CUDL Language Semantics: Conditions

Nikitas N. Karanikolas
Department of Informatics
Technological Educational Institution (TEI) of Athens
Ag. Spyridonos street
12210 Aigaleo
Greece
nnk@teiath.gr

Journal of Digital
Information Management

ABSTRACT: *Conceptual Universal Database Language (CUDL) is a new language designed to manage dynamic database environments, which conform to the Frame DataBase model (FDB). FDB is a generic database model (oversubscribe both the Entity-Attribute-Value and the Nested Relational). CUDL is not only an FDB database language but it is mainly an agent that provides an abstraction level superior to the logical level. CUDL permits the users to conceive a database schema where single fields (tags) accept repetitions (list of values), entertain subfields and also permit to entertain an entire table in the place of a single field. In this paper we investigate the conditions, used in both retrieval and update CUDL statements, and focalize especially on the range of affection of the update statements. This becomes necessary since the CUDL abstraction level reveals the differentiation between frame objects, tag repetitions and subfield repetitions and consequently the update statements should clearly define where the data modifications occur. This need becomes heavier where some frame object verifies two (or more) disjunctive conditions and these conditions specify different range of affection inside the frame. We provide two solutions where the second one is an improvement of the first one but it demands more refined statements and more instructed / skilled users. Some constructive discussion is also provided.*

Categories and Subject Descriptors

H.2.3 [Languages]; H.2.1 [Logical Design]: Data models

General Terms: Interoperability framework, E Government, European Union E Govt. Applications

Keywords: Dynamically evolving database environments, database languages, conditions

Received 7 November 2008, Revised 19 January 2009, Accepted 31 January 2009

1. Introduction

In previous work [19], [20] there has been an investigation of dynamically evolving database environments and corresponding schemata [1], [2], [3], [4], [5], [6], [12], [13], [14], [15], [16], [17], [18], allowing storage and manipulation of variable number of fields per record, variable length of fields, data subfields, multiple value fields, etc. The ultimate goal of Yannakoudakis et al [19], [20] was to make the design and maintenance of a database a much simpler task for database designers, so as that they would not have to put in a lot of effort to design the database and later they would not have to pay extra special attention and work on database changes. This has resulted in a new framework for the definition of a universal database schema that eliminates completely the need for reorganisation at both logical and internal levels, even when the slightest modification in the database requirements must occur. This new framework was called FDB [19].

The management however and operation of this model (framework) is laborious and time-consuming as the user would have to put in a lot of strain to understand and be familiar with the use of the proposed model, meaning the structures and organisation of it (metadata and data), as well as the processes of the management of elements that compose it. For this reason we focused our research in finding an efficient and easy way for the communication of users with the model [7], [8], [9], [21]. This has resulted in creating a language which can bridge the gap between the user and the FDB model, in other words can help the user to manipulate the applications that have been created based on the proposed model [7], [8], [9], [21], [22]. This language was called CUDL [21]. By the use of CUDL, which encapsulates methods with data structures, an FDB management system can execute complex meta-data and data manipulation operations to retrieve and transform information. FDB developers can write complete database applications with the modest amount of effort [7], [8], [9], [21], [22].

In [7] we introduced the syntax and semantics of the CUDL language. There we focused mainly in presenting and analysing the statement of value retrieval (in the schema and the data). In [9] we focused mainly in presenting and analysing the syntax and semantics of the CUDL statements used for value modification (in the schema and the data). The efficient building of applications with the CUDL usage has been also shown [8].

1.1 Motivation

CUDL permits the users to conceive a database schema as having composite data structures. In CUDL a single field (tag) can accept repetitions (list of values), entertain subfields and also permit to entertain an entire table in the place of a single field. In general, CUDL statements are composed by: some *action* (retrieval, update, removal, insertion), the *entity specification,* the *field of application* (which attributes (tags) are affected) and a *range of application* (which frame objects (instances of the entity) are affected). It is obvious that the range of application part of a CUDL statement is composed by a combination of conditions over tags and/or subfields. As a consequence of the composite data structures of the frame objects, the CUDL language should be equipped with control handles that will allow users to define precisely where and how the conditions in the range of application part of a CUDL statement are combined. This is a novel need since the most proliferated database model (the relational) provides simple data structures (tuples) and the conditions in the range of application part (Where part) of its language (SQL) should be satisfied simply inside the instances (tuples). In CUDL, on the contrary, if a range of application part contains conditions for two subfields of the same tag, the user should clarify whether these conditions should be satisfied in the same tag repetition (of the same frame object) or can be satisfied in different tag repetitions (of the same frame object). Some similar necessitation also arises for the field of application (which attributes (tags) are affected). The later necessitation has influence only on update, removal and insertion CUDL statements and not on retrieval statements.

