A SOLUTION IMPLEMENTED IN SPARQL FOR THE RAMIFICATION PROBLEM IN TEMPORAL DATABASES

By

KAPETANAKIS FANOURIOS

Ramification problem to Database Systems in sparql language

A THESIS

submitted in partial fulfillment of the requirements for the degree

Informatics Engineering

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

HELLENIC MEDITERRANEAN UNIVERSITY

Heraklion, 2023

Contents

Abstract 4
1. Introduction 5
2. Literature Review 8
2.1 Conventional Databases 8
2.1.1 The Ramification Problem 8
2.2 The Qualification Problem 11
3. Databases 13
3.1 What are databases? 13
3.2 Database Management Systems (GRD) 14
3.3 Time Databases (Temporal Databases) 15
3.4 Spatial Databases (Spatial Databases) 16
3.5 Space-time databases (SPATIOTEMPORAL Databases) 16
3.6 Ramification Problem 16
3.7 Proposed Solutions 19
3.8 Static rule Generation Algorithm 22
3.9 Classes 24
3.10 Variables 27
3.11 Sections 29
3.12 Arrays 31
3.12.1 Double arrays 32
4. Methods of Implementation 33
4.1 Java programming language 33
4.2 SQL programming language 34
4.2.1 Using SQL to convert static rules 35
4.3 Sparql Queries 38
Conclusions 42
References 44
Appendix 47

Pictures and Table Contents

Figure 1: The complex circuit. 9
Figure 2:The relationship between events and actions and the passage of time.................. 18
Figure 3: Variables. 26
Figure 4: Fluents class.. 28
Figure 5: Secions class. 30
Figure 6: MainPallet class. 31
Figure 7: Temporal Allen relations. 36

Table 1: Func dataset. 38

Abstract

The topic of ramification is concerned with determining the indirect implications of activities. A solution to this problem in database systems allows for reasoning about database dynamics and verification of consistency properties. This problem is growing increasingly complex in temporal databases, and no suitable solution has yet been proposed. We examine these two challenges in the context of temporal databases in this paper and provide a polynomial complexity solution based on the Situation Calculus language. This method expands on previous proposals for dealing with comparable problems in traditional (non-temporal) databases.

Περίληψη

Το θέμα της διακλάδωσης αφορά τον προσδιορισμό των έμμεσων επιπτώσεων των δραστηριοτήτων. Μια λύση σε αυτό το πρόβλημα στα συστήματα βάσεων δεδομένων επιτρέπει τον συλλογισμό σχετικά με την δυναμική της βάσης δεδομένων και την επαλήθευση των ιδιοτήτων συνέπειας. Αυτό το πρόβλημα γίνεται όλο και πιο περίπλοκο στις χρονικές βάσεις δεδομένων και μέχρι σήμερα δεν έχει προταθεί καμία κατάλληλη λύση. Στην εργασία αυτή, εξετάζουμε αυτές τις δύο προκλήσεις στο πλαίσιο των χρονικών βάσεων δεδομένων και παρέχουμε μια λύση με πολυωνυμική πολυπλοκότητα βασισμένη στη γλώσσα του Λογισμού της Κατάστασης. Αυτή η μέθοδος επεκτείνει προηγούμενες προτάσεις για την αντιμετώπιση παρόμοιων προβλημάτων σε παραδοσιακές (μη χρονικές) βάσεις δεδομένων.

Keywords: Databases, Reification problem, Java, Sparql.

1. Introduction

One of the major study concerns of the knowledge representation and planning communities during the last two decades has been reasoning about action and change. Action theories, which provide an axiomatic foundation for controlling change, are applicable to a broad range of fields, including software engineering (cognitive) robotics, and data/knowledge base systems. In this paper, we will look at database systems. Databases are dynamical systems with changing contents as a result of database transactions. Because an atomic database transaction may be considered an action, we can argue that changes in a database occur as a result of actions. Changes to a database may have an impact on its consistency. Appropriate measures must be used to ensure that a database never reaches an inconsistent state. To enforce this criterion, one must be able to dictate the specific changes (direct or indirect) that are produced by the execution of an action, and hence select which actions should be permitted to execute. McCarthy and Hayes first proposed these interconnected difficulties as the ramification and qualification problems in [1].

Assume the database's contents are expressed as propositions specifying the location of each item in the room, as illustrated below:

on(bookcase, x1) on(table, x2) on(book, x1) (1)

on(bottle, x2) on(chair, x3) (2)

Two things cannot share the same room space unless they are stacked on top of one other. As we can see, the book and the shelf are in the same place. This is due to the presence of a restriction demanding that books be kept on the bookcase (respectively for the bottle and table). The execution of the action move (chair, x_4) causes the chair's position to change from x_3 to x_4. The sole immediate result of this action is a change in the position of the chair. However, acts might have unintended consequences. The impact of the action move (bookcase, x_5) is both direct and indirect. The direct effect is to move the bookcase, but the indirect impact is to move the book, because the book is in the bookshelf and so moves along with the bookcase. The existence of the limitation that the book must be on the bookshelves causes the indirect effect. The existence of limitations causes such indirect consequences. The ramification issue [2] is the simple statement of an action's indirect consequences in the face of limitations.

In terms of the activities themselves, not all actions are permitted to take place in any given scenario. There are some preconditions for any action that, when met, allow the action to be carried out. Because a table occupies the target place in the preceding example, the operation move (bookcase, x_2) cannot be executed. The action move (bookcase, x) can only be performed if the location x is free. As a result, the prerequisite for the action move (p, x) is evident (x).

Whenever an action takes place it is necessary to be able to understand all the direct and indirect effects of this action. Otherwise, the contents of database may not satisfy the constraints that describe the consistent states of the database, and thus the database will be inconsistent. In the above example, after the execution of the action move (bookcase, x5), if the position of the book does not change, then the contents of database violate the aforementioned constraint.

Such indirect effects are caused by the presence of constraints. The ramification problem [3,4] refers to the concise description of the indirect effects of an action in the presence of constraints. As far as the actions themselves are concerned, not all actions are allowed to take place in any given situation. For each action there are some preconditions which when true, they permit the action's execution. In the previous example, the action move (bookcase, x_2) is not allowed to execute because a table occupies the target position. The action move (bookcase, x) can be executed only if the position x is clear. So, the precondition of action move (p, x) is clear(x).

The problem of determining the context in which an action is allowed to execute is the qualification problem [22]. As we observe, both problems appear in the context of our example and in the context of any changing world, giving rise to the qualified ramification problem [24]. To give a brief description of this problem consider that in above example the table and the chair are somehow connected. When the robot moves the table to a new the location, the chair will be moved too. Now the action move (table, x_3) can be executed because the indirect effect of the action move (table, x_3) is to change location of the chair. Hence, the preconditions clear (x_3) holds. Before the execution of the action move clear (x_3) was false. In cases, like this a solution must be able to take

into account the fact that the indirect effects of actions may make action preconditions true.
Many domains about which we wish to reason are dynamic in nature. Reasoning about action is concerned with determining the nature of the world (what holds in the world state) after performing an action in a known world state, and has found application in areas such as cognitive robotics.

The guarantee of consistency of data that is stored in a database is a very important and difficult problem. The consistency of data is determined by the satisfaction of the integrity constraints in the different databases states (situations). A database state is considered valid (consistent) when all integrity constraints are satisfied. New situations arise as the result of action (transaction) execution. In a new situation (which includes the direct effects of the transactions) the database may become inconsistent because some integrity constraints are violated by means of indirect effects of actions. Thus, it is necessary to produce all indirect effects in order to determine the satisfaction of the integrity constraints. In a large database system with hundreds or thousands of transactions and integrity constraints, it is extremely hard for the designer to know all the effects that actions may have on the consistency of the database [21].

The rest of paper is organized as follows: in section 2 we review the most prevalent solutions which have been proposed for addressing the ramification and the qualification problems in the context of conventional (non-temporal) databases. We also briefly examine the qualified ramification problem. The ramification and qualification problems in temporal databases are examined at section 3, and a solution is presented at section 4. The paper concludes with a summary and directions for further research.

2. Literature Review

2.1 Conventional Databases

2.1.1 The Ramification Problem

Many solutions have been proposed to the ramification problem. The Situation Calculus [1] underpins the vast majority of them. The situation calculus is a second-order language that expresses the changes that occur as a result of activities in an area of interest. A series of acts represents one potential development of the world and is represented by a first-order word called a circumstance. The initial circumstance S0 is a distinct word that denotes a scenario in which no action has yet happened. Do(a, s) is a binary function that returns the situation resulting from the execution of an action a while in situation s. Predicates, known as fluents, can vary their truth value from one scenario to the next, and one of their arguments is a situation term.

Those based on the minimal modification approach are among the simplest solutions presented [2,3]. These solutions imply that when an action occurs in a scenario S, the consistent situation S with the fewest modifications from the situation S should be found. Consider the modeling of a basic circuit with two switches and one bulb as an example. The lamp must be turned on when both switches are turned on. If one of the switches is turned off, the lamp must not be turned on. Consider the condition S = ¬up(s1), ¬up(s2), ¬light}. The action toggle switch(s2) changes the circuit state to S' = {up(s1), up(s2), ¬light}, which is inconsistent. S1 = {up(s1), up(s2), light} and S2 = {up(s1), ¬up(s2), ¬light} are two consistent scenarios. It makes sense to turn on the lighting, however turning off the switch s2 does not. It is fair for the lamp to light up as an indirect result of another switch being "up," but it is not reasonable to "down" a switch as an indirect effect of another switch being "up." As a result, we choose S1 over S2. Because they are both so similar to the original state, the minimum modification strategy cannot choose between them.

