulación directa de objetos físicos (tokens), ligados directamente a cierta funcionalidad/datos dentro del sistema, por lo que su manipulación afecta el comportamiento del sistema. Este artículo presenta TanQuery, un sistema tangible para ayudar en el proceso de aprendizaje de Algebra Relacional. TanQuery incorpora componentes de detección y seguimiento de tokens, para analizar y ejecutar árboles de consulta. El sistema fue probado por estudiantes universitarios y, como se puede observar por los resultados obtenidos, ellos encontraron útil y agradable este tipo de interfaz.

Palabras clave: interfaces tangibles, álgebra relacional.

Abstract. Tangible User Interfaces (TUIs) are those in which users interact with a digital system through the direct manipulation of physical objects (tokens). Tokens are directly linked to a certain data/functionality within the system, so manipulation of these objects affects the system behavior. This paper introduces TanQuery, a tangible system to support the process of learning Relational Algebra. TanQuery incorporates components to detect and track tokens, and to analyze and execute query trees. The system was tested by university students, and obtained results allowed to observe and analyze that students found this type of interface useful and pleasant.

Keywords: Tangible user interfaces, relational algebra.

1. Introduction

Over the last decades, Tangible User Interfaces (TUIs) [1] have become an important approach to the interaction between users and computer systems. In this kind of interface, users interact with a digital system through direct manipulation of physical objects linked to a system functionality, causing that the manipulation of these objects affects the behavior of the system. The types of interaction between users and computers have evolved over time thanks to several upgrades in computer technology and communications. These technological breakthroughs not only include traditional advances such as command line interfaces, WIMP systems (windows, icons, menus, pointer) [2], but also recent advances such as interaction through objects (TUIs). TUIs have been used in different application domains [3], for example in educational settings, where tangibility and its potential benefits for learning have been explored [4].

In the TUI research domain it is possible to find works related to databases, such as [5], [6], [7]. However, to the best of our knowledge, there is no research or work that deals with the subject of relational algebra (RA). Relational algebra is a query language associated to the relational database model which includes operators based on set theory. Such operators allow one to manipulate relations (tables) stored in databases. This language is important because it facilitates the understanding of some of the SQL language constructions that are basis for query processing in database management systems [8].

Tools to support the learning process of relational algebra have been developed as a consequence of the topic's level of difficulty. Although students can understand how RA works by using paper-based materials, tools that enable students to explore and understand interactively its concepts, are more useful. An interactive system can help the students to see the results of a RA expression and to enable them to learn through exploration.

On the one hand, there are some applications as WinRDBI [9], iDFQL [10], RALT [11], DBSnap [12], RAPT [13] that cover RA topics and that have different ways to achieve them; for example, building algebraic expressions through trees or using graphical elements to represent the queries. However, the type of interaction presented in those tools is WIMP type. On the other hand, database-related works that use tangible interaction do not cover topics of RA and they are not focused on the educational field. The aim of our work is then to propose a solution based on a tangible interface to support the learning process of RA.

This paper introduces TanQuery, a tangible system that supports the learning of RA. In this system, RA operators and operands are represented by tokens that can be manipulated by the user to interactively create query trees. For educational purposes, TanQuery generates equivalent RA and SQL expressions from the query trees built by the user and executes query trees on data to get a result. The design of TanQuery incorporates components to detect and track tokens, and to analyze and execute query trees.

The paper is organized as follows: first, Section 2 introduces the modeling of the TUI; then, Section 3 describes the system architecture and its components; section 4 explains system implementation; after that, Section 5 presents the system test, while Section 6 discusses results and section 7 discusses related works. Finally, Section 8 concludes this paper and discusses future work.

2. TUI modeling

This section presents the modelling for TanQuery's tangible user interface. This modelling is described in terms of the TAC paradigm, so it is briefly explained (see [14] for details). The rest of the section explains the elements of the TUI and its relationships.

2.1 TAC paradigm

The TAC paradigm was proposed by Shaer et al. [9] to describe the structure and the functionality of a TUI in terms of the following elements: Pyfo, Token, and Constraint, which are explained below.

A Pyfo is a physical object that takes part in a TUI, and that it may be comprised for a number of other pyfos; for example, a cube can have six pyfos or sides. This term was suggested as a way to avoid the term "physical object" because it may lead to confusion with the common use term "physical object", that makes reference to elements of the physical world, and that has no connection with TUIs. For instance, imagine a table with a book and a cell phone on it, both of these are physical objects; however, if only the book is used to interact with the TUI system then it is a pyfo and the cell phone is merely a physical object.

A *Token* is a graspable pyfo that represents digital information or a computational function within an application. The user interacts with the token so as to access or manage the digital information. Taking up the example showed above, let us suppose that the book is located on a specific area of the table. The system links that pyfo with a system variable (at that moment the pyfo becomes a token) that can be altered by manipulating the book.

A *constraint* is a pyfo that limits the behavior of the token it is associated with. The physical properties of the constraint guide the user to understand how to manipulate the token and also show the user how to interpret token compositions and constraints. In the example presented above, the size of the table can be a constraint to the book, if it is considered that the book can only be used on it.

2.2 TanQuery Pyfos design

We considered physical features in pyfos design, such as color, texture, weight, size, and shape. It is important to avoid some possible issues as user fatigue due to the use of a token with inadequate weight, or to reach a negative mental state due to the color of the token.

As mentioned above, a pyfo becomes a token when it is associated with a digital variable in the system. In TanQuery, pyfos are pieces of wood that are identified by color (see Figure 1) depending on the category to which they belong, and additionally they are labeled with a fiducial marker that identifies them uniquely within the system. The pyfos categories are listed below:

Database (black): A database pyfo is used to select the database that is going to be used during the interactions. To assign its value, it must be placed in the assignment area and then it has to be rotated it order to select a new value. If the value of this token is changed, the value associated with each token is deleted.

Relation (green): This category represents a relation (table) within a database. In order to assign a value to a relation pyfo, it is necessary to previously assign a value to the token representing the database. To associate a value to a relation pyfo, the pyfo needs to be placed in an assignment area and then it has to be rotated it, in order to select the value (the name of the relation).

Attributes (blue): A pyfo of this category is used to represent the attributes that compose the relation schema. The assignment of a value for this kind of pyfo depends on the presence a relation pyfo has associated with a value, in order to be able to select the values of the attributes that belong to that relation.

Constants (brown): These represent possible values of a particular attribute; for example, the possible values for attribute "username" could be "John" or "Mary". A constant pyfo depends on an attribute token.

Relational operators (yellow): These ones refer to relational operators such as union, project, select, and semi-join. It is possible to assign a value to each of these pyfos without depending on another token.

Comparison operators (red): They represent operators that can be used to test whether one value is equal to, greater than, or less than another. As well as relational operators, they do not depend on another token. Their available values are: ">" (greater than), "<" (less than), ">=" (greater than or equal to), "<=" (less than or equal to), "=" (equal to), "!=" (different).

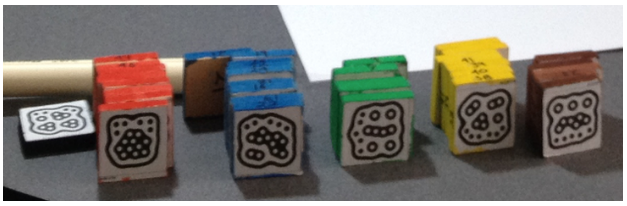


Figure 1. Sample of pyfos used for user and system interaction.