2. Background

2.1 FDB

FDB is a generic database schema, with a specific unique form, where we use only one database schema for every application. Whatever the entities needed for the application, no new tables are introduced. We merely define virtual tables corresponding to the needed entities having virtual fields (tags) and subfields. The FDB model schema has the form shown in Table 1 (note that primary keys are underlined):

Languages	(language_id, lang_name)
Datatypes	(datatype_id, datatype_name)
Messages	(message_id, language, message)
Entities	(frame_entity_id, title)
Tag_attributes	(entity, tag, title, occurrence, repetition, authority, language, datatype, length)
Subfield_attributes	(entity, tag, subfield, title, occurrence, repetition, language, datatype, length)
Catalogue	(entity, frame_object_number, frame_object_label, temp_stamp)
Tag_data	(entity, frame_object, tag, repetition, chunk, tdata)
Authority_links	(from_entity, from_tag, from_subfield, to_entity, to_tag, to_subfield, relationship_type)
Subfield_data	(entity, frame_object, tag, tag_ repetion, subfield, subfld_repetition, chunk, sdata)

Table 1. The FDB model schema (universal schema)

The terminology used in FDB is the following:

Entity: a collection of related objects that have certain attributes

Tag: an object used to represent some attribute of an entity

Subfield: an object associated to a specific tag, used to represent a sub-attribute of the tag that belongs to a certain entity

Compo tag: a composite tag hosting subfields

Simple tag: a tag that does not host subfields

Frame object: A specific instance of an entity, meaning an instance holding data that describe a certain case (view of the world) that belongs to the entity. It uniquely identifies this case inside the entity. (We could say that it is analogous to the tuple in the relational model).

We will display a simple example that concerns a very simple database schema that manages the projects of a company. It is based in a single entity (Projects) which is implemented with an FDB entity having four attributes (tags). The tags Project_code, Title, and Budget are simple tags with single values (without repetitions). It is evident from the names of tags what their contents are. The last tag is the Actions tag which is a compo tag with three subfields (Employee, Action and Deadline) and permits multiple values (repetitions). Consequently, by having subfields and also repetitions, the tag Actions is a case of a table in the place of a field. Moreover, two subfields (namely the Action and the Deadline subfields) of the Actions tag accept only single values and another subfield (namely the Employee subfield) permit multiple values (one or more employees can be charged with the same Action). The FDB application schemata are self-explained, because all the above information is declared in four real FDB tables (sets), the entities, the tag_attributes, the subfield_attributes and the Messages sets. The information, mentioned above, for the Projects application, is defined in the previously stated four real FDB tables. The contents of the later three tables, for the Projects application, are shown in Tables 2a, 2b and 2c.

Entity	Tag	Title	Occurrence	Repetition	Authority	Language	Datatype	Length
1	1	2	M	N	N	1	1	7
1	2	3	M	N	N	1	1	50
1	3	4	M	N	N	1	3	NULL
1	4	5	M	R	N	1	5	NULL

Table 2a. Content of the Tag_attributes for the company's projects application

Entity	Tag	Subfield	Title	Occurrence	Repetition	Language	Datatype	Length
1	4	1	6	M	R	1	1	20
1	4	2	7	M	N	1	1	30
1	4	3	8	M	N	1	4	NULL

Table 2b. Content of the Subfield_attributes for the company's projects application

Message_id	Language	Message
1	1	Projects
2	1	Project_code
3	1	Title
4	1	Budget
5	1	Actions
6	1	Employee
7	1	Action
8	1	Deadline

Table 2c. Content of the Messages for the company's projects application

All the data of the Projects application are stored in barely two real FDB schema tables (namely tag_data and subfield_data). We need no new tables to store each entity's individual data.

2.2 CUDL

CUDL was designed to manage dynamic databases (schema evolution databases) and also Generic database schemata. The language provides the users a higher abstraction level than the logical abstraction level and can be exploited by simple Generic database schemata (like the Entity Attribute Value – EAV) and more sophisticated ones (like the Frame DataBase – FDB). However, it is mainly designed to exploit all the structures of the FDB model with convenience and effectiveness. Without CUDL, the management and operation of Generic database schemata (like FDB) would require from the user (administrator, developer) a very good acquaintance of the proposed model, the structures and organisation of it as well as the processes of the management of elements that compose it. Otherwise, it would be very difficult and sometimes impossible to carry out even simple operations like the simplest retrieval of information. With the abstraction level that CUDL introduces, users keep away from difficult programming tasks that the Generic database schemata impose. For example, there is a need for many joins (self-joins) or many queries (or views) and then intersections of the results, in order to retrieve and project data. The CUDL abstraction level removes any such difficulty and let the users conceive the data with more flexible structures than the simple fields of the relational model. The users conceive the data attributes as lists of values, composite values with subfields, etc. Together with this higher perception of data CUDL language preserves the schema evolution characteristic of Generic database schemata. Now we shall portray one frame for the entity Projects, in the way the user apprehends it (CUDL abstraction level). The frame shown in Table 3 corresponds to an instance of the entity Projects defined in Tables 2a, 2b and 2c.

Project_code	Proj077
Title	Zeus
Budget	317,000
Actions	

Employee	**Action**	**Deadline**
Kostas	Software analysis	17/10/2007
Petros Stella	Software requirements	22/01/2008
Giorgos Manolis	Program code	23/04/2008

Table 3. A Projects frame object

In previous works [7], [8] we have described the syntax and semantics for CUDL data definition and data retrieval statements. However a single data retrieval statement could be helpful for the reader to understand how the CUDL abstraction level is materialized through statements. The following CUDL data retrieval statement is used in order to search for Projects with budget greater than 300,000 and having actions starting with the word Software:

# Find data when entity = ‘Projects’ and tag = ‘Budget’ restr data > ‘300,000’ and subfield = ‘Action’ restr data like ‘Software%’ and subfield = ‘Employee’ and tag = ‘Title’

This statement also projects the subfield Employee and the tag Title of the frame objects verifying the restrictions. Table 4 portrays the output of this statement.