The solutions based on fluent categorization [4,5,6] tackle the aforementioned difficulty. Primary and secondary fluents are distinguished. A primary fluent can only change as a direct result of an activity, whereas a secondary fluent can only change as an indirect result of an action. Following an action, we select the condition with the fewest changes in main fluents. The separation in the above example is F_p = { up(s_1), up(s_2) } and F_s = light, where F_p and F_s are the main and secondary fluents, respectively. Now we'll look at Situation S1, which has no modifications in the major fluents.

The ramification problem can only be solved by categorizing fluents if all fluents can be categorized. This technique is insufficient if some fluents are main for some actions and secondary for others. Consider the circuit shown in Figure 1. The following formulae express the integrity restrictions speculating on the behavior of this system:

$$\text{light} \equiv \text{up}(s1) \wedge \text{up}(s2) \quad (3)$$

$$\text{relay} \equiv \neg\text{up}(s1) \wedge \text{up}(s3) \quad (4)$$

$$\text{relay} \supset \neg\text{up}(s2) \quad (5)$$

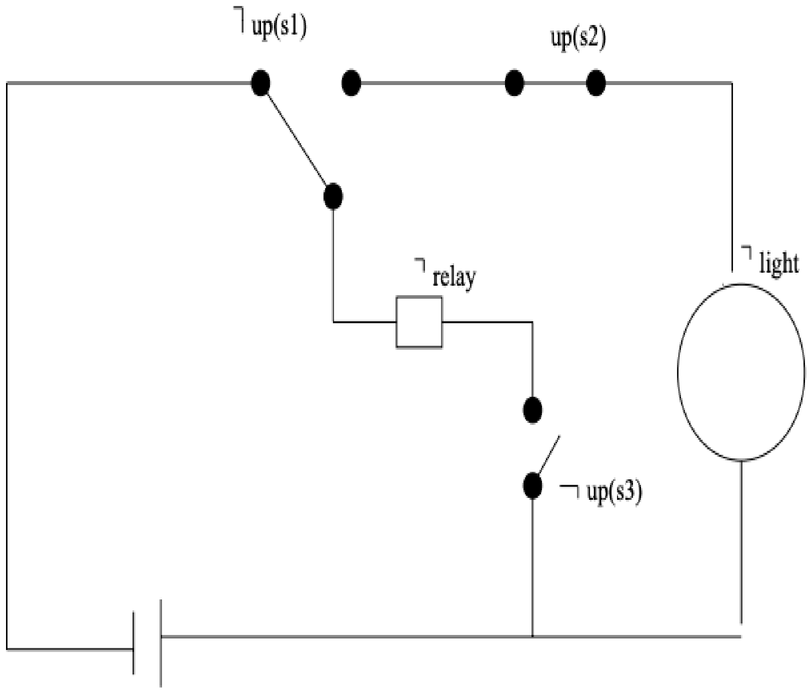


Figure 1: The complex circuit.

The fluents up(s1) and up(s3) are now main, whereas relay and light are secondary. The fluent up(s2) is the primary action toggle switch(s2) and the secondary action toggle switch(s2) (s1). When up(s1) and up(s3) hold after the operation toggle switch(s1) is completed, the proposition up(s1) up(s3) holds. This indicates that the fluent relay has become true. When the fluent relay is true, up(s2) must be true as well. As a result, the operation toggle switch(s1) has an indirect impact up (s2). This signifies that the fluent up(s2) is subordinate to the toggle switch operation (s1).

As it is observed, the indirect consequence of an action is determined by the database context. The context is a conjunctive statement composed of database fluents. and give the capability condition for actions' consequences to be realized In the above example, the context that must be in the database for the action toggle switch(s1) to have an indirect effect ¬up(s2) is the fluents up(s1) ∧ up(s3).

The preceding methods have the disadvantage of being unable to capture the dependency that exists between the indirect consequences of action and the context contained in the database. The resolution of causal links captures this dependency [2,3,4]. Each causal connection is made up of two elements. The first portion, known as context, is made up of a single fluent formula that, when true, demonstrates a causal link between an action and its result. The latter is the indirect result of an activity (called the cause of this effect). A causal connection has the following form:

e causes ρ if Φ

→ where e represents an action, ρ is the direct/indirect impact, and Φ is the context

McCain and Tuner's [7] language is one solution based on the concept of causal interactions. This language encompasses both static and dynamic laws. A static law is a formal formulation.

caused F if G

The essence of this static law is that when a formula G is valid, the fluent F must also be true. A dynamic law is a type of expression.

U causes F if G

A dynamic law of this type means that an action U has the direct effect F if the proposition G is true. For example, in the preceding section, the following dynamic legislation is defined:

move(x, l) causes on(x, l) if free(l)

A static law is also defined.

$$\text{on}(x, l) \text{ if } \text{on}(y,l) \land \text{on}(x, y)$$

This law states that if one item x is on another object y that is at position l (perhaps after some movement), then x must also be at l. It is important to note that static laws record the indirect impacts of activities, whereas dynamic laws capture the direct repercussions of actions.

2.2 The Qualification Problem

We now review briefly solutions proposed for solving the qualification problem. The so-called default solution [4] suggests that, for each action a, we must determine a formula Fa which, when true, prohibits action a from executing. The formula Fa is a disjunction of the form:

$$F^a \equiv \bigvee F_i$$

where each Fi is a fluent formula. When any of the disjoin Fi is true, the action a can not execute. Returning to our example, the disabling fluent formula of the action move(x, l) has one disjunct:

$$\boxed{F^{move(x,l)} \equiv on(y,l) \wedge x \neq y}$$

We say that when the formula Fa holds then the action a is disqualified and thus it cannot execute. This is represented by employing a predicate disq as:

$$\boxed{F^{a} \supset disq(a)}$$

Another solution [24] is an extension of the minimal-change possible-worlds approach that has been suggested for solving the ramification problem. After each action a executes, we try to find a consistent situation which contains all direct and indirect effects of a. If there is at least one such situation, then the action can execute, otherwise it cannot.

3. Databases

Every day we meet a variety of problems where you have to face and find a solution. Also, we are faced with some decisions either to be easy or difficult and we need to study it to come up with the most appropriate. An example that we can give is as follows: Most have been found to stand in a showcase with clothes and see various prices from blouses, trousers, shoes and various other species. Each of them has a price. Knowing that we have for example 150 euros we start and make a variety of calculations to come up with what we can take with the money we have, that is to say, to make a decision better. All of the above is data. Tags with prices, money we all are data that we need to work to draw useful and exploitable information on our final decision. The data can be expressed by a multitude of numbers, words, quantities, ideas and functions. The next step is to use all the data we have drawn to process them to collect information that will lead us to reach the decision. In our case with various acts, we will end up with An ideal market with the money we have after information we will export from acts that we will make by editing our data.

3.1 What are databases?

By database, we mean a set of information that is organized in such a way as to have the ability to manage them to inform them with easy access to them. Databases are "in order to simplify data storage, Recover them, as well as their processing and deletion in collaboration with various other data processing functions.

In ancient times where technology has not been developed, there were therefore no computers, data save in bulky data repositories that we call books. Then over time improving technology as well as expansion of knowledge created the need to transfer all communities Books in the first real libraries, namely a "data". The purpose of the library was to ensure that data can be stored and recovered easily and effectively. Since 1960 to today, since you have begun to develop the computer, we have various database models used. Each if from them had advantages as well as disadvantages. In 1960-1980 the Flat Files model was used. In 1970-1990 we have the Hierarchical Database model where it contained hierarchically arranged data [10].

We can liken as a family tree where there is a parent and child relationship and therefore every parent may have many children while a child can have a parent, etc. In 1970-1990 we have the Network Database model that his inventor is Charles Bachmann

[10]. The network database model It was an evolution of the hierarchical database model. It was designed to solve some problems of Hierarchical Database and more specifically the lack of flexibility. This model allows each child to have a lot of parents that was not allowed in the hierarchical model. Therefore, represents more complex data relationships. Since 1980, I am using the Relational Database model (relational database model). The above model proposed by E. F. Codd left behind the previous models and made a big step in front of the databases. The relational model allows the entities to be associated with each other through a common feature.

In the tables there are primary keys that identify the table information. The relationship between the tables can then be set through the use of foreign key keys on a table connected to the primary table key. In this way, the relational database model gives us an excellent feature, storing infinite information using small tables. Data access is extremely effective so the user simply by submitting a query on his base returns the requested information [11]. Related databases are created using a computer language SQL (Structured Query Language) which is easily readable by humans. From 1990 to date there is also the Object-Oriented Database model (object-based database). In the above database system data or information is presented in object format, as in each object-oriented programming language. The difference with the relational database is that the object-oriented database operates in the context of actual data languages such as Java or C ++.

3.2 Database Management Systems (GRD)

To create and manage a database We need a system software ie a database management system (DBMS). The GBR gives the user as well as developers to create to modify, recover, delete, update as well as managing data. The SBC is very useful because it is the medium for the interface between the database and end users or applications, achieving that our data is organized and easy access.

Most of the database management systems use the structured SQL search language [13]. Most GRDs to facilitate users provide a graphical environment in which you can easily manage and process the data in the background these processes are running through SQL. Today there are many Commercial GRDs that are open source. In fact, to choose the most appropriate SBR is a complex work. High-level SBCs at the top of the market nowadays are Oracle, Microsoft SQL Server, IBM DB2 which is one of the most reliable options for large data systems. For small companies or domestic use, the Microsoft Access as well as FileMakerPro.