2.3 Main Interactions

In TanQuery, there are two main interactions that can be made with tokens (see Figure 2) rotation and touch. These terms will be explained bellow.
Rotation: It is useful when it is required to assign a value to the pyfo; for example, to refer to a particular relational operator or a relation.
Touch: This interaction is used to build two relationships 1) parent-child to indicate that a token is the parent of another. It can only be established if the token parent is a relational operator and the children token are relations or relational operators, 2) operator-condition touch is used to add the condition to a relational operator as a parameter; for instance, in Figure 2, if "A" represent a relational operator, "B", "C" and "D" can be attributes, comparison operator or constants.

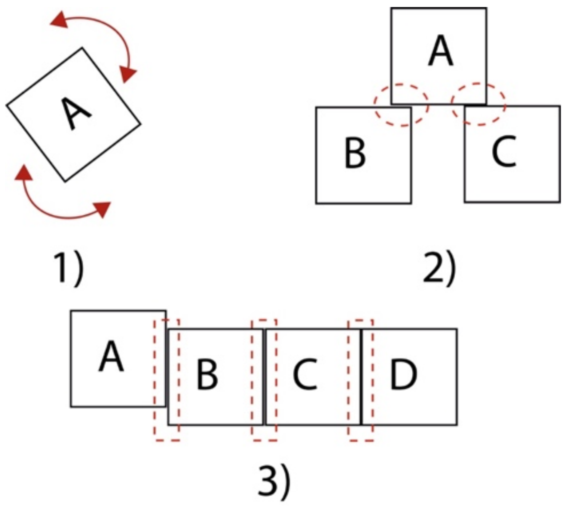


Figure 2. Interactions: 1) Rotation 2) Touch to add a parent-child relationship 3) Touch to add a condition to relational operator.

2.4 Logical constraints

To assign a value to a pyfo (to turn the pyfo into a token), there are different restrictions that must be complied depending on the type of pyfo in question.

Coupling: The coupling of a variable with an pyfo can be done 1) by the designer at design time; for example, the list of values for comparison operators are predefined to ">", "<", ">=", "<=", "=", "!=" and these values cannot be changed, 2) by the user at runtime when he/she assigns a value to a relational pyfo; for example, "Students", and in this case this value can be overwritten.

Relative Definition: It refers to existing constraints between pyfos when obtaining their associated value. For example, to assign a value to a pyfo relation, the database must be selected first to obtain a list of possible relations to choose from. This also happens with attribute pyfos, where it is necessary to know the token relation in order to retrieve its attributes list. Tokens that are also constraints are: database (relation token depends on a database one), relation (an attribute token depends on a relation value) and attribute (a constant depends on an attribute value).

Association: It is when a token is physically associated to a constraint, during query tree creation, new constraints are created; for example, the relation between parent node (relational operator) and child node (relational operator).

Computational interpretation: This is the way in which the manipulation of tokens is interpreted regarding constraints. When tokens are in a correct order, the system shows results retrieved from a database server; however, if the tree created is not coherent, the system will not show any results.

3. System architecture

Figure 3 shows the architecture of TanQuery. The user interacts with the system by means of tokens and in return he/she obtains feedback from the system by means of RA and SQL expressions, and data. The main components of the system are:

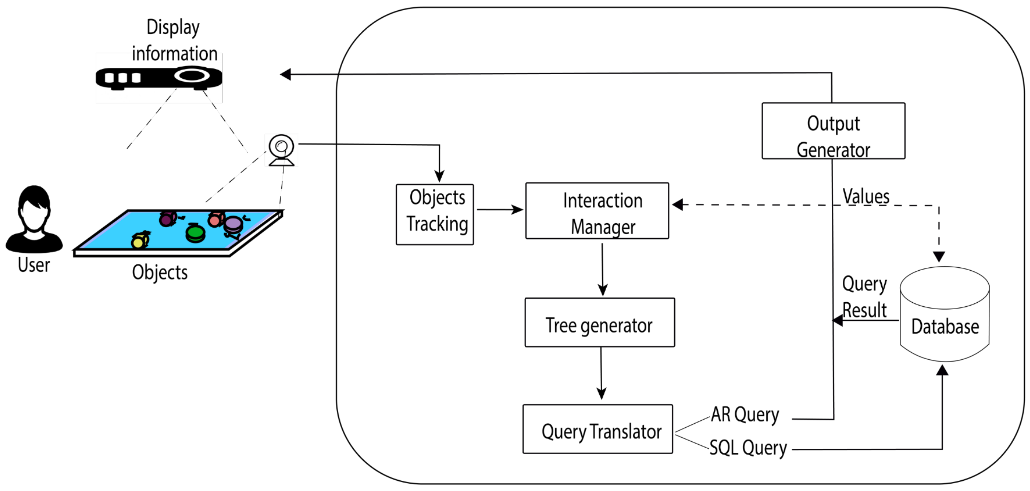


Figure 3. TanQuery Architecture.

Objects Tracking: It detects pyfos and tracks all of them over the surface.

Interactions Manager: The two main types of interactions (see Figure 2) are identified and filtered by this component, and subsequently to be sent to a tree generator when these are performed.

Tree generator: The workspace is divided into three areas: assignment area, work area and results area. Tree generator reacts to the interactions performed by user on the work area. When there is only one object of relation type, this one is taken as if it was the root of the query tree. Then, the system shows contents for that relation and their equivalent AR and SQL expressions in the results area. When tokens touch, nodes are added to the tree. These can be either be children nodes or conditions for the relational operator located next to the relational operator.

Query translator: This component creates two tree traversals. The first one is in charge of generating equivalent AR expressions, and the other one is in charge of constructing its equivalent expression in SQL language. The latter is sent for execution to the database server (can be local or remote), and the results of such query are projected on the work surface, along with the expressions in relational algebra and SQL.

Output generator: It processes information (token values, messages, data sets, SQL and RA queries and other messages) and displays information-related terms on the surface.

4. Implementation

The system was implemented using hardware and software elements that are going to be described below.

4.1 Hardware

In Figure 4, one can see the hardware setting used in the system: one mini display port to HDMI cable adapter, one Microsoft LifeCam hd-300 webcam, one Optoma DS316L projector, one TV stand, one meter of 1/4 pvc tube, one desk table with white cardboard, some pieces of Medium Density Fibreboard (MDF) such as pyfos, and additionally a computer MacBook Pro with Processor Core i5 2.5 GHz and 8 GB RAM.

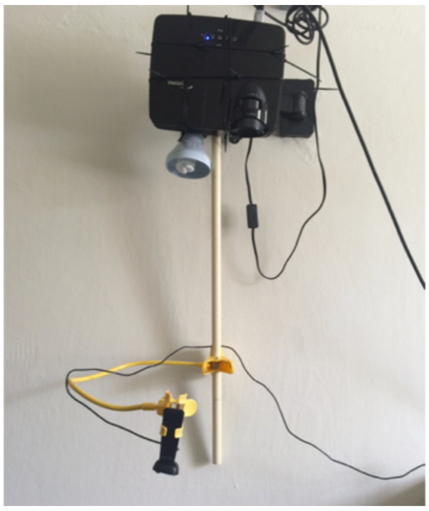


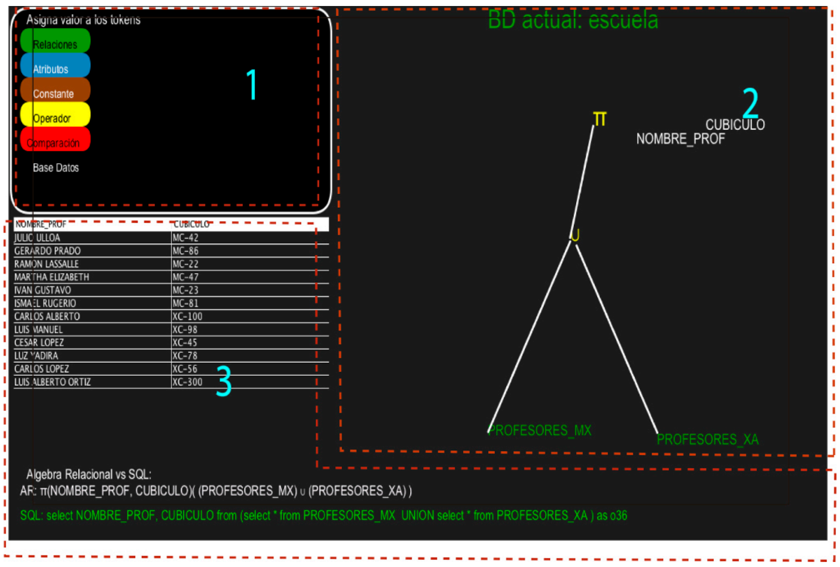
Figure 4. Hardware used in system.