Entity	**Frame_ object**	**Tag**	**Tag_ Repetition**	**Subfield**	**Subfld_ Repetition**	**Data**
Projects	1	Budget	1	-	-	317,000
		Actions	1	Action	1	Software analysis
				Employee	1	Kostas
			2	Action	1	Software requirements
				Employee	1	Petros
					2	Stella
		Title	1	-	-	Zeus

Table 4. Results of a CUDL data retrieval statement

Since, the problem analysis and the solutions introduced in the following sections are based on the semantic forms of the CUDL update data statements, we will give a brief introduction of them. All *update data* statements (‘Alter data’, ‘Delete data’ and ‘Insert data’) have similar semantic forms and it is satisfactory to provide the semantic forms for one of them. The ‘Alter data’ statement has the following five semantic forms:

(1) Alter data set <tag> = <new value> <entity specification condition> <frame specification conditions>

(2) Alter data set <tag> = <new value> <entity specification condition> <frame & repetition of affected tag specification conditions>

(3) Alter data set <subfield> = <new value> <entity specification condition> <frame specification conditions>

(4) Alter data set <subfield> = <new value> <entity specification condition> <frame & repetition of affected tag specification conditions>

(5) Alter data set <subfield> = <new value> <entity specification condition> <frame & repetition of affected tag & repetition of affected subfield specification conditions>

For the first (1st) semantic form we can provide the following indicative example:

# Alter data set ‘Title’ = ‘Hercules’ when entity = ‘Projects’ and tag = ‘Project_code’ restr data = ‘Proj095’

This statement sets the title to ‘Hercules’ for the project having Project_code equal to ‘Proj095’.

Behind the FDB data modelling and the CUDL language there is our suggestion for moving in a higher level the database design process [10]. We claim that the Information System design should not decompose the real world (that we were called to impress in an Information System) in its fundamental characteristics and afterwards to proceed with simple compositions of characteristics that relational model allows. We claim another approach where the Information System designers would be able to portray directly the real world in a model that provides more powerful structures, as those of the real world. Therefore, there is a necessity for a database query and manipulation language able to manipulate directly the composite (real world) data types. The Conceptual Universal Database Language (CUDL) was designed to satisfy the mentioned necessitation. We have adopted the Frame Database Model as the underlying model for implementing our goal for a data manipulation language able to manipulate directly composite data types. We preferred the FDB model, since it is more compact and well defined than other models (that offers composite data types) and also supports schema evolution [9]. Some comparison between FDB / CUDL and other formalisms (either theoretical or industrial) can be found in [10].

Project_code	Proj001
Title	Hercules
Budget	200,000
Actions	

Employee	**Action**	**Deadline**
Giorgos	grubbing	28/2/2007
Kostas	watering	30/3/2007
Manolis	pruning	15/4/2007

(1st)

Project_code	Proj002
Title	Afrodite
Budget	300,000
Actions	

Employee	**Action**	**Deadline**
Manolis	grubbing	30/3/2007
Petros	watering	20/4/2007
Giorgos	pruning	10/5/2007

(2nd)

Project_code	Proj003
Title	Hermes
Budget	250,000
Actions	

Employee	**Action**	**Deadline**
Manolis	grubbing	20/3/2007
Petros	watering	30/3/2007

(3rd)

Table 5. Three frame objects of the Projects entity

3. The Problem and its Solution

3.1 Co-verification of Conditions

Since the CUDL language reveals the differentiation between frame objects, tag repetitions and subfield repetitions, the conditions (in the range of application part of statements) should clearly define where the conditions are co-verified. Otherwise, the statements are counterintuitive if not ambiguous. In order to make the problem more easily perceptible we will attempt its presentation through examples. First we will examine a complex condition that is constituted by two other simple conditions which declare restrictions that are applied in two subfields that emanate from (belong to) the same tag:

# Alter data set 'Employee' = 'Petros' when entity = 'Projects' and subfield = 'Action' restr data = 'pruning' and subfield = 'Employee' restr data = 'Manolis'

# Find data when entity = 'Projects' and subfield = 'Action' restr data = 'pruning' and subfield = 'Employee' restr data = 'Manolis'

In both statements we have the same complex condition (subfield = 'Action' restr data = 'pruning' and subfield = 'Employee' restr data = 'Manolis') and regardless of the action of the statement (simple retrieval and projection of information in the case of the Find statement or change of data in the case of the Alter statement) we need the complex condition to determine the frame objects where the action of the statement will take place. This complex condition contains two simple conditions; one of them concerning the subfield 'Action', the other concerning the subfield 'Employee' and both subfields belong to the tag 'Actions' which accepts repetitions. Here a question arises about whether the values indicated by the conditions should coexist in the same repetition of the tag 'Actions' or it's sufficient enough that they exist even in different repetitions of the tag 'Actions', so as the frame object containing them can be considered as validating the condition.

If we accept the first point of view then only the first of the frame objects of table 5 verifies the complex condition. On the contrary, if we accept the second point of view then the first two frame objects of table 5 verify the complex condition.

Many readers may think that only the first point of view is right and that the question we have introduced is excessive examination. This thought can be brought to a lot of readers of this article, precisely because they have common sense. But we can not exclude that some (even few) when expressing a statement containing the under consideration condition (subfield = 'Action' restr data = 'pruning' and subfield = 'Employee' restr data = 'Manolis') would want the result, where the statement will take action, to include both the first two mentioned above frame objects.