3.3 Time Databases (Temporal Databases)

A Temporal Database is a database that is designed to handle time-sensitive data. Databases often retain information just about the present state and not about previous states. For example, in an employee database, if a specific person's address or pay changes, the database is updated, and the previous value is no longer there. However, for many applications, it is necessary to save the previous or historical values as well as the moment at which the data was changed. That is, understanding evolution is essential. This is when temporal databases come in handy. It saves data from the past, present, and future. Temporal data refers to any data that is time dependent and is saved in temporal databases.

Temporal databases hold information about real-world conditions throughout time. A Temporal Database is a database that has built-in capabilities for dealing with time-related data. It holds information about all occurrences' past, present, and future times.

In databases as in fact a very important element is time because it is connected to almost everything that concerns us. Over time our data and therefore our information is often changing and this creates various problems in our database. However, there must be full support of the time-changing nature of the things they represent, for this reason the temporal databases (Temporal Databases) were created [15]. At specific points, events are recorded in which objects and relationships between objects are changing over time. Modeling the time dimension in the real world is a capacity very important and fully necessary for many IT applications. In a variety of disciplines, We meet databases that are in relation to time, such as accounting, econometrics, geographic information systems, control systems, banks, even in sectors with scientific data analysis and in other cases.

Conventional databases are not useful in such cases because they are addressed to a single business situation at a time. The problem you create with conventional databases is that while the contents of the base continue to change them as new information is added, these Changes are recognized as modifications to the existing situation, resulting in the older data to be deleted from the database. The contents containing a database in current time are considered as snapshots. In a database in relation to time (TEMPORAL DATABASE) We have full support for the preservation of time-changing data and specialized questions related to the past, present and future of these data, which is impossible to occur in conventional databases [15].

3.4 Spatial Databases (Spatial Databases)

Another equally very important concept associated with bases Data out of time is the space. For this reason, beyond Temporal Databases, we also have spatial databases, which are databases designed for storage and access to spatial data or data that define the geometric space [16] [17]. Such data are mainly linked to geographic locations. Storing data in a spatial database has the Form of coordinates, points, lines, polygons and topologies [17]. A more difficult situation that can handle a spatial database is to handle data that are more complex, i.e. three-dimensional objects, topological coverage and linear networks [16].

3.5 Space-time databases (SPATIOTEMPORAL Databases)

Until now, we have seen the database function with time the time databases (Temporal Databases) and the database function with the space databases (Spatial Databases). However, the need to have a database in relation to time and space created spatial databases (spatiotemporal databases) [15]. Space-time databases manage spatial data in which geometric characteristics are dynamically changing. Space-time databases have many important applications, use them in geographic information systems, GPS, traffic monitoring systems even in environmental information systems. It is therefore perceived that space-time database systems are very active in the field of databases [16].

3.6 Ramification Problem

All actions in temporal database systems take place at precise times in time. Items and interactions between objects exist across time as well. A fluents value is determined by the time instant at which it is evaluated. As a result, a finer-grained change description technique is necessary in this case. Remember that in traditional (nontemporal) databases, we only need to know the value of fluents after an action has occurred.

In this part, ramification issues are discussed in the context of temporal databases. Assume the following regulation is in effect: if a public employee commits a misdemeanor, he is deemed unlawful for the next five months.

When a public employee is found to be unlawful, he or she must be suspended for the duration of the illegality. A public employee can only be promoted if he or she has held the same post for at least five years and is not under suspension. These are stated in propositional form by the limitations listed below [21].

$$\text{occur}(\text{misdemeanor}(p), t) \supset \text{illegal}(p, t_1) \land t_1 < t + 5m$$

$$\text{illegal}(p, t_1) \supset \text{suspended}(p, t_1)$$

$$\text{suspended}(p, t_1) \lor (\text{sameposition}(p, d) \land d < 5y) \supset \neg\text{receivepromotion}(p, t_1),$$

→ where t and t_1 are temporal variables and the predicate occur(crime(p), t) indicates that the action crime(p) is carried out at time t In a temporal database, we must represent the direct and indirect impacts of an activity not just in the immediately following scenario, but also potentially in many future situations. In the above example, the action misdemeanor(p) has the indirect consequence of suspending the public employee for the next five months. During this five-month time, a variety of additional events may take place, resulting in a variety of outcomes. In all of these cases, the action misdemeanor(p) has the indirect consequence of suspending the sentence (p).

In temporal databases, causal links cannot address the ramification problem since they only define the direct and indirect consequences for the next circumstance. All other ramification problem solutions in traditional databases share the same flaw. Furthermore, as we can see, the execution of the action misdemeanor(p) prohibited the action from receiving promotion for the next five months. Because they cannot capture the fact that one activity might disqualify another for a specified time span, the solutions provided for the qualification problem in conventional databases cannot handle the qualification problem in temporal databases.

The aforementioned flaw can be mitigated by creating a time-based connection between events and actions. Previous works [7] proposed such a connection. We accept the relationship first proposed in [7] and illustrated in Figure 2. The situation axis is the first parallel axis, the time axis is the second, and the actions axis is the third. All activities are assumed to be immediate. When an action occurs, the database is transformed into a new state.

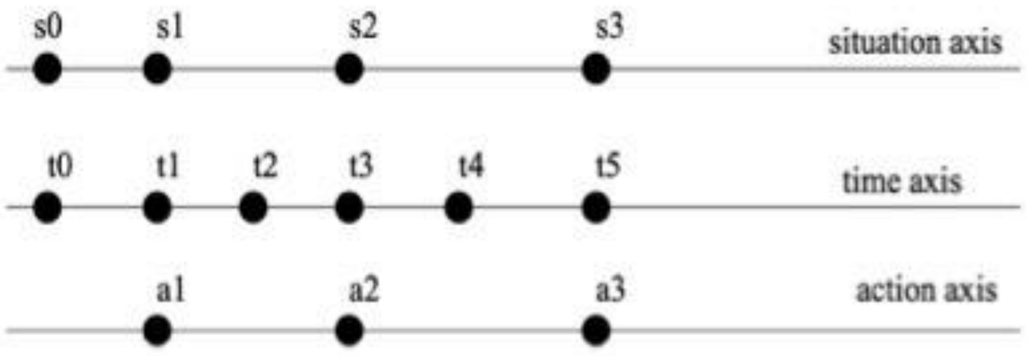


Figure 2: The relationship between events and actions and the passage of time.

A solution provided to the ramification problem in temporal databases in [7]. More particularly, we declare two axioms for any pair (a, f) comprising an action a and a fluent f:

a(t) causes f(t') if E^{+}_{fa},

a(t) causes ¬f(t') if E^{-}_{fa}

→ where E^{+}_{fa} and $E^{-}_{fa}$are the formulae that must hold for fluent f to become true or false at time t following the execution of action an at time t. The axioms stated above must be given for every action and the fluents affected by its execution. The maximum number of axioms that must be defined is O(2 * F * A), where F represents the number of fluents and A represents the number of actions. The next section presents an enhancement to this approach in terms of the number of axioms required. The better solution necessitates the formulation of O(A + 2 * F) such causal rules.

3.7 Proposed Solutions

The suggested technique is based on scenario calculus [9] and McCain and Turner's work [8]. We expanded on McCain and Turner's prior idea [8]. McCain and Turner contend that there is a dynamic rule for each action A.

$$\boxed{\text{occur(A)} \text{ -> } \wedge \text{ F}}$$

→ which signifies the immediate consequences of an action There are also two static rules for each fluent f, one for affirmation and one for denial.

$$\boxed{\begin{array}{c} G \rightarrow f \\ B \rightarrow \neg f \end{array}}$$

→ G and B are fluent formulae. The indirect consequences are shown by the static rules. The former are dynamic because they are assessed after the activity is completed, whereas the latter are static since they are evaluated at each time point. This method is appropriate for temporal databases because it allows for the assessment of the real value of each fluent at each time point. At each time point, we run a set of static rules until no change happens.

Following that, we identified the real values of all fluents. As we discussed in [9], in a temporal database, we must represent the direct and indirect impacts of an action not only in the next scenario, but also in many future situations. This implies that we want a solution that distinguishes between direct and indirect impacts (dynamic and static rules). This is required because another action may occur in the meanwhile, canceling the indirect consequences for the remainder of the time span. This distinction is possible using McCain and Turner's technique. We improved this strategy in [10] to address the ramification problem in temporal databases and to reduce the number of static rule executions.

The proposed approach is based on the concepts of McCain and Turner [7], who advocate using static rules to capture the indirect consequences of actions (based on the integrity requirements of the specific domain) and dynamic rules to reflect the direct impacts of actions. In the proposed method, there is a dynamic rule of the type A -> $\wedge F_i(L_i')$ for each action A, where each Fi(L_i') is fi(Li) or ¬fi(Li) for a given fluent f.

The preceding rules outline the immediate consequences of an action. Furthermore, for any fluent f, we define two rules, G(L) f (L) and B(L) f (L). The G(L) is a fluent formula that, when true (at list L), causes fluent f to become true at the time intervals in list L (corresponding for B(L)). These rules encapsulate an action's indirect consequences. The former rules are dynamic because they are assessed after an action is performed, but the later are static because they are evaluated every time the related fluent is false.