Source: Infraestructura para el Desarrollo de Interfaces de Usuario Tangibles en Escritorios Inteligentes [15].

4.2 Software

TanQuery was developed using ECLIPSE programming environment Luna Version: Luna Service Release 2 (4.4.2), Java 1.7.0_67 language, Processing version 2.2.1, reacTIVision framework 1.5 (1410). As database server we used MySQL (version 5.5.42). The system implementation is described as follows:

Workspace distribution: The workspace is divided in three areas for user interaction (see Figure 5). (1) Assignment area: a pyfo is placed in this area in order to assign a value to it (depending on the type of pyfo, the system generates its possible values), and the user must rotate the pyfo until the desired value is displayed on the surface. (2) Work area: this is where the user builds the query tree executing connections between child and parents nodes. (3) Results area: this is where the results are displayed (in form of a chart) generated from the tree built by the user, and in the lower part both AR and SQL expressions are also displayed.



NOMBRE_PROF	CUBICULO
JULIO ULLOA	MC-42
GERARDO PRADO	MC-86
RAMON LASSALLE	MC-22
MARTHA ELIZABETH	MC-47
IVAN GUSTAVO	MC-23
ISMAEL RUGERIO	MC-81
CARLOS ALBERTO	XC-100
LUIS MANUEL	XC-98
CESAR LOPEZ	XC-45
LUZ YADIRA	XC-78
CARLOS LOPEZ	XC-56
LUIS ALBERTO ORTIZ	XC-300

Algebra Relacional vs SQL:
AR: π(NOMBRE_PROF, CUBICULO)((PROFESORES_MX) ∪ (PROFESORES_XA))
SQL: select NOMBRE_PROF, CUBICULO from (select * from PROFESORES_MX UNION select * from PROFESORES_XA) as o36

Figure 5. Workspace distribution: 1. Assignment area, 2. Work area, 3. Results area.

Token Detection and tracking: The reacTIVision engine [16] acquires images from the camera. Then, it searches video stream for fiducial markers and sends (using its own communication protocol TUIO [17]) data about all identified symbols by means of a network socket to a listening application (in our case TanQuery, which implements a TUIOClient). The client application retrieves a list with that information calling the *tuioClient.getTuioObjectList()* method, and then it can get information of each element (see Figure 6).

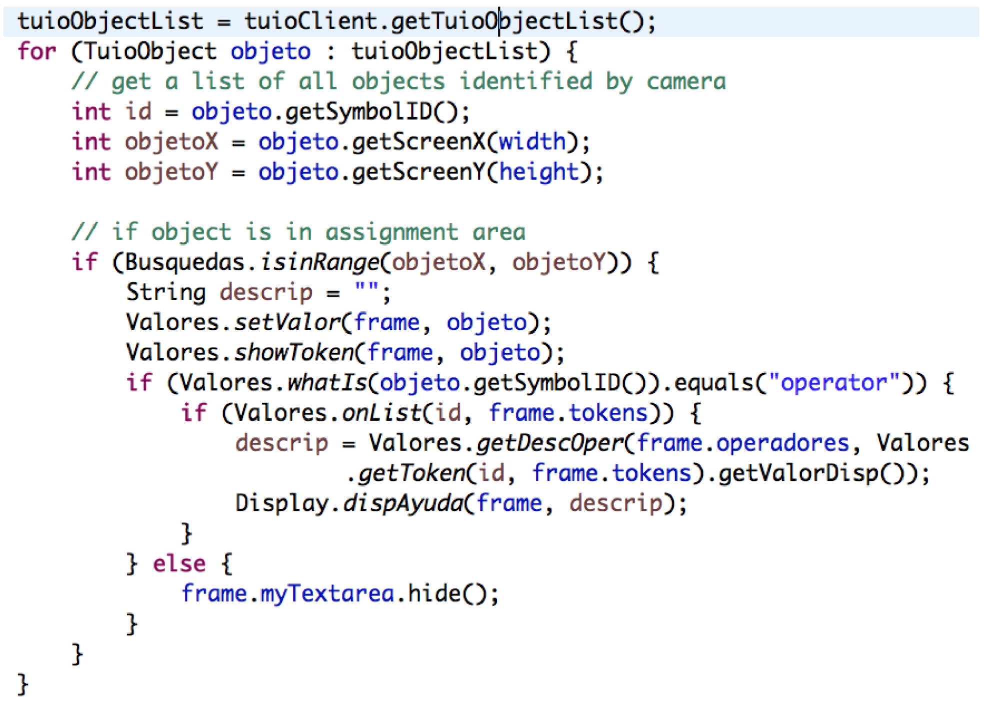
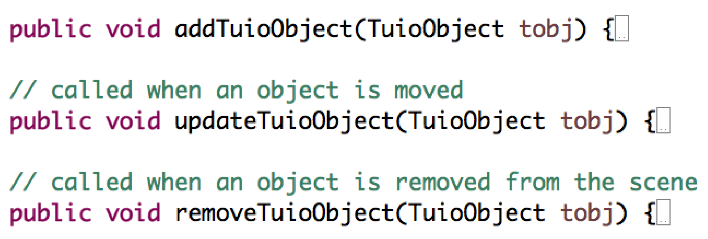

```
tuioObjectList = tuioClient.getTuioObjectList();
for (TuioObject objeto : tuioObjectList) {
    // get a list of all objects identified by camera
    int id = objeto.getSymbolID();
    int objetoX = objeto.getScreenX(width);
    int objetoY = objeto.getScreenY(height);

    // if object is in assignment area
    if (Busquedas.isinRange(objetoX, objetoY)) {
        String descrip = "";
        Valores.setValor(frame, objeto);
        Valores.showToken(frame, objeto);
        if (Valores.whatIs(objeto.getSymbolID()).equals("operator")) {
            if (Valores.onList(id, frame.tokens)) {
                descrip = Valores.getDescOper(frame.operadores, Valores
                        .getToken(id, frame.tokens).getValorDisp());
                Display.dispAyuda(frame, descrip);
            }
        } else {
            frame.myTextarea.hide();
        }
    }
}
```

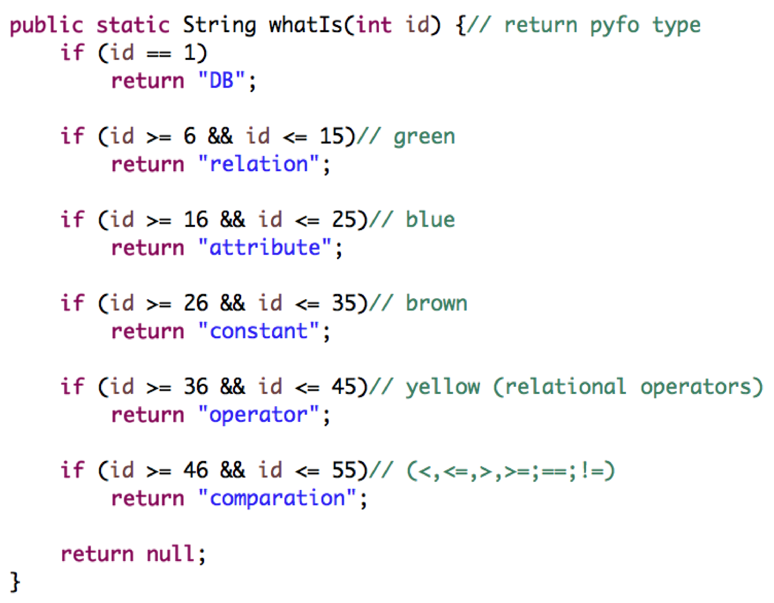
Figure 6. Code to get the list of fiducial markers on the workspace.