In the following we will examine a complex condition containing restrictions for tags as well as for subfields. Indicatively, let us presume that the user expresses the following statement:

# Find data when entity = 'Projects' and (subfield = 'Employee' restr data = 'Manolis' or tag = 'Budget' restr data = '200,000') and subfield = 'Deadline' restr data = '30/3/2007'

This statement, based on the Boolean algebra, can be transcribed in:

# Find data when entity = 'Projects' and (subfield = 'Employee' restr data = 'Manolis' and subfield = 'Deadline' restr data = '30/3/2007') or (tag = 'Budget' restr data = '200,000' and subfield = 'Deadline' restr data = '30/3/2007')

In this example it is less clear, than the previous couple of statements (Alter and Find, in the beginning of the current section), whether the user wants the data 'Manolis' and '30/3/2007' to coexist in the same repetition of the tag 'Actions' or simply to coexist in the same frame object. This doubt (*ambiguity*) comes from the fact that the user, when writing the statement (before our transcription), kept the subfields 'Employee' and 'Deadline' in distance.

In general we can say that because frame objects are more complex objects than the tuples in the relational model, statements should permit the user to determine the amplitude where the co-verification of restrictions should be examined. We could provide such a possibility with an extension of the CUDL language which would allow the use of indicators. For example in the next two (imaginary) statements the user uses a common indicator (I) when he wishes the data 'Manolis' and '30/3/2007' to coexist in the same repetition of the tag 'Actions' and he uses different indicators (I and J) when he wishes the data 'Manolis' and '30/3/2007' to simply coexist in the frame object.

# Find data when entity = 'Projects' and (subfield$_I$ = 'Employee' restr data = 'Manolis' or tag = 'Budget' restr data = '200,000') and subfield$_I$ = 'Deadline' restr data = '30/3/2007'

# Find data when entity = 'Projects' and (subfield$_I$ = 'Employee' restr data = 'Manolis' or tag = 'Budget' restr data = '200,000') and subfield$_J$ = 'Deadline' restr data = '30/3/2007'

3.2 Affected Structures

Another problem, raised from the compound form of frame objects, is the ambiguity for the field of application (which tags / subfields are affected) of the update data statements ('Alter data', 'Delete data' and 'Insert data'). For the better understanding of the problem we will examine four statements expressed in the CUDL language, in combination with the facts of table 5:

# Alter data set 'Employee' = 'Petros' when entity = 'Projects' and tag = 'Project_code' restr data = 'Proj003'

# Alter data set 'Employee' = 'Petros' when entity = 'Projects' and subfield = 'Action' restr data = 'pruning'

# Alter data set 'Employee' = 'Petros' when entity = 'Projects' and subfield = 'Action' restr data = 'pruning' or tag = 'Project_code' restr data = 'Proj003'

# Alter data set 'Employee' = 'Petros' when entity = 'Projects' and subfield = 'Action' restr data = 'grubbing' or tag = 'Project_code' restr data = 'Proj003'

The first statement contains only one condition which is verified by only one frame object of the table's 5 data and more precisely the last one. In this case we have a statement of the third semantic form of alter (see section 2.2) which will result in the replacement of the employees that take part in any repetition of the tag 'Actions' in the frame objects verifying the condition. Based on our example the modified form of the last frame object (of table 5) is portrayed in table 6.

Project_code	Proj003
Title	Hermes
Budget	250,000

Actions

Employee	**Action**	**Deadline**
Petros	grubbing	20/3/2007
Petros	watering	30/3/2007

Table 6. The 3rd (last) Projects frame object after its modification

Project_code	Proj001
Title	Hercules
Budget	200,000

Actions

Employee	Action	Deadline
Giorgos	grubbing	28/2/2007
Kostas	watering	30/3/2007
Petros	pruning	15/4/2007

(1st)

Project_code	Proj002
Title	Afrodite
Budget	300,000

Actions

Employee	Action	Deadline
Manolis	grubbing	30/3/2007
Petros	watering	20/4/2007
Petros	pruning	10/5/2007

(2nd)

Table 7. The 1st and 2nd Projects frame objects after their modification

The second of the previously mentioned statements contains also only one condition which is verified in two frame objects of the table's 5 data, and more precisely the first and the second frame object. In this case we have a statement of the fourth semantic form of alter which will result in the replacement of the employee of a certain repetition of the tag 'Actions' in the frame objects verifying the condition. Based on our example data (of table 5) the first two frame objects are modified. Their modified forms are portrayed in table 7.

Now we can examine the third statement containing one complex condition which is a disjunction of the simple conditions of the two previous statements. This statement is a combination of the third as well as the fourth semantic form of alter as the 1st and 2nd frame objects of table 5 verifies the first sub condition and determines specific repetitions of the tag 'actions' and the 3rd frame object of table 5 verifies the second sub condition and determines the whole frame object. In this case the update in the two frame objects verifying the first sub condition will take place only in the specified repetitions of the tag 'actions' and in the frame object verifying the second sub condition the changing will take place in every repetition of the tag 'actions'.

So far there is no problem. The problem occurs when a CUDL statement contains two disjunctive sub conditions leading in different semantic forms and there are frame objects verifying both sub conditions. The last (fourth) statement of the previously mentioned statements causes such a problem. More precisely the first sub condition determines a certain repetition (row) of the tag 'Actions' for each one of the three frame objects (of table 5) and the second sub condition determines the whole 3rd frame object. Here, we face the problem on the way of handling the third frame object which verifies at the same time both sub conditions, therefore causing a dilemma whether it will be dealt as a fourth semantic form (and the change will take place in the first repetition of the tag 'Actions') or it will be dealt as a third semantic form (and the change will take place in every repetition of the tag 'Actions').