Consider the preceding example of the public employee. The following dynamic rules apply to us:

occur(misdemeanor(p),t) → illegal(p,[[t,t + 5]]))
occur(receive_pardon(p),t) → ¬illegal(p,[[t,∞]])
occur(bad_grade(p),t) → ¬good_employee(p,[[t,∞]])
occur(good_grade(p),t) → good_employee(p,[[t,∞]])

The initial situation is:

S0 = {¬receive_bonus(p,[[0,∞]]),
receive_salary(p,[[0,∞]]),
¬suspended(p,[[0,∞]]),
¬good_employee(p,[[0,∞]]),
¬illegal(p,[[0,∞]])}

Assume that the following activities occur at the following time points, with time beginning at zero and temporal granularity of months. Assume that the activity is now being carried out.

occur(misdemeanor(p),2)

The new state is

S1 = {¬receive_bonus(p,[[0,∞]]),
receive_salary(p,[[0,∞]]),
¬suspended(p,[[0,∞]]),
¬good_employee(p,[[0,∞]]),
illegal(p,[[2,7]]),¬illegal(p,[[7,∞]])}

→ S1 does not satisfy the following integrity requirement (at time points in [2,7]):

illegal(p,t1) → suspended(p,t1)

However, the integrity requirement is enforced by a set of static rules.

illegal(p,[[2,7]]) → suspended(p,[[2,7]])

Following its completion, the following information is added to the situation:

suspended(p,[[2,7]]),¬suspended(p,[[7,∞]])

The following integrity restriction is not met in the new situation:

$$\text{suspended}(p,t_1) \rightarrow \neg\text{receive\_salary}(p,t_1)$$

However, because of the static rule,

$$\text{suspended}(p,[[2,7]] \rightarrow \neg\text{receive\_salary}(p,[[2,7]])$$

the final situation is →

$$\begin{gathered} S2 = \{\neg\text{receive\_bonus}(p,[[0,\infty]]), \\ \neg\text{receive\_salary}(p,[[2,7]]), \\ \text{receive\_salary}(p,[[7,\infty]]), \\ \text{suspended}(p,[[2,7]]),\neg\text{suspended}(p,[[7,\infty]]), \\ \neg\text{good\_employee}(p,[[0,\infty]]), \\ \text{illegal}(p,[[2,7]]),\neg\text{illegal}(p,[[7,\infty]])\} \end{gathered}$$

The technique described above is used to generate the collection of static rules.

3.8 Static rule Generation Algorithm

The static Rules Class is the one that contains the most of the properties, variables and methods of our Java code. At first, we import the packages of the other PelletPackage classes (i.e. the PelletPackage.fluent and the PelletPackage.sections), the one with the arrayLists and the packages for the input/output and the java language system. In this class the five steps of the Algorithm that we are based on are mainly implemented. It is, also, quotable to mention that at this point we also have methods that print some results and tables in order to test it and try a beta edition. Finally, this class has also the main function of our program that calls all the functions needed to be compiled and run, as an integraded make-all file.

1. Transform each integrity constraint in its CNF form. Now each integrity constraint has the form $C_1 \wedge C_2 \wedge C_3 \ldots\ldots \wedge C_n$, where each C_i is a disjunction of all fluents.
2. Set R = { False $\rightarrow$ f, False $\rightarrow \neg f : for\ each\ fluent\ f\}$
3. For each I from 1 to n do: assume $C_i = f_1 \vee \ldots. \vee f_m$

```
   For each j from 1 to m do
       For each k from 1 to m, and $k \neq j$, do
           If $(f_i, f_k) \in I$ then
               R= R U ( $\neg\ f_j\ \ causes\ f_k\ if\ \ \wedge\ \neg f_l\ \neq j, k$.
```

4. For each fluent f_k the rules in R have the following form

$$\boxed{\bigwedge f_i \text{ causes } f_k \text{ if } \Phi, \bigwedge f'_i \text{ causes } \neg\ f_k \text{ if } \Phi'}$$

We change the static rules from G $\rightarrow f_k$, K $\rightarrow \neg f_k$

to $(G \vee (\bigwedge\ f_i \bigwedge\ \Phi)) \rightarrow f_k$,

$(K \vee (\bigwedge f_i \wedge \Phi'\ )) \rightarrow \neg f_k$.

5. At time moment t, for each static rule $G^S (t, t_1)$, K$\rightarrow$ f do

   (a) Let $G^s = G1 \vee \ldots. \vee G_{s\_n}$

   (b) For each j from 1 to s_n do

      (i) Let G_j = f1 ([..]) $\vee \ldots. \vee f_n$ ([..])

```
        (A) For each fluent fi (L) (s.t fi (L) ∈ $G_j$) take the first element [t´,
            t´´] of the list L.
        (B) If t´>t then G is false and terminates.
        (C) else ti = t´´- t and remove [t´, t´´] from L
(ii)    Let $t_{min}$ = min ($t_1$, ….$ts_n$)
(iii)   Replace Gi with Gi (t, t + tmin)
```

3.9 Classes

The staticRules Class is the one that contains the most of the properties, variables and methods of our Java code. At first, we import the packages of the other PelletPackage classes (i.e., the PelletPackage.fluent and the PelletPackage.sections), the one with the arrayLists and the packages for the input/output and the java language system. In this class the five steps of the Algorithm that we are based on are mainly implemented. It is, also, quotable to mention that at this point we also have methods that print some results and tables in order to test it and try a beta edition. Finally, this class has also the main function of our program that calls all the functions needed to be compiled and run, as an integraded make-all file.

In this Class we find the single-array variables :

• Static fluent [] *nameOftheVariable

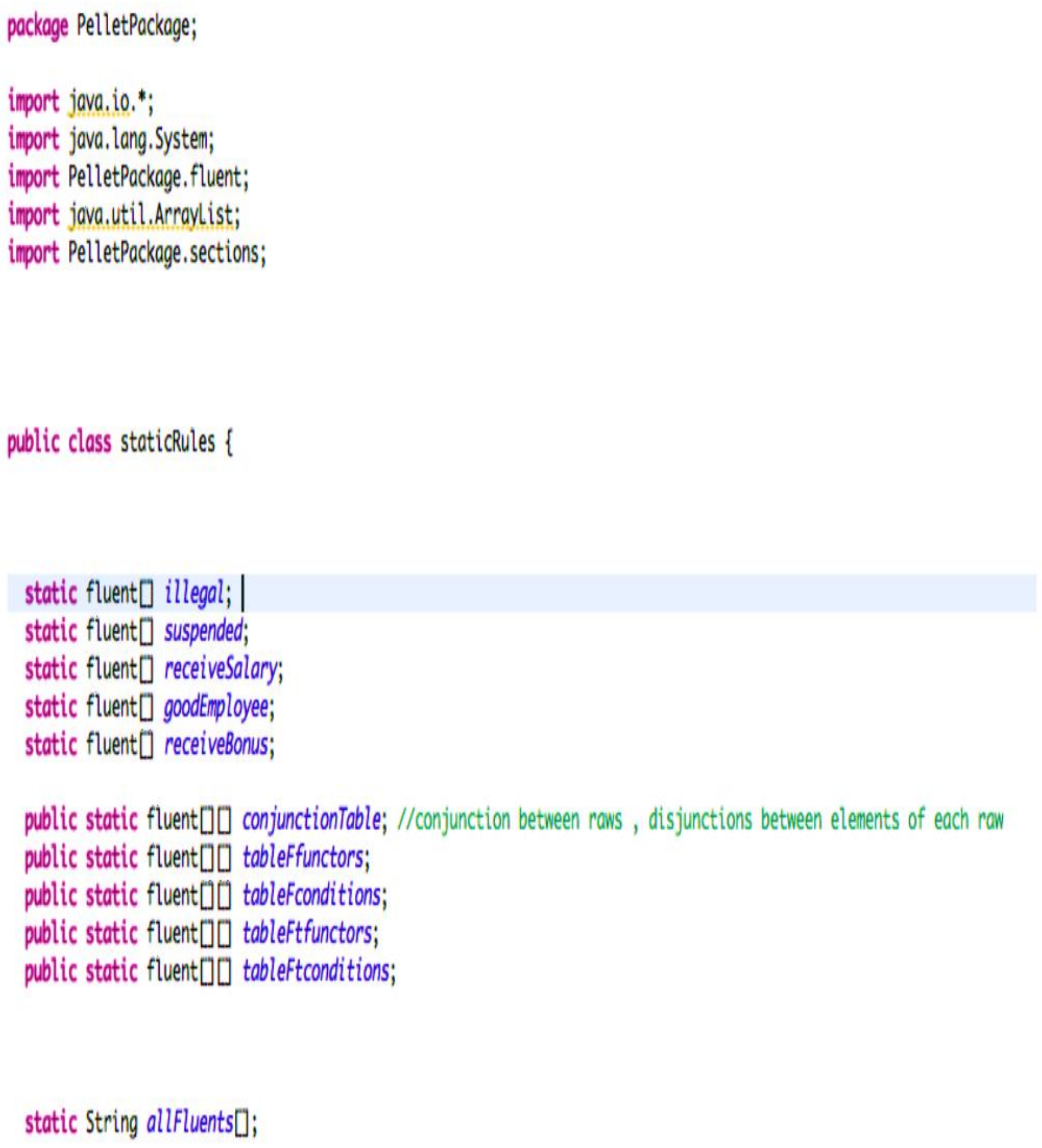
We also find the double-array variables :

→ Public static fluent [] [] conjunctionTable
→ Public static fluent [] [] tableFfunctors
→ Public static fluent [] [] tableFconditions
→ Public static fluent [] [] tableFtfunctors
→ Public static fluent [] [] tableFtconditions

Other futures that we find are the properties – methods of these Class :

- → public static void printConjunctionTable
- → public static void printTableF
- → public static String [] initializeFluentNames
- → public static fluent [][] initializeExample
- → public static fluent [][] step2table
- → public static void checkDoubles
- → public static fluent [][] step4tableFfunctors
- → public static fluent [][] step4tableFconditions
- → public static fluent [][] step4tableFtfunctors
- → public static fluent [][] step4tableFtconditions
- → public static fluent[][] step4tableG
- → public static fluent[][] step4tableK
- → public static fluent[][] copyFluentTable
- → public static fluent copyFluentElement
- → public static sections setGipair
- → public static sections [] setGi
- → public static sections setUnionPair
- → public static sections setUnionG
- → public static rulesIOalgorithm
- → public static void main(String[] args

All the variables, the properties and the methods of the staticClass will be described in details at the upcoming subsection and the upcoming figure presents our staticRules class' public global variables.



```
package PelletPackage;

import java.io.*;
import java.lang.System;
import PelletPackage.fluent;
import java.util.ArrayList;
import PelletPackage.sections;



public class staticRules {



  static fluent[] illegal;
  static fluent[] suspended;
  static fluent[] receiveSalary;
  static fluent[] goodEmployee;
  static fluent[] receiveBonus;

  public static fluent[][] conjunctionTable; //conjunction between raws , disjunctions between elements of each raw
  public static fluent[][] tableFfunctors;
  public static fluent[][] tableFconditions;
  public static fluent[][] tableFtfunctors;
  public static fluent[][] tableFtconditions;



  static String allFluents[];
```

Figure 3: Variables.