Figure 7 shows three main methods used in TanQuery for identifying object events. *addTuioObject* method is called when user adds a pyfo to the surface*,* while *updateTuioObject* method is executed when a pyfo is manipulated on the table (rotated, moved). Finally *removeTuioObject* method is activated when the reacTIVision engine detects an object that was removed from the surface. The obtained data from the reacTIVision makes reference to pyfo's location within the x and y axes, the angle and speed of the pyfo's motion, as well as, a unique identifier to recognize the pyrfo in the system. Tokens are classified in categories according to their id (see Figure 8) as follows: (i) 1 = "Database", (ii) 6 - 15 = "relation", (iii) 16 - 25 = "attribute", (iv) 26 - 35 = "constant", (v) 36 - 45 = "relational operator", and (vi) 46 - 55 = "comparison operator".



```
public void addTuioObject(TuioObject tobj) {

// called when an object is moved
public void updateTuioObject(TuioObject tobj) {

// called when an object is removed from the scene
public void removeTuioObject(TuioObject tobj) {
```

Figure 7. Principal reacTIVision methods used to identify object events.

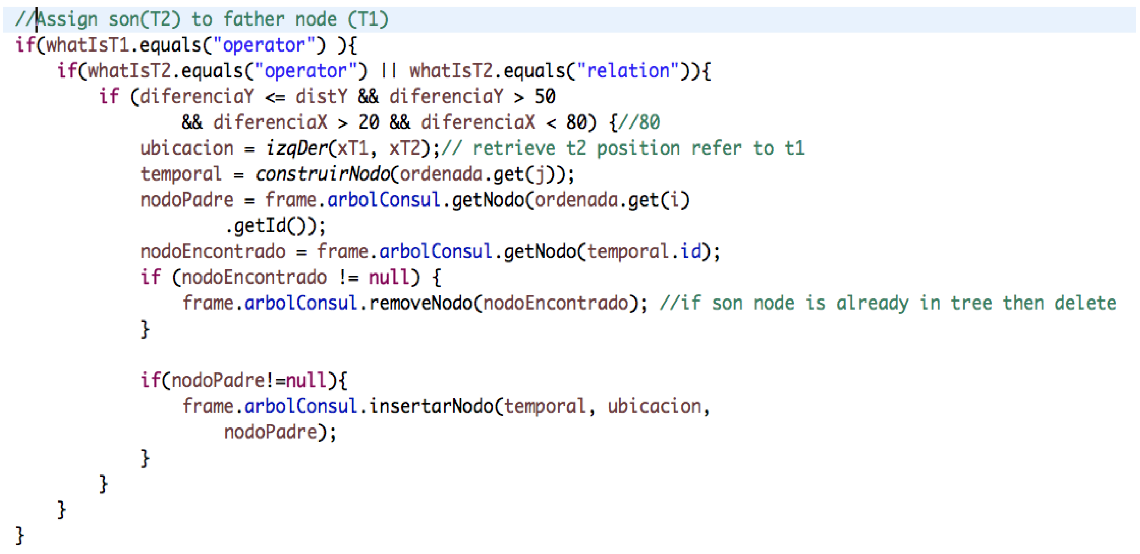


```
public static String whatIs(int id) {// return pyfo type
    if (id == 1)
        return "DB";

    if (id >= 6 && id <= 15)// green
        return "relation";

    if (id >= 16 && id <= 25)// blue
        return "attribute";

    if (id >= 26 && id <= 35)// brown
        return "constant";

    if (id >= 36 && id <= 45)// yellow (relational operators)
        return "operator";

    if (id >= 46 && id <= 55)// (<,<=,>,>=;==;!=)
        return "comparation";

    return null;
}
```

Figure 8. Classification of pyfos by range.

Interaction detection: To detect user-token interactions, the system uses the token location within the workspace (assignment or work area) and then it acts accordingly. For example, when a token is rotated in the assignment area, its possible values are retrieved from the database (as a result of a query issued to the database server), and then the 360 degrees are divided by the total number of possible values in order to obtain the number of degrees between each value. However, if the rotation is done outside the assignment area, then the value of the token is not modified.

In the case of touch interaction, the distance between all tokens, identified by the camera, is computed. Distance ranges are predefined in order to know if two tokens are "touching". When the system identifies a touch, this one needs to know which side is touching; and to do so, the lowest value token located in the y axis is taken as reference (parent), meanwhile the left or right location of the other token that is touching is calculated, see Figure 9.



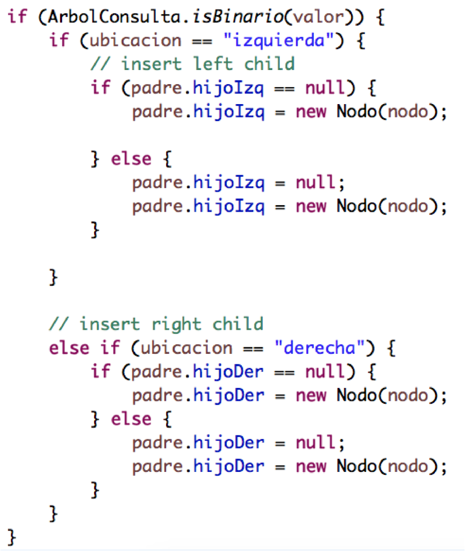
```
//Assign son(T2) to father node (T1)
if(whatIsT1.equals("operator") ){
    if(whatIsT2.equals("operator") || whatIsT2.equals("relation")){
        if (diferenciaY <= distY && diferenciaY > 50
                && diferenciaX > 20 && diferenciaX < 80) {//80
            ubicacion = izqDer(xT1, xT2);// retrieve t2 position refer to t1
            temporal = construirNodo(ordenada.get(j));
            nodoPadre = frame.arbolConsul.getNodo(ordenada.get(i)
                    .getId());
            nodoEncontrado = frame.arbolConsul.getNodo(temporal.id);
            if (nodoEncontrado != null) {
                frame.arbolConsul.removeNodo(nodoEncontrado); //if son node is already in tree then delete
            }

            if(nodoPadre!=null){
                frame.arbolConsul.insertarNodo(temporal, ubicacion,
                    nodoPadre);
            }
        }
    }
}
```

Figure 9. Adding child node to a relational operator.

4.3 Tree construction.

A query tree is built by identifying token-user interactions within the work area. At this point, additional conditions must be satisfied, apart from the location of a token, there are conditions to be analyzed, such as validating the type of a token (relation, relational operator or comparison, attribute, constant) that participates in the interaction.

A parent-child relationship can only occur when the parent token is a relational operator and its children are relational operators or relations. If the parent is a binary relational operator (for example union, join, semi-join), it is possible to assign both left and right child (see Figure 10).