In this dilemma (whether it will be dealt as a fourth or a third semantic form) and in every similar dilemma (choice between the fourth and fifth, third and fifth semantic form, etc) a solution can be easily given by the definition of the rule "we handle every frame object based on the biggest semantic form determined by the disjunctive conditions which verify it" or alternatively by the definition of the rule "we handle every frame object based on the smallest semantic form determined by the disjunctive conditions which verify it". Therefore, the problem is simplified in a transcription of the original complex condition of the CUDL statement in "disjunctive primitive subqueries". Here the adjective "primitive" is used to indicate that the sub conditions do not contain by themselves any more disjunctions. Having determined the "disjunctive primitive subqueries" (DPSs) of the original complex condition we can easily find which DPSs are verified by any specific frame object that verifies the whole condition. Therefore we are in a position to know which semantic form (or forms) apply in each frame object. Afterwards by applying the defined rule we can go on changing the data. For example, let us assume that we have the next CUDL statement:

# Alter data set 'Deadline' = '30/4/2007' when entity = 'Projects' and (subfield = 'Employee' restr data = 'Manolis' or tag = 'Budget' restr data = '200,000') and tag = 'Title' restr data like 'K%'

This statement can be transcribed in:

# Alter data set 'Deadline' = '30/4/2007' when entity = 'Projects' and (subfield = 'Employee' restr data = 'Manolis' and tag = 'Title' restr data like 'K%') or (tag = 'Budget' restr data = '200,000' and tag = 'Title' restr data like 'K%')

In this form we have two disjunctive primitive subqueries. The first one (subfield = 'Employee' restr data = 'Manolis' and tag = 'Title' restr data like 'K%') determines the fourth semantic form of alter, whereas the second (tag = 'Budget' restr data = '200,000' and tag = 'Title' restr data like 'K%') determines the third semantic form. If there exists one frame object verifying both the two disjunctive primitive subqueries and the manipulation is done based on the bigger semantic form then the change will take place only in the repetition of the tag 'Actions' having in its subfield 'Employee' the value 'Manolis'. On the contrary, if the manipulation is done based on the smaller semantic form then the change will take place in every repetition of the tag 'Actions' of the frame object verifying both disjunctive primitive subqueries.

3.3 Naive Solution

The needs to determine where the conditions are co-verified as well as to transcript the original condition in disjunctive primitive subqueries are raised very often. The next example of a statement in the CUDL language shows both needs:

# Alter data set 'Employee' = 'Petros' when entity = 'Projects' and (subfield = 'Employee' restr data = 'Manolis' or tag = 'Budget' restr data = '200,000') and subfield = 'Deadline' restr data = '30/3/2007'

In this statement it is important the existence of a mechanism to transcript the original condition in disjunctive primitive subqueries by the system as well as the existence of a mechanism which will allow the user to indicate whether the values 'Manolis' and '30/3/2007' should coexist in the same repetition of the tag 'Actions' or it is sufficient enough to exist even in different repetitions of the tag 'Actions'.

The simplest and therefore the most limited in possibilities solution is presented afterwards. This solution can be sufficient enough in a practical level and moreover to constitute a metre of comparison for other following extensions of the CUDL language trying to meet in a better way the need of defining where the conditions are co-verified as well as the transcription of the original conditions in "disjunctive primitive subqueries". According to this solution the CUDL language does not allow the use of parentheses and the only allowed Boolean operators are the conjunction (and) and the disjunction (or) operators. Moreover the *or* operator has smaller precedence than the operator *and*. Therefore a condition of the form "x and y or z and w" is dealt as if it was "(x and y) or (z and w)". As an example we can provide the next statement:

# Find data when entity = 'Projects' and tag = 'Title' restr data like 'A%' and tag = 'Budget' restr data > '200,000' or tag = 'Project_code' restr data like 'Proj??1' and tag = 'Title' restr data like 'H%'

and expect to be processed by the system as if it was:

# Find data when entity = 'Projects' and (tag = 'Title' restr data like 'A%' and tag = 'Budget' restr data > '200,000') or (tag = 'Project_code' restr data like 'Proj??1' and tag = 'Title' restr data like 'H%')

Practically, the user expresses only conditions (following the entity specification condition) of the form:

x_1 and x_2 and … and x_p or y_1 and y_2 and … y_q or … or w_1 and w_2 and … and w_r

and expects them to be processed by the system as if it was:

(x_1 and x_2 and … and x_p) or (y_1 and y_2 and … y_q) or … or (w_1 and w_2 and … and w_r)

What we really achieve with the previously mentioned syntactical restrictions is to enforce the user to write statements by using directly "disjunctive primitive subqueries" and therefore not to need any transcription for these statements from the system.

As far as the problem of where the conditions are co-verified is concerned, we define that in each one of the disjunctive primitive subqueries, its partial conditions are dependent on one another, that is to say that they have to be verified in the smallest structure of data. On the contrary the application (the place of verification) of conditions of one disjunctive primitive subquery is independent from the application of conditions of any other disjunctive primitive subquery (DPS). To make these clear, we examine the next condition:

tag = 'Budget' restr data > '240,000' and subfield = 'Employee' restr data = 'Giorgos' **or** subfield = 'Employee' restr data = 'Manolis' and subfield = 'Action' restr data = 'pruning' **or** subfield = 'Employee' restr data = 'Giorgos' and subfield = 'Action' restr data = 'grubbing'

For the first DPS the smallest structure of data that can accommodate the tag 'Budget' and the subfield 'Employee' is the whole frame object. In our example data (of table 5), the second frame object verifies this (first) DPS. For the second and third DPS the smallest structure of data that can accommodate the subfield 'Employee' and the subfield 'Action' is a repetition of tag 'Actions'. In our example data, the third repetition of tag 'Actions' of the first frame object verifies the second DPS and the first repetition of tag 'Actions' of the first frame object verifies the third DPS.