3.10 Variables

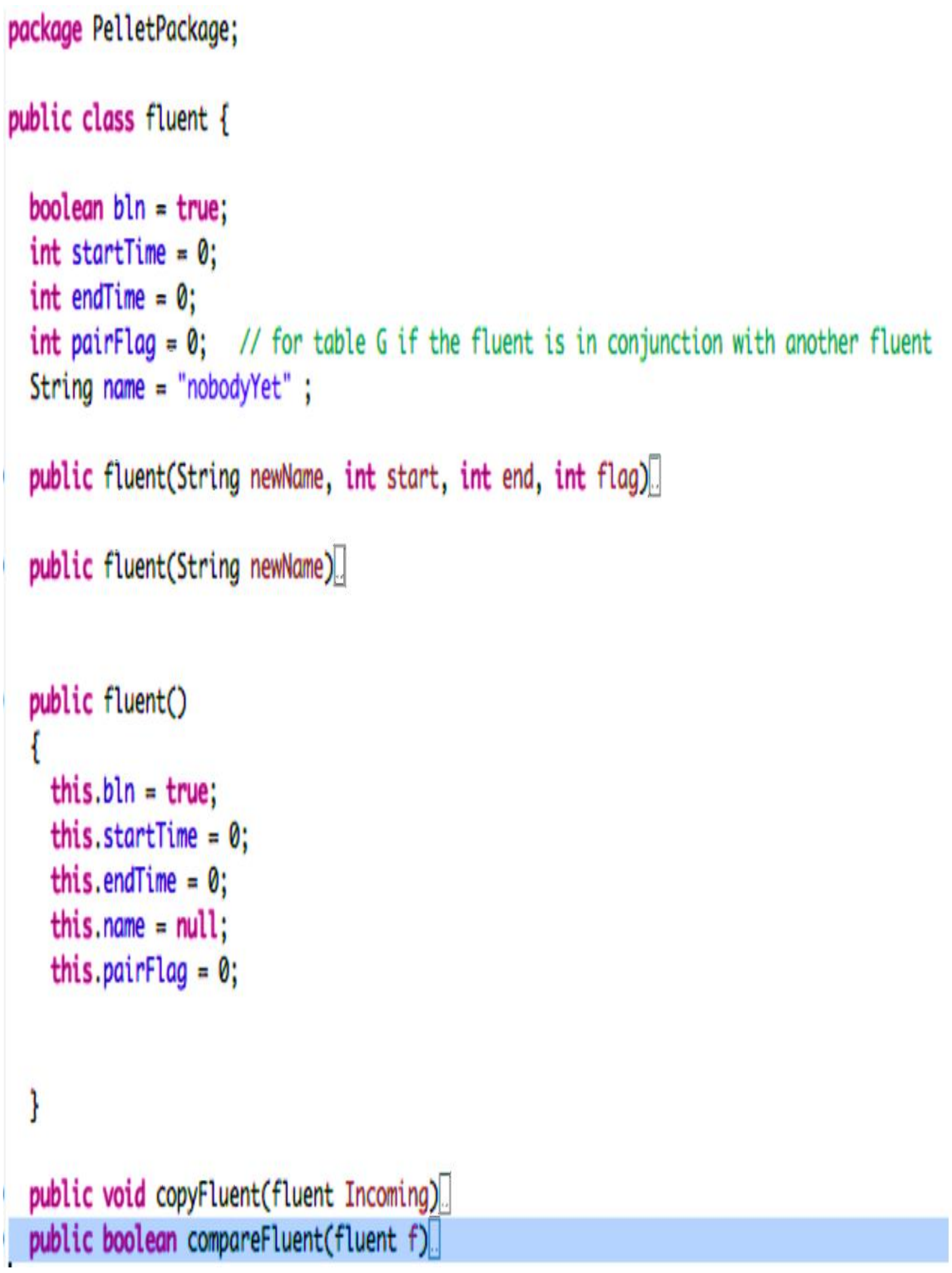
In this class we implement an interesting point of our framework development. In fact, it is an ontology that represent all the atrributes and the functions needed of any fluent that will be imported or exported from our database schema. There are many different Contructors and variables :

- boolean bln
- int startTime
- int endTime
- int pairFlag
- String name

This class generates also two different methods when called :

- public void copyFluent
- public boolean compareFluent

All the variables, the properties and the methods of the fluent will be described in details at the upcoming subsection and the upcoming figure presents our fluent class' public global variables.



```
package PelletPackage;

public class fluent {

  boolean bln = true;
  int startTime = 0;
  int endTime = 0;
  int pairFlag = 0;   // for table G if the fluent is in conjunction with another fluent
  String name = "nobodyYet" ;

  public fluent(String newName, int start, int end, int flag)

  public fluent(String newName)


  public fluent()
  {
    this.bln = true;
    this.startTime = 0;
    this.endTime = 0;
    this.name = null;
    this.pairFlag = 0;


  }

  public void copyFluent(fluent Incoming)
  public boolean compareFluent(fluent f)
```

Figure 4: Fluents class.

3.11 Sections

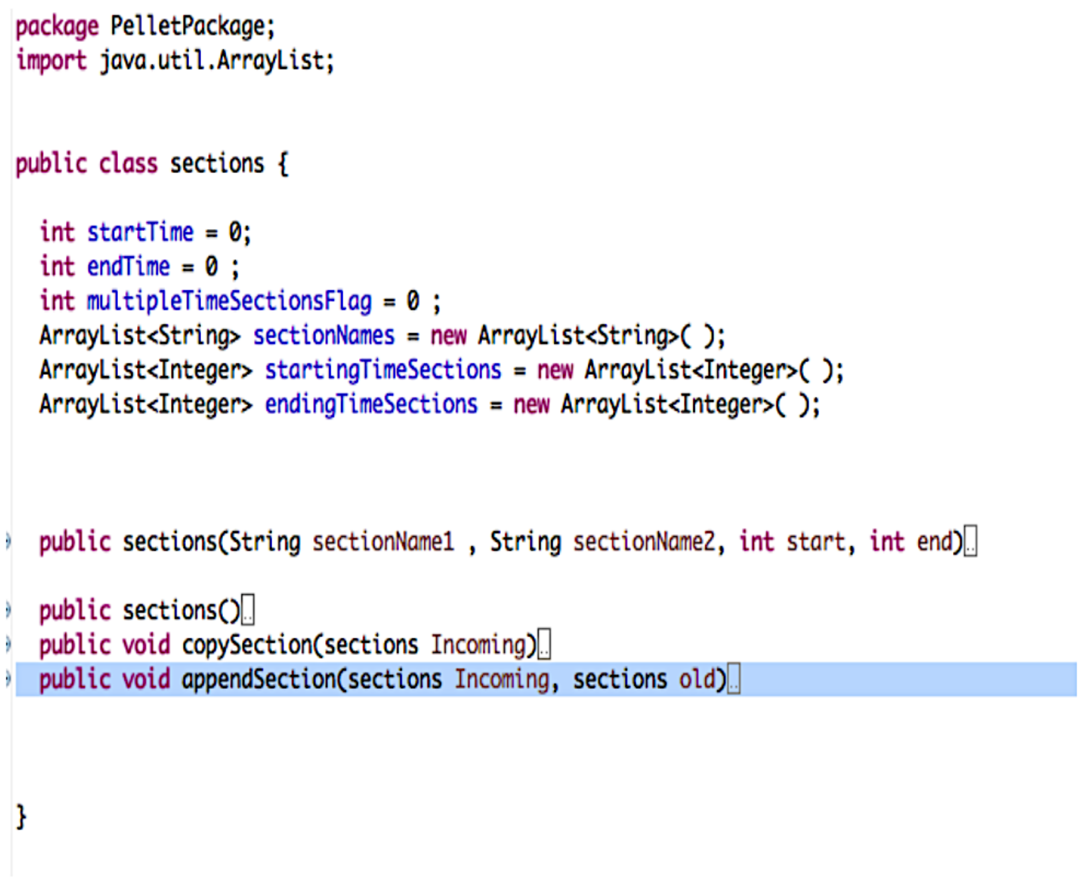
The sections is the third major class of our project. It implements arrays for creating the proper timeslots for each and every fluent. It supports a quick one time slot entity with the variables given, either a multi-timeslot one with the arrays. Variables found in this class are :

- Int startTime
- Int endTime
- Int multipletimeSectionsFlag

Except the variables, the class utilizes also one constructor and the methods :

- Public void copySection
- Public void appendSection

All the variables, the properties and the methods of the sections will be described in details at the upcoming subsection and the upcoming figure presents our sections class' public global variables.



```java
package PelletPackage;
import java.util.ArrayList;


public class sections {

  int startTime = 0;
  int endTime = 0 ;
  int multipleTimeSectionsFlag = 0 ;
  ArrayList<String> sectionNames = new ArrayList<String>( );
  ArrayList<Integer> startingTimeSections = new ArrayList<Integer>( );
  ArrayList<Integer> endingTimeSections = new ArrayList<Integer>( );



  public sections(String sectionName1 , String sectionName2, int start, int end)

  public sections()
  public void copySection(sections Incoming)
  public void appendSection(sections Incoming, sections old)



}
```

Figure 5: Secions class.