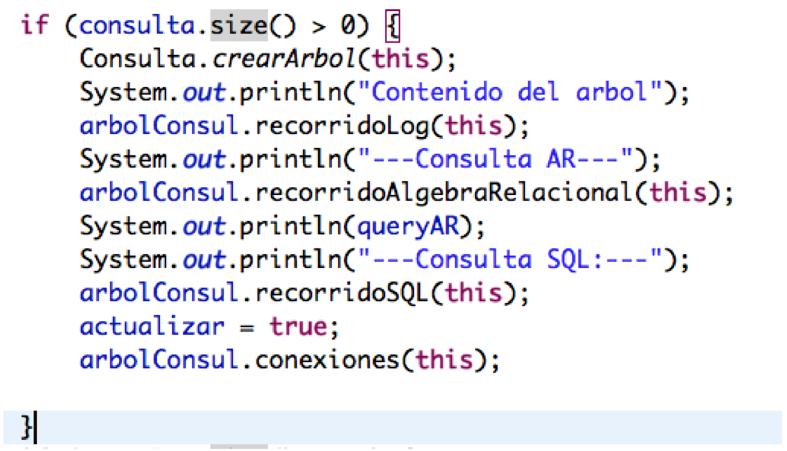
```
if (ArbolConsulta.isBinario(valor)) {
    if (ubicacion == "izquierda") {
        // insert left child
        if (padre.hijoIzq == null) {
            padre.hijoIzq = new Nodo(nodo);

        } else {
            padre.hijoIzq = null;
            padre.hijoIzq = new Nodo(nodo);
        }

    }

    // insert right child
    else if (ubicacion == "derecha") {
        if (padre.hijoDer == null) {
            padre.hijoDer = new Nodo(nodo);
        } else {
            padre.hijoDer = null;
            padre.hijoDer = new Nodo(nodo);
        }
    }
}
```

Figure 10. Add child to binary operator.

An attribute token can only be used to add a condition to a relational operator, comparison operators, constants or another attribute (lateral touch, see Figure 2-3).
When the conditions mentioned above are satisfied, the tree is created. If a token of type relational operator is added in the upper part, this is considered as a new root of the tree, then the latter one creates one connection to the previous root.

4.4 Query generation

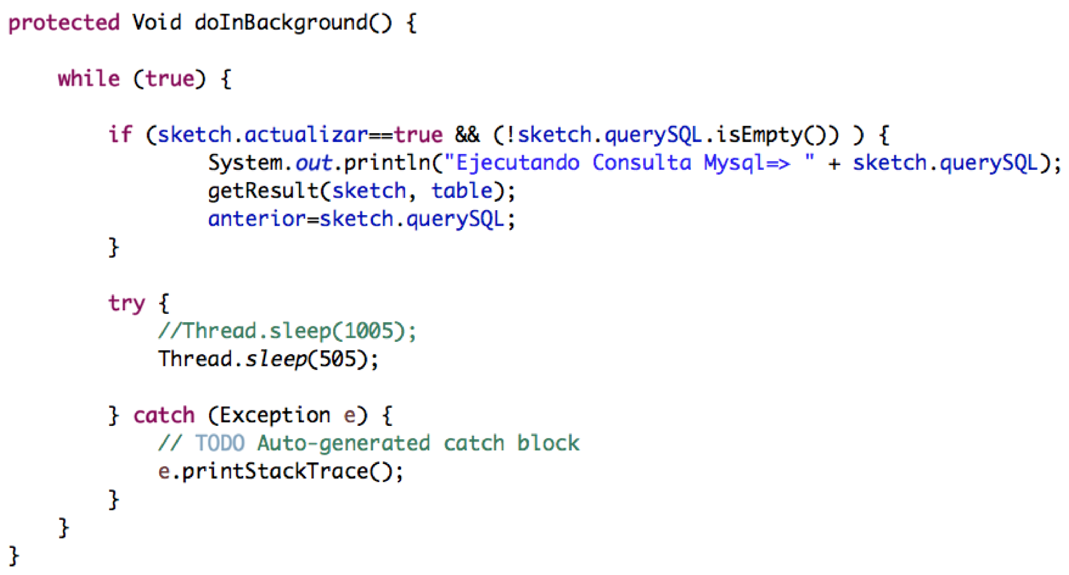
Once a tree has been created, a traversal is made in order to obtain an equivalent expression in relational algebra. Another tree traversal is done to generate the query in SQL language, which is sent to the database server so to get the results of the query (see Figure 11).



```
if (consulta.size() > 0) {
    Consulta.crearArbol(this);
    System.out.println("Contenido del arbol");
    arbolConsul.recorridoLog(this);
    System.out.println("---Consulta AR---");
    arbolConsul.recorridoAlgebraRelacional(this);
    System.out.println(queryAR);
    System.out.println("---Consulta SQL:---");
    arbolConsul.recorridoSQL(this);
    actualizar = true;
    arbolConsul.conexiones(this);

}
```

Figure 11. Tree traversal to get both AR and SQL queries.