4. An Advanced Solution

The naive version (solution) of CUDL is an easily understandable confrontation with satisfactorily expressive power. However, more expressive versions can be designed permitting more refined statements, possibly with overhead in the syntax of statements and with requirements for more instructed (skilled) users. Of course the naive CUDL language version is not a toy database language and can be adopted as the dominant database language for database management systems adopting and afford the CUDL abstraction level. In this section we will provide a more expressive than the naive CUDL language version as a first step to open the researh for more refined languages for handling composite data. We will name this advanced CUDL language as "CUDL naive + indices" and shortly "CUDL npi". "CUDL npi" offers to the user, more than what the Naive CUDL version offers, control handles that will allow them to define precisely where and how the conditions of a CUDL statement are co-verified. The handles are optional indices that the user can apply on subfields (only). The form of indices is single Latin letters in square brackets and can follow immediately after the reserved word "subfield". The usage of the same index in two conditions in the same disjunctive primitive subquery entails that the requested restrictions must be verified in the same repetition of the tag hosting the coindexed subfields. Obviously, the usage of the same index (in two conditions in the same disjunctive primitive subquery) for subfields that don't belong in the same tag is a mistake and should be handled accordingly. In order to make clear the usage of indices we will provide indicative examples and will discuss them in combination with their semantic forms. The data against which the examples are examined are presented in tables 3 and 5.

# Alter data set Employee = 'Nikitas' when entity = 'Projects' and subfield[i] = 'Employee' restr data = 'Petros' and subfield[i] = 'Deadline' restr data = '22/01/2008'

In this statement we express the requirement that the employee assigned some action to be named 'Petros' and the deadline for the same action to be '22/01/2008'. In other words we require that both values 'Petros' and '22/01/2008' to coexist in the same repetition of tag 'Actions'. Moreover, since the 'Employee' subfield participates in the conditions part and also it is the under modification data substructure (of frame object), it defines also which is the concrete repetition of the subfield that will be updated (modified). Here we have a fifth semantic form of 'Alter data' statement. Obviously, the update will take place in the first repetition of subfield 'Employee', in the second repetition of tag 'Actions', in the frame object of table 3. This statement can also be expressed with the naive solution by simply removing the indices.

# Alter data set Employee = 'Nikitas' when entity = 'Projects' and subfield[i] = 'Action' restr data = 'watering' and subfield[i] = 'Deadline' restr data = '20/04/2007'

In this statement we require that both values 'watering' and '20/04/2007' to coexist in the same repetition of tag 'Actions'. However, since the under modification subfield 'Employee' does not participate in the conditions part, the statement does not specify any concrete repetition of the under modification subfield. That means that the modification applies in any repetition of the under modification subfield, inside the affected tag repetition. Here we have a fourth semantic form of 'Alter data' statement. Obviously, the update will take place in any repetition of subfield 'Employee', in the second repetition of tag 'Actions', in the second frame object of table 5 (the one with 'Project_code' equal to 'Proj002'). This statement can also be expressed with the naive solution by simply removing the indices.

# Alter data set Employee = 'Nikitas' when entity = 'Projects' and tag = 'Budget' restr data = '200,000'

In this statement we require that the modifications (of subfield 'Employee') will take place in frame objects having tag 'Budget' equal to '200,000'. We don't provide any restriction that could narrow the update to specific repetitions of tag 'Actions' in the modified frame objects. That means that every repetition of subfield 'Employee', in every repetition of tag 'Actions', in the frame objects having tag 'Budget' equal to '200,000', will be modified. Here we have a third semantic form of 'Alter data' statement. Obviously, the update will take place in any value of subfield 'Employee', in the first frame object of table 5 (the one with 'Project_code' equal to 'Proj001'). This statement is the same in the present and in the naive version of CUDL.

# Alter data set Employee = 'Nikitas' when entity = 'Projects' and subfield[i] = 'Action' restr data = 'watering' and subfield[j] = 'Deadline' restr data = '20/03/2007'

In this statement we require that values 'watering' and '20/03/2007' to coexist in the same frame object, without requiring to coexist in the same repetition of tag 'Actions'. Since the statement does not focus the restrictions in any substructure of frames, the modifications of subfield 'Employee' can not also be focused in specific tag or specific subfield repetitions. Here we have a third semantic form of 'Alter data' statement. Obviously, the update will take place in any value of subfield 'Employee', in the third frame object of table 5 (the one with 'Project_code' equal to 'Proj003'). This statement can not be expressed with the naive version of CUDL.