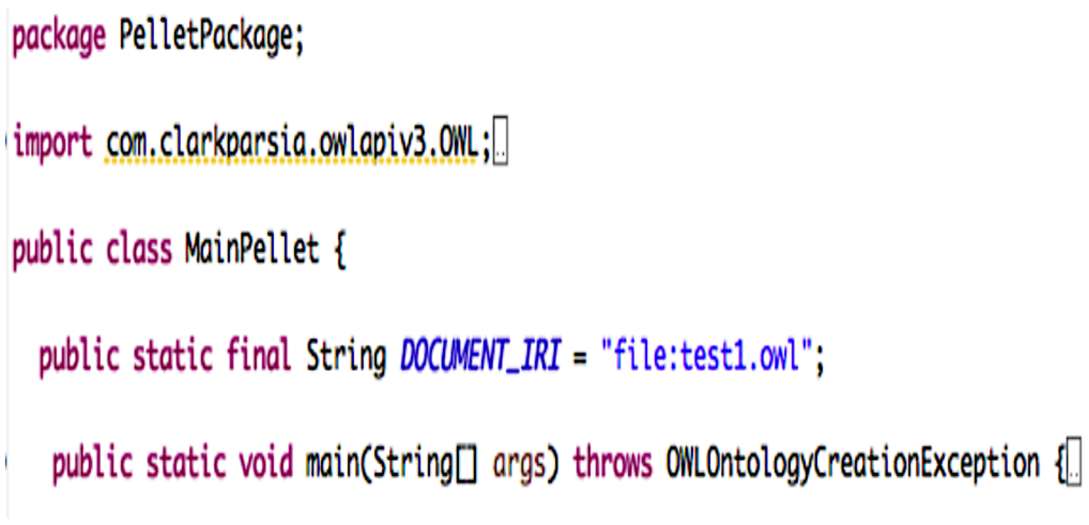
We have already described the three important classes that are called in order to compile the code for algorithm. On the other hand, the mainPellet Class is the one that is used for the import and export of the data, which will be the input and output respectively of our main algorithm. There is only one variable :

- Public static finale string

And only three functions

- Public static void main
- Void sparqInsertTurtle

- Void sparqDeleteTurtle



```
package PelletPackage;

import com.clarkparsia.owlapiv3.OWL;

public class MainPellet {

  public static final String DOCUMENT_IRI = "file:test1.owl";

   public static void main(String[] args) throws OWLOntologyCreationException {
```

Figure 6: MainPallet class.

3.12 Arrays

→ **illegal**

This is an array of variables that was created for the test of the algorithm. In this array we imported the values of database schema under the entity “illegal”.

→ **suspended**

This is an array of variables that was created for the test of the algorithm. In this array we imported the values of database schema under the entity “suspended”.

→ **receiveSalary**

This is an array of variables that was created for the test of the algorithm. In this array we imported the values of database schema under the entity “receiveSalary”.

→ **goodEmploy**

This is an array of variables that was created for the test of the algorithm. In this array we imported the values of database schema under the entity “goodEmploy”.

→ **receiveBonus**

This is an array of variables that was created for the test of the algorithm. In this array we imported the values of database schema under the entity “receiveBonus”.

3.12.1 Double arrays

→ **allFluents[];**

This is an array of variables where we imported the names of all the fluents from the database schema. The names of the fluents that we need for the new time intervals.

→ **conjunctionTable**

This is an array of variables which is implemented in order to creat a series of conjunctions. It means that the elements of the array are in a raw connected with a conjunction between them.

→ **tableFfunctors**

This array implements a table of the elements that are the factors of the Φ of the fourth step of the algorithm that we were based on during the integration of our algorithm. Factors of the Φ is each and every of the elements that are a part of the Φ.

→ **tableFconditions**

This array implements a table of the elements that are the conditions of the Φ of the fourth step of the algorithm that we were based on during the integration of our program. Conditions of the Φ is all the conditions that must exist in order the statement of the Φ to be a true value.

→ **tableFtfunctors**

This array implements a table of the elements that are the factors of the Φ’ of the fourth step of the algorithm that we were based on during the integration of our code. Factors of the Φ’ is all the elements that create the Φ’.

→ **tableFtconditions**

This array implements a table of the elements that are the conditions of the Φ’ of the fourth step of the algorithm that we were based on during the integration of our program. Conditions of the Φ’ is all the conditions that must exist in order the statement of the Φ’ to be tru.

4. Methods of Implementation

4.1 Java programming language

Java is a language created by James Gosling in 1991, where he worked in Sun Microsystems and was made by the need for a new tool for software development in micro devices. Java's initial name was Oak and was inspired by an oak that was outside James's office where he passed most of his time. Later for copyright reasons took this name inspired by Java Coffee where his beloved coffee was also. The tools of that time were C and C ++ and then resulted in the conclusion that existing languages could not meet the needs of that work. So, as it was reasonable Java was built on standards and syntax of C / C ++, so that it seems familiar and easy to developers of the time.

The first version of Java appeared from Sun Microsystems in 1996 and promised that he could run on any platform at no extra cost, which made her quite popular and easy to use. To do this and be understood by any system of independent processor and operating system, a virtual machine set Her work was to create a series of .Class files, where the Java source code is getting when compiled. So, when it comes to running on a machine, Java Virtual Machine that is installed on it, will read them. Class files and will make the translation of machine language that the processor and operating system supports.

Although the virtual machine offers all these advantages, Java was initially slower than other languages such as popular for C ++ season. Measurements made have shown that C ++ is several times faster than Java. This becomes continuous improvements and efforts from Oracle to optimize the virtual machine. In addition to establishing Jit (Just In Time) compilers, which convert the Byte code directly into a machine language, the speed difference from C ++ has greatly reduced.

The advantage of the language is the objective that offers and is based on the use of objects. Objects are Information Fields, Projection and Processing Method and belong to classes, declared by a type and a visibility converter. There are four visibility converters, PUBLIC, PROTECT, PRIVATE, Package-Private. PUBLICs are visible in all application classes. Protected are visible from classes of the same pack if it is extending in class and out of package. Private is visible only from the same class and finally Package-Private are visible from classes of the same package. It follows an object example for the class below.

4.2 SQL programming language

The SQL programming language has been developed for the first time in the 1970s by IBM Raymond Boyce and Donald Chamberlin researchers. The programming language, known as Sequel, was created after publishing the Edgar Frank Todd [19]. "A relational data model for large common data banks" in 1970 [19]. At his work, Todd suggested that all data is represented in a database in the form of relationships. Relied on this theory that Boyce and O Chamberlin suggested SQL. In the book "Oracle Quick Guides (Cornelio Books 2013)", the Author Malcolm Coxall writes that the original

SQL version was designed to manipulate and recover the data stored in the original IBM relational base management systems, known as "System R." Only a few years later, SQL language was publicly divided. In 1979, a Company named Relational Software, which later became Oracle, commercially released its own version of SQL language called Oracle V2. Since then, the US National Institute of Standards (ANSI) and the International Standardization Organization considered that SQL language is the formal language in relational database communication. While large SQL vendors modify the language in their wishes, most of them base their SQL programs from the issued version approved by ANSI [20].

SQL, which represents the structured language language, is a programming language used to communicate and handle databases. In order to exploit most of the data they collect, many businesses have to invest in SQL. SQL programs are created by businesses and other organizations as How to access and handle information and data stored in their databases, as well as to create and modify new tables. In order to fully understand SQL, it is important to be known first exactly what is a database. According to Microsoft, a database is a tool for collecting and organizing information [19]. Databases can store information about people, products, orders or anything else. Many databases begin on a text editor or spreadsheet, but as they grow, many businesses will be useful to carry them into a database that It is created by a database management system.

To control information on these databases is used SQL, which allows users to recover the specific data they are looking for when they need it. While it is a simple programming language, SQL is also very strong. A database says that SQL can enter data into database tables, modify data to existing database tables and deletes data from SQL database tables. In addition, SQL can modify the database structure itself by creating, modifying and deleting tables and other database items [20].

4.2.1 Using SQL to convert static rules

A fluent of the form f_1([[a,b],[c,d],[e,f]]) is kept in a table as triples (ID, timestamp start, timestamp end) in a relational temporal database. Number is the type of ID, whereas date or timestamp is the type of timestamp start and timestamp end. All of the fields make up the table's key. The fluent f_1([[a,b],[c,d],[e,f]]) will be stored in three rows (triples) in this assumption: (1,a,b), (1,c,d), (1,e,f) (1,e,f). We must identify the intersection of the time intervals in which the fluents f_1 and f_2 are true in order to translate and execute a rule of the type f_1(...) f_2(...) f_3(...) in SQL.

To do so, it is necessary to compare all the rows that refer to f_1 to all the rows that refer to f_2 and locate all the intersections of the time periods. For example, if there are two rows (1,'26-7-2007','31-7-2007') and (2,'29-7-2007','6-8-2007'), the time interval ('29-7-2007','31-7-2007') intersects. This indicates that row (3,'29-7-2007','31-7-2007') must be added to the table. If we enter the row (1, '4-8-2007','14-8-2007') into the table, we get the following rows: (1, '4-8-2007','14-8-2007') and (2,'29- 7-2007','6-8-2007'). This indicates that, as a result of the execution of the aforementioned static rule, we must insert the row (3,'4-8-2007','6-8-2007') immediately after inserting the row (1,'4-8-2007','14-8-2007').