The results are finally projected into the workspace. If the results are empty, then the result area remains unchanged (see Figure 12).



```
protected Void doInBackground() {

    while (true) {

        if (sketch.actualizar==true && (!sketch.querySQL.isEmpty()) ) {
                System.out.println("Ejecutando Consulta Mysql=> " + sketch.querySQL);
                getResult(sketch, table);
                anterior=sketch.querySQL;
        }

        try {
            //Thread.sleep(1005);
            Thread.sleep(505);

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

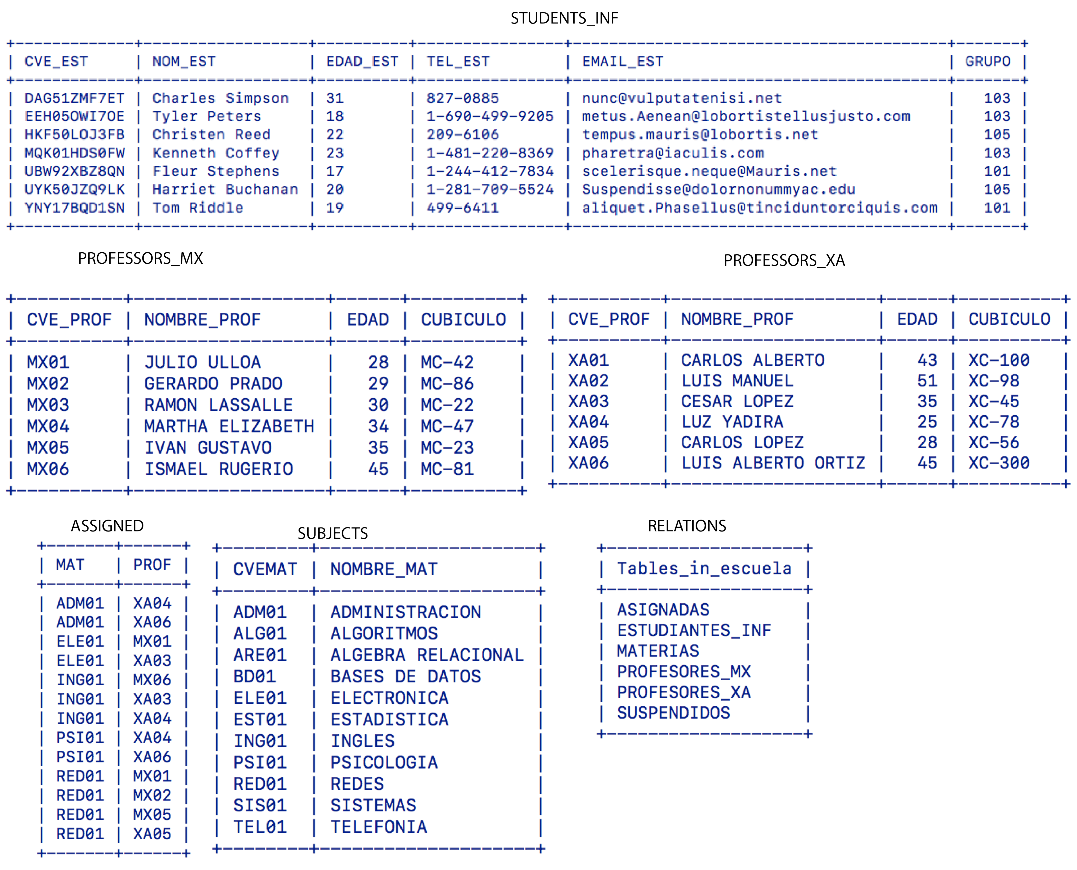
Figure 12. SQL query monitor.

5. Testing

The aim of the test was to obtain qualitative information about the satisfaction of potential users while using the system. The participants were asked to solve five exercises related to relational algebra with different complexities (taking as reference the number of tokens needed to solve it). At the end of test, an interview was conducted with the participants. The objective of this one was to obtain relevant information about the system.

5.1 Data

The default database used by system is “SCHOOL”; however, the user can select any database in the server by manipulating a database pyfo. Figure 13 show the fictitious data stored in the SCHOOL database.



STUDENTS_INF

CVE_EST	NOM_EST	EDAD_EST	TEL_EST	EMAIL_EST	GRUPO
DAG51ZMF7ET	Charles Simpson	31	827-0885	nunc@vulputatenisi.net	103
EEH05OWI7OE	Tyler Peters	18	1-690-499-9205	metus.Aenean@lobortistellusjusto.com	103
HKF50LOJ3FB	Christen Reed	22	209-6106	tempus.mauris@lobortis.net	105
MQK01HDS0FW	Kenneth Coffey	23	1-481-220-8369	pharetra@iaculis.com	103
UBW92XBZ8QN	Fleur Stephens	17	1-244-412-7834	scelerisque.neque@Mauris.net	101
UYK50JZQ9LK	Harriet Buchanan	20	1-281-709-5524	Suspendisse@dolornonummyac.edu	105
YNY17BQD1SN	Tom Riddle	19	499-6411	aliquet.Phasellus@tinciduntorciquis.com	101

PROFESSORS_MX

CVE_PROF	NOMBRE_PROF	EDAD	CUBICULO
MX01	JULIO ULLOA	28	MC-42
MX02	GERARDO PRADO	29	MC-86
MX03	RAMON LASSALLE	30	MC-22
MX04	MARTHA ELIZABETH	34	MC-47
MX05	IVAN GUSTAVO	35	MC-23
MX06	ISMAEL RUGERIO	45	MC-81

PROFESSORS_XA

CVE_PROF	NOMBRE_PROF	EDAD	CUBICULO
XA01	CARLOS ALBERTO	43	XC-100
XA02	LUIS MANUEL	51	XC-98
XA03	CESAR LOPEZ	35	XC-45
XA04	LUZ YADIRA	25	XC-78
XA05	CARLOS LOPEZ	28	XC-56
XA06	LUIS ALBERTO ORTIZ	45	XC-300

ASSIGNED

MAT	PROF
ADM01	XA04
ADM01	XA06
ELE01	MX01
ELE01	XA03
ING01	MX06
ING01	XA03
ING01	XA04
PSI01	XA04
PSI01	XA06
RED01	MX01
RED01	MX02
RED01	MX05
RED01	XA05

SUBJECTS

CVEMAT	NOMBRE_MAT
ADM01	ADMINISTRACION
ALG01	ALGORITMOS
ARE01	ALGEBRA RELACIONAL
BD01	BASES DE DATOS
ELE01	ELECTRONICA
EST01	ESTADISTICA
ING01	INGLES
PSI01	PSICOLOGIA
RED01	REDES
SIS01	SISTEMAS
TEL01	TELEFONIA

RELATIONS

Tables_in_escuela
ASIGNADAS
ESTUDIANTES_INF
MATERIAS
PROFESORES_MX
PROFESORES_XA
SUSPENDIDOS

Figure 13. Possible values for relation pyfos type.

The exercises used in the test are listed below:

1. Select Database “SCHOOL”.