# Alter data set Employee = 'Nikitas' when entity = 'Projects' and subfield[i] = 'Employee' restr data = 'Manolis' and subfield[j] = 'Action' restr data = 'pruning' and subfield[k] = 'Deadline' restr data = '30/03/2007'

In this statement we impose three restrictions on subfields of the tag 'Actions' but do not require being satisfied (coexist) in the same tag repetition. This statement, as the previous, does not focus the restrictions in any substructure of frames and consequently the modifications of subfield 'Employee' cannot be focused in any specific tag or any specific subfield repetition. Here we have a third semantic form of 'Alter data' statement. Obviously, the update will take place in any value of subfield 'Employee', in the first and second frame objects of table 5 (the frame objects with 'Project_code' equal to 'Proj001' and 'Proj002', respectively). This is another statement that can not be expressed with the naive version of CUDL.

# Alter data set Employee = 'Nikitas' when entity = 'Projects' and subfield[i] = 'Employee' restr data = 'Manolis' and subfield[i] = 'Action' restr data = 'pruning' and subfield[j] = 'Deadline' restr data = '30/03/2007'

In this statement we impose three restrictions on subfields of the tag 'Actions' and require two of them being satisfied (coexist) in the same tag repetition but do not require the third of them being satisfied in the same tag repetition. The condition of this statement is satisfied by the first frame object of table 5 (the frame object with 'Project_code' equal to 'Proj001') because in its third repetition of tag 'Actions' the values of Manolis' and 'pruning' coexist and the value '30/03/2007' exists in the second repetition of tag 'Actions'. This statement, as the previous two ones, does not focus the restrictions in any substructure of frames and consequently the modifications of subfield 'Employee' cannot be focused in any specific tag or any specific subfield repetition. Here we have a third semantic form of 'Alter data' statement. Obviously, the update will take place in any value of subfield 'Employee', in the first frame object of table 5. This is another statement that cannot be expressed with the naive version of CUDL.

There is a simple rule for distinguishing the semantic form of any CUDL subfield modification statement that reveals from the above examples. This rule applies whenever a DPS contains restrictions on two or more subfields of the tag that hosts the modified subfield. This rule can be adapted for any subfield data modification (update, removal and insertion) statement. Here we will present it in regard to the 'Alter data set <*subfield name*> = …' (update) statement:

- Whenever the conditions in the DPS does not contain indices on the modified and its brethren subfields or the indices are same and the modified subfield is included in some of the DPS's conditions then the semantic form is the fifth.
- Whenever the conditions in the DPS does not contain indices on the brethren subfields or the indices are same and the modified subfield is not included in any of the conditions composing the DPS then the semantic form is the fourth.
- In any other case the semantic form is the third.

4.1 Combining DPSs in the Same Frame Object

In section 3.2 we had come up against a dilemma concerning the selection of the semantic form when a frame object verifies two (or more DPSs) with different semantic forms. There, we had given two alternative rules, namely: "we handle every frame object based on the biggest semantic form determined by the disjunctive conditions which verify it" or "we handle every frame object based on the smallest semantic form determined by the disjunctive conditions which verify it". Consequently, the dilemma solution is a mater of selection between two alternative rules. However this alternative rules, nonetheless they constitute the basis of the general idea, they don't cover all the possible DPS's semantic forms combinations. Thus, they should be refined. The refinement follows:

a. The DPSs combination is stepped. Firstly, the first DPS is combined with the second. Next, the result of the first combination is combined with the third DPS. We continue in a similar way if more than three DPS exist.

b. The alternative combination rules (refined) are:

	Nested affected structures	No intersection of affected structures
Smallest semantic form	Select the wider affected structure (e.g. select the affected structure of the DPS having 3rd semantic form, whenever we combine one DPS of the third with a DPS of the fourth semantic form)	Select the union of the affected structures of the combined DPS (e.g. if the affected structure of one DPS is the 2nd repetition of some tag 'X' and the affected structure of the second DPS is the 4th repetition of the same tag then the affected structure of the combined DPS is the 2nd plus the 4th tag repetition of tag 'X')
Biggest semantic form	Select the narrower affected structure (e.g. select the affected structure of the DPS having 4th semantic form, whenever we combine one DPS of the third with a DPS of the fourth semantic form)	same

Of course the combination rules operate and in more complicated than simple nesting and no intersection cases. The following examples make it clear. (The indices used in the examples reflect the semantic forms, namely: 3 when the affected structure is the whole frame, 4 when the affected structure is a concrete tag repetition, 5 when the affected structure is a concrete subfield repetition.)

Example 1:

Let us assume that we have to combine $X_4 \cup Y_4$ with Z_5 (for example the third and fifth repetition of some tag with the fourth subfield repetition of the third tag repetition of the same tag) then

- In case of activation of the smallest semantic form rule the affected structure remains $X_4 \cup Y_4$

- In case of activation of the biggest semantic form rule the affected structure becomes $Z_5 \cup Y_4$

Example 2:

Let us assume that we have to combine $X_4 \cup Y_4$ with $X_4 \cup K_4$ (for example the third and fifth repetition of some tag with the third and sixth tag repetition of the same tag) then

- In both rules, the affected structure becomes $X_4 \cup Y_4 \cup K_4$

Example 3:

Let us assume that we have to combine $X_4 \cup Y_4$ with $L_5 \cup M_5$ and moreover L_5 is a subfield repetition of X_4 but M_5 is not subfield repetition of any of X_4 or Y_4 (for example the third and fifth repetition of some tag with the second subfield repetition of the third repetition of the same tag and the first subfield repetition of the seventh repetition of the same tag) then

- In case of activation of the smallest semantic form rule the affected structure becomes $X_4 \cup Y_4 \cup M_5$
- In case of activation of the biggest semantic form rule the affected structure becomes $L_5 \cup Y_4 \cup M_5$

5. Conclusions

In this paper we have addressed the problem of co verification of conditions in the CUDL language, designed to manage dynamic database environments such as the FDB model. We have shown the difficulties that arise from the evaluation of conditions into the composite data structures of the FDB model. We have made clear why this problem appears and why it is important. We have provided two solutions to address this problem. The first one (the “naive” one) is more easily understandable and does not impose complexity to the language’s syntax. The second one is an improvement of the first but more demanding. It requires more advanced users and very carefully written CUDL language statements.