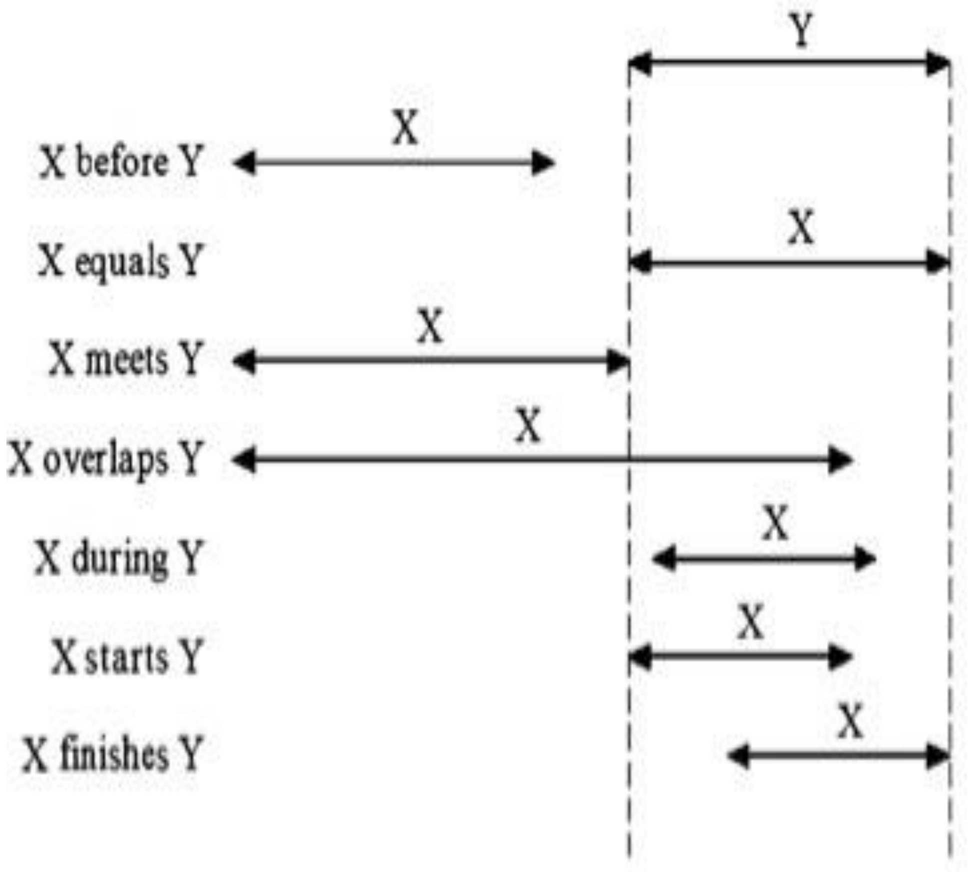


Figure 7: Temporal Allen relations.

The intersection of two temporal intervals is computed with the help of the following algorithm:

```
1. Take the first pair (fi,fj) of fluents in the left side of the
static rule.
2.      For each rows row11,row22 s.t. row1.ID = i and
row2.ID = j do
3.              s1 := row1.START;
                e1 := row1.FINISH;
                s2 := row2.START;
                e2 := row2.FINISH;
                changes := FALSE;
4.      IF s1 < s2 AND e1 >= s2 AND e1 <= e2 THEN
                resSt := s2;
                resEnd := e1;
                changes := TRUE;
5.      ELSIF s1 >= s2 AND s1 <= e2 AND e1 >= s2 AND
                e1 <= e2 THEN
                resSt := s1;
                resEnd := e1;
                changes := TRUE;
6.      ELSIF s1 <= s2 AND e2 <= e1 THEN
                resSt := s2;
                resEnd := e2;
                changes := TRUE;
7.      ELSIF s1 >= s2 AND s1 <= e2 AND e1 > e2 THEN
                resSt := s1;
                resEnd := e2;
                changes := TRUE;
        END IF;
8.      IF changes = TRUE THEN INSERT INTO TABLENAME_temporary_table
                VALUES ( currTempTableId, resSt, resEnd );
        END IF;
```

4.3 Sparql Queries

At this section the sparql queries for the data given are presented

Func	Start	End
F1	0	5
F2	2	8
F3	10	17
F4	3	4
F5	6	18
F6	12	16
F7	17	20
F8	30	34
F9	21	25
F10	19	28

Table 1: Func dataset.

After the application of the algorithm above sparql queries that are generated are the following. The schema used is at the appendix:

if $E_2 =< S_1$ -> false (1)

if $S_2 =< S_1$ && $S_1 < E_2$ && $E_2 =< E_1$ (2) (returns union S1, E2]

For F3 & F5 the equation above returns 17-18

```
SELECT ?S1, ?E2
WHERE
```

```
  {


    ?S1 untitled-ontology-64;F3 ? FStart .
    ?S2 untitled-ontology-64;F5 ? FStart .
    ?E1 untitled-ontology-64;F3 ? FEnd .
    ?E2 untitled-ontology-64;F5 ? Fend .


    FILTER ( ?S1 >= ?S2 ) .
    FILTER ( ?S1 < ?E2 ) .
    FILTER ( ?E1 >= ?E2 ) .
  }
```

if $S_2 =< S_1$ && $E_2 > E_1$

For F9 & F10 the equation above returns 21-25

```
SELECT ?S2, ?E3
WHERE
  {


    ?S1 untitled-ontology-64;F9 ? FStart
    ?S2 untitled-ontology-64;F9 ? FStart
    ?E1 untitled-ontology-64;F10 ? FEnd
    ?E2 untitled-ontology-64;F10 ? FEnd


    FILTER ( ?S1 >= ?S2 ) .
    FILTER ( ?E1 < ?E2 ) .


  }
```

if S_2 >= S_1 && E_2 =< E_1

For F3 & F6 the equation above returns 12-16

```
SELECT ?S1, ?E1
WHERE
  {


    ?S1 untitled-ontology-64;F3 ? FStart
    ?S2 untitled-ontology-64;F3 ? FStart
    ?E1 untitled-ontology-64;F6 ? FEnd
    ?E2 untitled-ontology-64;F6 ? FEnd



    FILTER ( ?S2 >= ?S1 ) .
    FILTER ( ?E1 >= ?E2 ) .


  }
```

if S_2 >= S_1 && S_2 =< E_1 && E_2 >=E_1

For F1 & F2 the equation above returns 2-5

```
SELECT ?S2, ?E1
WHERE
  {
    ?S1 untitled-ontology-64;F1 ? FStart
    ?S2 untitled-ontology-64;F1 ? FStart
    ?E1 untitled-ontology-64;F2 ? FEnd
    ?E2 untitled-ontology-64;F2 ? FEnd

    FILTER ( ?S2 >= ?S1 ) .
    FILTER ( ?E2 >= ?E1 ) .
    FILTER ( ?E1 >= ?S2 ) .
  }
```

if S_2 >= E_1 -> false

Conclusions

At the section above the ramification problem was presented in the context of relational temporal databases in this study. In particular, the solution proposed from was described. The proposed algorithm is based on the following important ideas:

- to broaden the scenario calculus.
- to develop appropriate timing, action, and situational correspondences.
- to employ dynamic rules to record actions' direct consequences and static rules to capture actions' indirect effects

Upon the completion of the literature, static rules were implemented in sparql query language with the help of a schema that was generated for this purpose. In the future all queries generated may be included inside a double for loop like the tool described in.

In the implementation and design of this dissertation Work, achieved a better understanding of more complex bases. That is, a base where its operation is not only the storage of useful data but also the interaction of the tables between them. Difficulty has been dealt with compliance with the integrity constraints that had been taken from the outset.

This has led to the understanding of new technologies as well as research into technologies already known for the implementation team. Spatial-time databases have plenty of applications, one of the important applications you meet but And it will meet even more in the future is in Models of Intelligence Craftsman. Artificial intelligence is a branch of science or rather advanced science that deals with how intelligence can be applied. But after applying the next important aspect that needs to be addressed is how the data to be saved, so we need databases. The database is basically a data group that stores the data, both in sequential and non-sequential format. The data is an integral part of artificial intelligence. One of the challenges with training models and deep learning models are the huge volume of data and processing power you need to Train a neural network, for example, in complicated identification of patterns in areas such as image classification or physical language processing.

Therefore, artificial intelligence databases begin to emit the market as a way to optimize the learning and training process of artificial intelligence for businesses.

expected in the future Even more round publications from this area as this model can contribute many to the technologies of the future.

SQL Server provides us with a variety of ways to convert the SQL Server data type of an expression to another type of data. Starting from SQL Server 2012, three new features are entered, which can also be used to convert data types, in addition to data conversion exemption mechanism that makes the data conversion process more optimal.

Available mandates for execution are: TRANSPORT, CONVERSATION AND ANALYSIS. In this article, we briefly described the differences between these three functions, how to use and, finally, a comparison of their performance. In our demonstration, running using three different types of data conversions, it can be found that CAST is the fastest function that can be used to convert the data type of the supplied value and the Parse function is the slower.

References

[1] McCarthy, J., & Hayes, P. (1969). Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer & Donald Michie (eds.), _Machine Intelligence 4_. Edinburgh University Press. pp. 463-502.