2. Show the contents of the following relations: “PROFESSORS_MX”, “PROFESSORS_XA”, “SUBJECT”,” ASSIGNED”, “STUDENTS_INF”, and observe both its structure and contents.

3. Perform the PROJECTION of attributes “NAME”, “EMAIL_ST” that belong to the relation “STUDENTS_INF”.
4. Perform the UNION of relations “PROFESSORS_MX” and “PROFESSORS_XA”.
5. Get the attributes of the tuples of the “PROFESSORS_MX” relation which have at least one assigned subject, using the SEMI_JOIN operator.

Figure 14 shows the workspace with a correct solution for exercise 5, where six tokens were used to solve it. In Figure 15, one can see specific values for each token: two relation tokens (PROFESSORS_MX, ASSIGNED), two attribute tokens (CVE_PROF, PROF), one relational operator (SEMI_JOIN) and one comparison operator (equal).

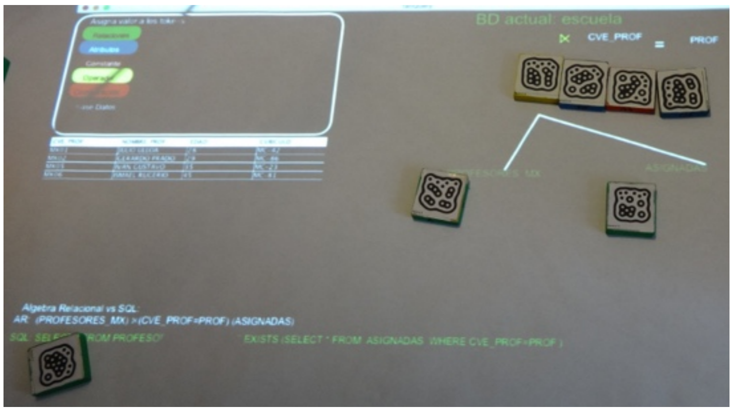

BD actual: escuela
⋉ CVE_PROF = PROF
Algebra Relacional vs SQL:
AR: (PROFESORES_MX) ⋉(CVE_PROF=PROF) (ASIGNADAS)
EXISTS (SELECT * FROM ASIGNADAS WHERE CVE_PROF=PROF)

Figure 14. Sample tree built by a student using a relational operator with a condition.

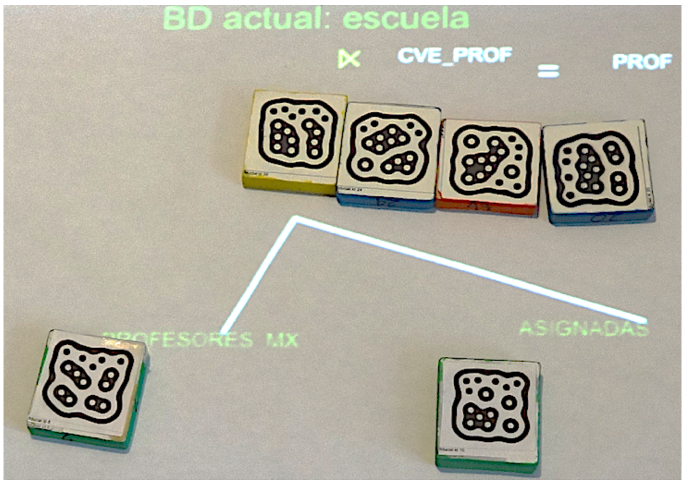
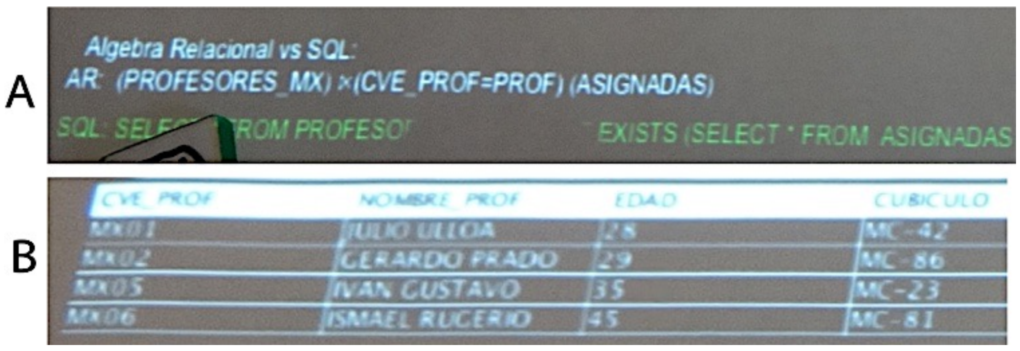


Figure 15. Zooming in of tree build for student.

To construct an RA tree, users do not need to follow specific steps. The important thing is that they are able to identify necessary tokens, assign a value for each token, and execute the appropriate interactions for each case. For example, a possible path to solve the exercise is: (i) the user places a relation-type pyfo within the assignment area, and then assigns it to the value "PROFFESSORS_MX", (ii) the user places an attribute-type pyfo and assigns it the attribute value "CVE_PROF", (iii) the user places a relation-type pyfo and assigns it the value "ASSIGNED", (iv) the user places an attribute-type pyfo and assigns it the value “PROF”, (v) the user uses a relational operator type pyfo and assigns it the value "SEMI_JOIN", and then (vi) assigns the value of "=" to comparison operator pyfo. After all these conditions are completed, the necessary tokens are ready to use.

Now it is necessary to follow the next steps in order to make connections for building the AR tree, for example: (i) place "SEMI_JOIN" token on the work area, (ii) touch the “SEMI-JOIN” on the lower left corner with “PROFFESSORS_MX” token (the system draws a line between both tokens), (iii) touch the lower right corner of “SEMI_JOIN” token with the “ASSIGNED” token, (iv) place the “CVE_PROF” token on the right side of “SEMI_JOIN” token, (v) place the token “=” on the right side of “CVE_PROF”, (vi) place the token "PROF" on the right side of the "=" token. The result is shown in Figure 15.

Figure 16 illustrates the results obtained from the RA tree construction. Figure 16(A) corresponds to the RA and SQL statements, so the user can see differences and equivalences between them. Figure 16(B) corresponds to the resulting data set, which is retrieved from a database server after the SQL query is executed.


A