References

[1] Andany, J., Leonard, M., Palisser, C (1991). Management of schema evolution in database. *In Proc. of the 17th VLDB Conf.,* Barcelona, p. 161-170

[2] Banerjee, J., Kim, W., Kim, H. and Korth H. F (1987). Semantics and Implementation of Schema: Evolution in Object-Oriented Databases. *ACM SIGMOD* 16 (3) 311 - 322

[3] Bertino, E (1992). A View Mechanism for Object-Oriented Databases. *In: Pirotte, A., Delobel, C. and Gottlob G. (eds): Proc. of Advances in Database Technology (EDBT’92) - 3rd international Conference on Extending Database Technology,* Lecture Notes in Computer Science, 580, Springer, p. 136-151

[4] Bratsberg., S. E (1992). Unified Class Evolution by Object-Oriented Views, *In proceedings of the Internaltional Conference / the Entity-Relationship Approach,* Springer Verlag, p. 423-439

[5] Clamen., S. M (1994). Schema Evolution and Integration. *Distributed and Parallel Databases, Special issue on distributed/parallel database object management* 2 (1) 101-126

[6] Ferrandina, F., Meyer, T., Zicari, R., Ferran, G., Madec, J (1995). Schema and database evolution in the O2 object database system. *Proceedings of the 21th International Conference on Very Large Databases (VLDB ‘95),* Zurich, p. 170-181

[7] Karanikolas, N. N., Nitsiou, M., Yannakoudakis, E. J., Skourlas, C (2007). CUDL language semantics, liven up the FDB data model. *In Eleventh East-European Conference on Advances in Databases and Information Systems (ADBIS 2007),* September 29 - October 03, 2007, Varna, Bulgaria

[8] Karanikolas, N. N., Nitsiou, M., Yannakoudakis, E. J., Skourlas, C (2008). Conceptual Universal Database Language (CUDL) and Enterprise Medical Information Systems. *In 10th International Conference On Enterprise Information Systems (ICEIS’2008),* June 12-16, 2008, Barcelona, Spain.

[9] Karanikolas, N. N., Nitsiou, M., Yannakoudakis, E. J., Skourlas, C (2009). CUDL Language Semantics: Updating Data. *Journal of Systems and Software* 82 (6) 947-962, doi:10.1016/j.jss.2008.12.031

[10] Karanikolas, N. N., Vassilakopoulos, M. Gr (2009). Conceptual Universal Database Language: Moving Up the Database Design levels. *In Thirteen East-European Conference on Advances in Databases and Information Systems (ADBIS 2009),* September 7-11, 2009, Riga, Latvia.

[11] Kazimierz, S (1985). Semantics of Query Languages for Network Databases. *ACM Transactions on Database Systems (TODS)* 10 (3) 347 – 394

[12] McKenzie, E., Snodgrass, R (1990). Scheme evolution and the relational algebra. *Information Systems,* 15 (2) 207–232

[13] Mohamed A.-N., Estublier, J (2000). Schema evolution in software engineering databases - a new approach in Adele environment. *Computers and Artificial Intelligence,* 19 (2)

[14] Monk, S., Sommerville, I (1993). Schema Evolution in OODBs Using Class Versioning. *SIGMOD Record* 22 (3) 16 - 22

[15] Roddick, J. F (1992). Schema evolution in database systems: an annotated bibliography. *SIGMOD Record* 21 (4) 35 - 40

[16] Skarra, A. H., Zdonik, S. B (1987). Type Evolution in an Object-Oriented Database. In Mit Press Series. Computer Systems, Research directions in object-oriented programming, Cambridge, MA, USA, MIT Press, p. 393 - 416

[17] Zdonik, S. B (1990). Object-Oriented Type Evolution. In Advances in Database Programming Languages, Addison-Wesley

[18] Zicari, R (1991). A Framework for Schema Updates in an Object-Oriented Database System. *In Proceedings of the Seventh International Conference on Data Engineering,* p. 2 -13

[19] Yannakoudakis, E. J., Tsionos, C. X., Kapetis, C. A (1999). A new framework for dynamically evolving database environments. *Journal of Documentation* 55 (2) 144-158.

[20] Yannakoudakis, E. J., Diamantis, I. K (2001). Further improvements of the Framework for Dynamic Evolving of Database environments. *In Proceeding of the 5th Hellenic – European Conference on Computer Mathematics and its Applications (HERCMA 2001),* Athens, Greece

[21] Yannakoudakis E. J., Nitsiou, M (2006). A new conceptual universal database language (CUDL). *In Second International Conference From Scientific Computing to Computational Engineering (2nd IC-SCCE),* Athens, Greece

[22] Yannakoudakis E. J., Nitsiou, M., Skourlas, C., Karanikolas, N. N., 2007. Tarski algebraic operations on the frame database model (FDB). *In proceedings of the 11th Panhellenic Conference in Informatics (PCI 2007),* Patras, Greece