[2] Ginsberg M., & Smith D., (1998), “Reasoning about action I: A possible worlds approach Artificial Intelligence”, pp 165-195

[3] Winslett, M., (1988), “Reasoning about action using a possible models approach”,In Proceeding of the AAAI National Conference on Artifical Intelligence, Saint Paul, MN,pp 89-93

[4] Lifshitz, V. (1991), Towards a metatheory of action. In J.F. Allen, R. Fikes, and E. Sandewall, editors, Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning, Cambridge, MA, pp 376-386

[5] Lifshitz V., (1990), Frames in the space of situations, Artificial Intelligence, pp 46:365-376

[6] Lifshitz, L. (1993) Restricted monotonicity. In Proceedings of the AAAI National Conference on Artifical Intelligence, , Washington DC, pp 432-437

[7] McCain, N. & Hudson, T. (1995) “A causal theory of ramifications and qualifications”, In C. S. Mellish, editor, Proceedings of the International Joint Conference on Artifical Intelligence (IJCAI), Montreal, Canada, pp 1978-1984

[8] McCain, N, Turner, H. (1995) “A causal theory of ramifications and qualifications”, In: Proceedings of IJCAI-95, pp 1978–1984

[9] McCarthy J, Hayes PJ (1969), “Some philosophical problem from the standpoint of artificial intelligence” In: Machine intelligence, vol 4, pp 463–502

[10] Papadakis N., Plexousakis, D., (2003) “Action with duration and constraints: the ramification problem in temporal databases”. Int J Artif Intell Tools, 12(3), pp 315–353

[11]“A Timeline of Database History.” [Online]. Available: https://www.quickbase.com/articles/timeline-of-database-history.

[12] D. R. Brackenridge, “United States Patent,” vol. 2, no. 12, 1992.

[13] Craig S. Mullins and S. Christiansen, “database management system (DBMS).” [Online]. Available: https://searchsqlserver.techtarget.com/definition/databasemanagement-system.

[14] Essays, “The Evolution Of Database Management System,” 23rd March, 2015, 2015. [Online]. Available: https://www.ukessays.com/essays/informationtechnology/the-evolution-of-database-management-system-informationtechnology-essay.php.

[15] M. Chapple, “What Is a Database Management System (DBMS)?,” July 03, 2018, 2018. [Online]. Available: https://www.lifewire.com/databasemanagement-system-1019609. [15] Θ. Τζουραμάνης, “Μέθοδοι προσπέλασης και επεξεργασίας ερωτήσεων σε χρονικές και χωροχρονικές βάσεις δεδομένων,” 2002. [Online]. Available: 40 | P a g e http://thesis.ekt.gr/thesisBookReader/id/20183#page/1/mode/2up.

[16] S. Farooq, “SPATIAL DATABASES Concept, Design and Management.” [Online]. Available: http://www.geol-amu.org/notes/m14b-4-4.htm.

[17] Techopedia, “Spatial Database.” [Online]. Available: https://www.techopedia.com/definition/17287/spatial-database.

[18] A. Neumann, Encyclopedia of GIS. 2017.

[19] J. Biscobing, “relational database.” [Online]. Available: https://searchdatamanagement.techtarget.com/definition/relational-database.

[20] C. Brooks, “What is SQL?,” January 21, 2014, 2014. [Online]

[21] Papadakis N., Plexousakis, D., Christodoulou Y., (2012) “The ramification problem in temporal databases: a solution implemented in SQL”. AppI Inell 36: 749-767.

Appendix

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [

   <!ENTITY owl "http://www.w3.org/2002/07/owl#" >

   <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >

   <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >

   <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >

   <!ENTITY untitled-ontology-64
"http://www.semanticweb.org/fanis/ontologies/2021/7/untitled-ontology-64#" >

]>



<rdf:RDF xmlns="http://www.semanticweb.org/fanis/ontologies/2021/7/untitled-
ontology-64#"

    xml:base="http://www.semanticweb.org/fanis/ontologies/2021/7/untitled-
ontology-64"

    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"

    xmlns:untitled-ontology-
64="http://www.semanticweb.org/fanis/ontologies/2021/7/untitled-ontology-64#"

    xmlns:owl="http://www.w3.org/2002/07/owl#"

    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"

    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

   <owl:Ontology
rdf:about="http://www.semanticweb.org/fanis/ontologies/2021/7/untitled-ontology-
64"/>
```

```
    <!--
    ///////////////////////////////////////////////////////////////////////////////////////
    //
    // Data properties
    //
    ///////////////////////////////////////////////////////////////////////////////////////
     -->

    <!-- http://www.semanticweb.org/fanis/ontologies/2021/7/untitled-ontology-
64#FEnd -->

    <owl:DatatypeProperty rdf:about="&untitled-ontology-64;FEnd"/>
        <!-- http://www.semanticweb.org/fanis/ontologies/2021/7/untitled-ontology-
64#FStart -->

    <owl:DatatypeProperty rdf:about="&untitled-ontology-64;FStart"/>
        <!--
    ///////////////////////////////////////////////////////////////////////////////////////
    //
    // Classes
    //
    ///////////////////////////////////////////////////////////////////////////////////////
     -->
    <!-- http://www.semanticweb.org/fanis/ontologies/2021/7/untitled-ontology-
64#Func -->

    <owl:Class rdf:about="&untitled-ontology-64;Func"/>

        <!--

    ///////////////////////////////////////////////////////////////////////////////////////

    //

    // Individuals

    //

    ///////////////////////////////////////////////////////////////////////////////////////

     -->

     <!-- http://www.semanticweb.org/fanis/ontologies/2021/7/untitled-ontology-64#F1
-->

    <owl:NamedIndividual rdf:about="&untitled-ontology-64;F1">

        <rdf:type rdf:resource="&untitled-ontology-64;Func"/>

        <FStart rdf:datatype="&xsd;integer">0</FStart>

        <FEnd rdf:datatype="&xsd;integer">5</FEnd>

    </owl:NamedIndividual>

        <!-- http://www.semanticweb.org/fanis/ontologies/2021/7/untitled-ontology-
64#F10 -->

    <owl:NamedIndividual rdf:about="&untitled-ontology-64;F10">

        <rdf:type rdf:resource="&untitled-ontology-64;Func"/>
        <FStart rdf:datatype="&xsd;integer">19</FStart>

        <FEnd rdf:datatype="&xsd;integer">28</FEnd>

    </owl:NamedIndividual>



    <!-- http://www.semanticweb.org/fanis/ontologies/2021/7/untitled-ontology-64#F2
-->

    <owl:NamedIndividual rdf:about="&untitled-ontology-64;F2">

        <rdf:type rdf:resource="&untitled-ontology-64;Func"/>

        <FStart rdf:datatype="&xsd;integer">2</FStart>

        <FEnd rdf:datatype="&xsd;integer">8</FEnd>

    </owl:NamedIndividual>

        <!-- http://www.semanticweb.org/fanis/ontologies/2021/7/untitled-ontology-
64#F3 -->

    <owl:NamedIndividual rdf:about="&untitled-ontology-64;F3">

        <rdf:type rdf:resource="&untitled-ontology-64;Func"/>

        <FStart rdf:datatype="&xsd;integer">10</FStart>

        <FEnd rdf:datatype="&xsd;integer">17</FEnd>

    </owl:NamedIndividual>

       <!-- http://www.semanticweb.org/fanis/ontologies/2021/7/untitled-ontology-
64#F4 -->

    <owl:NamedIndividual rdf:about="&untitled-ontology-64;F4">

        <rdf:type rdf:resource="&untitled-ontology-64;Func"/>

        <FStart rdf:datatype="&xsd;integer">3</FStart>

        <FEnd rdf:datatype="&xsd;integer">4</FEnd>
    </owl:NamedIndividual>

        <!-- http://www.semanticweb.org/fanis/ontologies/2021/7/untitled-ontology-
64#F5 -->

    <owl:NamedIndividual rdf:about="&untitled-ontology-64;F5">

        <rdf:type rdf:resource="&untitled-ontology-64;Func"/>

        <FEnd rdf:datatype="&xsd;integer">18</FEnd>

        <FStart rdf:datatype="&xsd;integer">6</FStart>

    </owl:NamedIndividual>



    <!-- http://www.semanticweb.org/fanis/ontologies/2021/7/untitled-ontology-64#F6
-->

    <owl:NamedIndividual rdf:about="&untitled-ontology-64;F6">

        <rdf:type rdf:resource="&untitled-ontology-64;Func"/>

        <FStart rdf:datatype="&xsd;integer">12</FStart>

        <FEnd rdf:datatype="&xsd;integer">16</FEnd>

    </owl:NamedIndividual>

        <!-- http://www.semanticweb.org/fanis/ontologies/2021/7/untitled-ontology-
64#F7 -->

    <owl:NamedIndividual rdf:about="&untitled-ontology-64;F7">

        <rdf:type rdf:resource="&untitled-ontology-64;Func"/>

        <FStart rdf:datatype="&xsd;integer">17</FStart>

        <FEnd rdf:datatype="&xsd;integer">20</FEnd>

    </owl:NamedIndividual>
```

```
        <!-- http://www.semanticweb.org/fanis/ontologies/2021/7/untitled-ontology-
64#F8 -->

    <owl:NamedIndividual rdf:about="&untitled-ontology-64;F8">

        <rdf:type rdf:resource="&untitled-ontology-64;Func"/>

        <FStart rdf:datatype="&xsd;integer">30</FStart>

        <FEnd rdf:datatype="&xsd;integer">34</FEnd>

    </owl:NamedIndividual>

        <!-- http://www.semanticweb.org/fanis/ontologies/2021/7/untitled-ontology-
64#F9 -->

    <owl:NamedIndividual rdf:about="&untitled-ontology-64;F9">

        <rdf:type rdf:resource="&untitled-ontology-64;Func"/>

        <FStart rdf:datatype="&xsd;integer">21</FStart>

        <FEnd rdf:datatype="&xsd;integer">25</FEnd>

    </owl:NamedIndividual>

</rdf:RDF>
```