```
Algebra Relacional vs SQL:
AR: (PROFESORES_MX) ⋉(CVE_PROF=PROF) (ASIGNADAS)
SQL: SEL[illegible]ROM PROFESO[illegible]  EXISTS (SELECT * FROM ASIGNADAS
```

B

CVE_PROF	NOMBRE_PROF	EDAD	CUBICULO
MX01	JULIO ULLOA	28	MC-42
MX02	GERARDO PRADO	29	MC-86
MX05	IVAN GUSTAVO	35	MC-23
MX06	ISMAEL RUGERIO	45	MC-81

Figure 16. A) AR and SQL query. B) Data result set.

6. Results

As can be seen in Table 1, the exercises that required most of the time to be solved by the students, were 3 and 5. The exercise that took less time was the first one, while exercises number 2 and 4 were solved in similar times. The above is due to the complexity of exercises: 3 and 5 required to use certain conditions for their solutions, whereas 1 and 2 only required the assignment of values. In the case of exercise 4, it was necessary to connect two relations as child nodes to a parent operator.

Table 1. Time information about exercises.

	Minimum	Maximum	Average	Std. Deviation
Exercise 1	00:00:06	00:00:42	00:00:15	00:00:13
Exercise 2	00:02:00	00:05:00	00:03:45	00:01:28
Exercise 3	00:03:00	00:16:00	00:06:57	00:04:34
Exercise 4	00:01:27	00:05:00	00:02:23	00:01:19
Exercise 5	00:07:17	00:18:00	00:10:42	00:03:56

Figure 17 shows the time spent by students for each one of the exercises. In general terms, it can be seen that students took most of the time to solve exercise number 5. It can also be analyzed that student number 5 required more time for completing exercises 1, 3 and 5. This student's individual time is higher than the time spent by other students, affecting the total time required for participants to solve exercises; however, in exercises 2 and 4, student number 5 performed equally to the rest of the students.

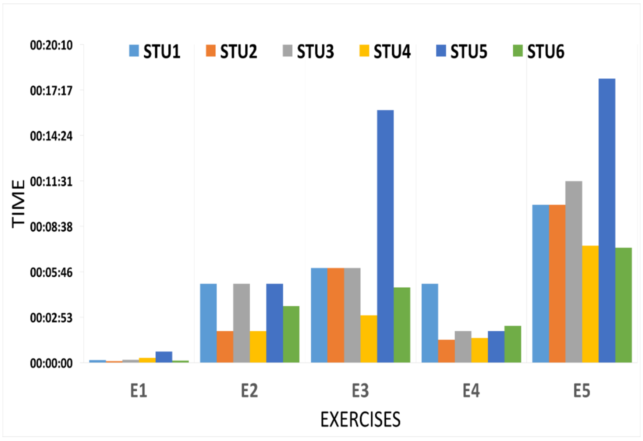


Figure 17. Student time by exercise.

Figure 18 shows the total time spent by student to solve these five exercises. It can be seen that the shortest time to solve all the exercises was the one of student number 4, who spent 14 minutes with 17 seconds, and the longest time was the one of student number 5, who spent 41 minutes 42 seconds.

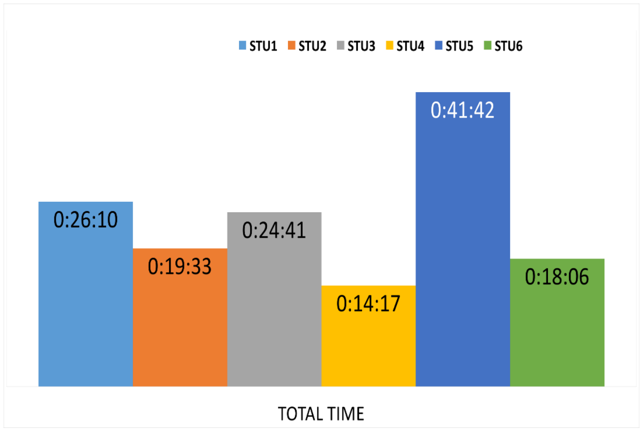


Figure 18. Total time used by the students for the resolution of the exercises.

Apart from times and answers to questions, it was possible to observe certain situations that arose during the sessions. Firstly, the participants had difficulties to use attribute type tokens. Most of them did not read the message that appear in the workspace indicating that they had to put a relation type token in order to obtain the list of possible values for the attribute token. Another circumstance that arose was that they sometimes were confused between tokens of the relation operator type and the comparison operator type. While students were creating a tree, it was difficult for them to identify where to place the tokens that composed the operator condition, so they first performed the type 2 interactions shown in Figure 2.

Finally, some comments of the students were: (i) “Sometimes it is hard to see the information because of the lighting”, (ii) “The quality of the colors need to be clearer, especially for red”, (iii) “The space in the assignment area is reduced”, (iv) “It would be better if the shape of objects was a polygon, so that it would easier to rotate”, (v) “I found the system helpful to learn relational algebra, but I feel that it lacks a manual that specifies instructions on how to use the objects”, (vi) “I would like the system to have an interactive tutorial”, (vii) “it would be useful to have auditory feedback”.

7. Related work

Some tools have been developed to support the teaching-learning process of relational algebra, such as WinRDBI [9], iDFQL [10], RALT [11], DBSnap [12], RAPT (Relational Algebra Parsing Tools) [13], among others. However, these tools have been developed as traditional WIMP systems.

Tangible user interfaces, on [5] a table (not digital), are combined with a bi-dimensional screen. In this combination the user makes queries placing or removing the tokens on the table while the screen shows the result of the query made by the user. Although the used database makes reference to demographic data of radio station listeners, the system can also query other databases. The screen only shows results in graphic form, while the source queries cannot be viewed. Sources’ particular role of collaboration can be a differentiating factor between tangible interfaces and WIMP. Thus, TanQuery queries and their results are shown to allow the user to see different ways to do the same thing.

Another work is presented by Ullmer, Ishii and Jacob [6]. They propose two systems of tangible interfaces to execute queries: “parameter wheels” and “parameter bars”. They focus on using tokens with physical constraints to express, manipulate and visualize database queries within the parameters. These are based on ideas exposed in the "Dynamic Homefinder" GUI [18], where the concept of "dynamic queries" was introduced. This concept allows the user to make queries by adjusting graphical elements, like sliders, and observing the result immediately. They make use of a realty database to locate homes that met specific search criteria. The difference with TanQuery is that queries have a predefined structure and the position of tokens cannot be modified (the placement of the tokens only allows their values to be modified). This situation limits the freedom of manipulation by the user on the workspace. Still, this work could be useful for teaching how to use information filters.

CubeQuery is other tangible system that possess the ability to display a physical form, not only for creating databases, but also to modify queries. Proposed by Langer [7], CubeQuery uses active tangibles (tangible equipped with some kind of electronics, for example a display, sensors or wireless communication within them) to perform searching tasks within a music information database. To input data into the system, the user can combine the manipulation of tokens with gestures that are used to assign a value to each token. This work, as shown in [5], does not display the queries that were used to generate the information. TanQuery do not use any sensors within the pyfos, so any object with same dimensions can be used only when labeling it with a fiducial marker, for this extend, decreasing the cost.

In short, the difference between TanQuery and other projects above mentioned is that they do not consider topics of relational algebra. Instead they focus on different goals, such as collaboration issues or exploration of the spatial arrangement practicality of tangibles. At least for RA teaching, these projects focus on fields other than education.

8. Conclusions and future work

This paper proposed TanQuery as a system that can support the teaching of relational algebra. Using TAC paradigms terms, this system can explain relational algebra's functional architecture and describe its elements. Moreover, the objective of this paper was to present an interpretation for the interactions that can be made between tokens and elements used for tokens' development (hardware and software). The paper also intended to present some of the problems identified during the test, as well as some recommendations to improve the system. It is important to note that a recurrent recommendation done by participants was (1) the integration of an interactive tutorial so to explain the system interactions, (2) the inclusion of menus with a list of all possible values for tokens, and (3) the need to include auditory feedback.

As future work, we plan to improve the system with actuation tokens, so to enhance the feedback provided to the user, and to study broken links between the digital data and the physical objects; for example, when users perform and undo an action.

Acknowledgments

This research was supported by3 Consejo Nacional de Ciencia y Tecnología (CONACYT Mexico), thanks to a Ph. D. scholarship (Ref. 326140, Support No. 449513) and the Project "Infraestructura para Agilizar el Desarrollo de Sistemas Centrados en el Usuario" (Ref. 3053).

References

[1] H. Ishii, "The tangible user interface and its evolution," *Commun. ACM*, vol. 51, no. 6, p. 32, Jun. 2008.

[2] M. Green and R. Jacob, "Software architectures and metaphors for non-WIMP user interfaces," *ACM SIGGRAPH Comput. Graph.*, vol. 25, no. 3, pp. 229–235, 1991.

[3] O. Shaer, "Tangible User Interfaces: Past, Present, and Future Directions," *Found. Trends® Human–Computer Interact.*, vol. 3, no. 1–2, pp. 1–137, 2009.

[4] P. Marshall, "Do tangible interfaces enhance learning?," *Proc. 1st Int. Conf. Tangible Embed. Interact. - TEI '07*, p. 163, 2007.

[5] A. Jofre, S. Szigeti, S. T. Keller, D. Czarnowski, F. Tomé, and S. Diamond, "A Tangible User Interface for Interactive Data Visualization," 2015.

[6] B. Ullmer, H. Ishii, and R. J. K. Jacob, "Tangible Query Interfaces: Physically Constrained Tokens for Manipulating Database Queries," *Science (80-.).*, vol. 3, no. c, pp. 279–286, 2003.

[7] R. Langner, "CubeQuery: Tangible Interface for Creating and Manipulating Database Queries," pp. 423–426, 2014.

[8] D. C. Costa, "El modelo relacional y el álgebra relacional," *Barcelona, Eureca Media SL*, p. 58, 2005.

[9] S. W. Dietrich, E. Eckert, and K. Piscator, "WinRDBI: A Windows-based Relational Database Educational Tool," *SIGCSE Bull.*, vol. 29, no. 1, pp. 126–130, 1997.

[10] A. P. Appel, E. Q. Silva, C. Traina Jr, and A. J. M. Traina, "iDFQL-A query-based tool to help the teaching process of the relational algebra," in *World Congress on Engineering and Technology Education. WCETE*, 2004.

[11] P. Mitra and F. Sadri, "Relational algebra learning tool," *Imp. Coll. London*, 2009.

[12] Y. N. Silva and J. Chon, "DBSnap: Learning Database Queries by Snapping Blocks," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 2015, pp. 179–184.

[13] O. Karpova, N. D. Souza, D. Horton, and A. Petersen, "RAPT: Relational Algebra Parsing Tools," p. 2015, 2015.

[14] O. Shaer, N. Leland, E. Calvillo-Gamez, and R. K. Jacob, "The TAC paradigm: specifying tangible user interfaces," *Pers. Ubiquitous Comput.*, vol. 8, no. 5, pp. 359–369, Jul. 2004.

[15] P. Omar and S. Vásquez, "Infraestructura para el Desarrollo de Interfaces de Usuario Tangibles en Escritorios Inteligentes ," 2017.

[16] M. Kaltenbrunner and R. Bencina, “reacTIVision: a computer-vision framework for table-based tangible interaction,” *Proc. 1st Int. Conf. Tangible Embed. Interact.*, pp. 69–74, 2007.
[17] M. Kaltenbrunner, T. Bovermann, R. Bencina, and E. Costanza, “TUIO: A protocol for table-top tangible user interfaces,” *Neuroinformatics*, no. May 2005, pp. 1–5, 2005.
[18] B. Shneiderman, “The Dynamic HomeFinder: Evaluating Dynamic Queries in a Real-Estate Information Exploration,” 1992.

About the authors

MSICU. Antonio Xohua

Graduated from the Computer Science undergraduate program at Universidad Veracruzana, he obtained a master’s degree in Interactive Systems with a focus on user from the Universidad Veracruzana. Professor at the Faculty of Statistics and Informatics of Universidad Veracruzana.

Dr. Edgard Iván Benítez Guerrero

Computer Systems engineer graduated from Universidad de las Américas Puebla and Master in Artificial Intelligence from Universidad Veracruzana. He holds a PhD program in Computer Science from Joseph Fourier University (University of Grenoble I, Grenoble, France). Full Time Professor (Titular "C") at the Faculty of Statistics and Informatics of Universidad Veracruzana. His areas of interest range from databases, artificial intelligence, and human computer interaction to mobile and ubiquitous computing, and context aware computing.

Dra. María Del Carmen Mezura Godoy

She holds a degree in Computer Science from Instituto Tecnológico de Tijuana-ITT. She also finished her Master program in Artificial Intelligence from Universidad Veracruzana and her PhD program in Computer Science that specialized in Groupware from University of Savoie in France. Full time Professor (Titular "C"). Her areas of interest include CSCW, IHC, context-aware computing, e-learning and multiagent systems.