4D-Fluents Plug-In: A Tool for Handling Temporal Ontologies in Protégé

Polyxeni Makri

Department of Electronic and Computer Engineering
Technical University of Crete

Dissertation Thesis Committee:
Euripides G.M Petrakis, Associate Professor (Supervisor)
Stavros Christodoulakis, Professor
Michail G. Lagoudakis, Assistant Professor

2011

Στους γονείς μου,
στον παππού μου Μήτσο,
και στη μνήμη του παππού Θανάση.

Acknowledgments

This thesis would not have been possible without the help of several people who in one way or another, contributed and offered their valuable assistance in the preparation and completion of this study.

First and foremost, my utmost gratitude to my advisor, Associate Professor Euripides G.M Petrakis for his supervision, advise and guidance from the very early stage of this research. I am grateful for his encouragement and precious contribution throughout this study. I would also like to thank him for giving me the opportunity to work on this very interesting field of research.

Also, I would like to thank Professor Stavros Christodoulakis and Assistant Professor Michail G. Lagoudakis who agreed to evaluate my diploma thesis.

Moreover, I would like to thank my laboratory colleagues for their patience and constructive comments.

I would like to thank my friend Ioannis for his enormous help and support all these years. And my friend Eleni for being next to me whenever I needed her.

Finally, my eternal gratitude goes to my parents and my brother, who supported me for as long as they could and believed in me. I will never forget your dedication and support.

Abstract

An ontology describes the concepts and relationships that are important in a particular domain, provides a vocabulary for that domain that helps both people and machines to communicate concisely. Many approaches have been proposed to deal with representing information that evolves in time (e.g., objects in a video) in ontologies including among others, N-ary relations, reification, and recently the 4D-fluents approach with the later being the one we adopt in this work. However, representing the time dimension of concepts that evolve in time in ontologies require that temporal relationships become ternary (from binary). Typically, this is handled by decomposing ternary relationships to a set of binary relations and by using additional new classes to represent their relationships. This introduces additional complexity into the representation (e.g., property restrictions of temporal classes might refer to the additional classes introduced by the representation rather than the classes on which they are originally defined). Writing a temporal ontology using an editor takes time and effort and requires lots of attention to detail. Although there are editors for handling ontologies, with the most popular being the Protégé editor, there is no tool for crafting temporal concepts in ontologies. This is exactly the problem this work is dealing with. In this thesis, we design and implement a Plug-In for the Protégé editor that facilitates the crafting (i.e., creating, editing) of temporal ontologies. Particular emphasis is given to making the Plug-In portable and easy to use by ordinary users of the semantic Web. The Plug-In is realized as a Protégé tab that provides a front-end interface which is easy to use, handles temporal ontologies similarly to static ontologies and does not require that users are familiar with peculiarities of the underlying representation of temporal information (the 4D-fluents approach in our case).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Ontologies represent a set of concepts and the relationships between these concepts that describe a particular domain. Issues relating to knowledge representation has been in the center of intensive research activities in artificial intelligence and databases and information systems research. The increasing role of ontologies in information systems design for knowledge representation has been acknowledged many times by the research community and recently with the advent of the Semantic Web [5]. Because of their increasing significance in research for the Semantic Web, ontologies need to be extended to incorporate additional information types which are routinely used in applications of the real world. A particular aspect of knowledge that needs to be interpreted by ontologies is temporal information.

Several representation languages are defined for the Semantic Web, the most important of them are referred to as the OWL-family [13] of ontology languages (OWL-Full, OWL-DL and OWL-Lite) for ontology building and knowledge representation. OWL-Time [7] has been developed that is very simple and provides a vocabulary for expressing the most needed time-related facts. Dealing with information that changes over time is a critical problem in Knowledge Representation (KR). Representation languages such as OWL, RDF (which are based on description logics), are all based on binary relations. Binary relations simply connect two instances (e.g., the employee with the company) without any temporal information. Nevertheless, time representation using OWL is feasible, although complicated [1].

1.1 Problem Definition

A critical problem in practical Knowledge Representation (KR) is dealing with information that evolves over time. The OWL-Time [7] temporal ontology describes the temporal content of Web pages and the temporal properties of Web services. Apart from language constructs for the representation of time in ontologies, there is still a need for mechanisms for the representation of the evolution of concepts (events) in time. For example, the fact that a person will go to elementary school, to high school, and then to college or that a company will be established, hire personnel, and develop products which evolve as a result of time, cannot be sufficiently described using existing methods. Languages such

as OWL, RDF etc., are biased towards binary relations making the representation of time complicated and tricky, as temporal relations become ternary with the addition of the temporal property.

It is possible to enhance the capabilities of state of the art information representation over the semantic Web and its support for information analysis and reasoning, by exploiting the time dimension in the information possessed. This can be achieved by adding the concepts of time and change (evolution) in a rich semantics ontology representation enabling context aware information analysis and reasoning based on evolution over time.

Ontologies offer the means for representing high level concepts, their properties and interrelationships. Dynamic or temporal ontologies will in addition enable representation of time evolving information in ontologies through e.g., versioning [12] or the 4-D perdurandist approach [1]. According to this approach all entities are perdurants, making no distinction between endurants (physical objects such as cars, companies, people) and occurants (events such as buying a car). The idea is that each entity is considered to be an event that has a start and an end point. The temporal ontology then becomes more complicated compared to static ontologies (where all properties are directly attached to the static classes) as any temporal properties are attached to temporal classes (i.e. the *TimeSlice* and *TimeInterval* classes) introduced by the representation [13] and are only indirectly associated with the dynamic classes that the property refers to. In turn, the creation of a temporal ontology, or the conversion of a static one to temporal is a complex enterprise.

Ontology editors, such as Protégé are particularly well suited for crafting (creating, editing) static ontologies with binary relations but have no means for facilitating the development of dynamic ontologies (with ternary relationships between classes). As it is common in all known approaches for representing dynamic concepts (such as the N-ary or the 4D-fluents approach) the ternary relationships are decomposed into sets of binary relations and properties holding between classes now refer to properties between classes introduced by the temporal representation. This not only complicates the ontology, but also, requires that the user be familiar with the peculiarities of the temporal representation method adopted. In addition, information such the above cannot be handled directly by an ontology editor such as Protégé (i.e., property restrictions between temporal classes cannot be defined using Protégé). These are exactly the problems this work is dealing with.

1.2 Proposed Solution

We present Plug-In for the Protégé editor (version 4.1 beta, supporting OWL 2.0) that facilitates the crafting of temporal ontologies and the handling of temporal information, such the definition and handling of temporal classes and of temporal properties as well as of the restrictions defined over temporal properties. The tool is implemented as a Tab in Protégé which is portable and easy to use, handles temporal ontologies and static ones and does not require the user be familiar with the peculiarities of the underlying representation of temporal information is ontologies (i.e., the 4D-fluent approach is our case).

The contribution of this work is the development of a tab widget Plug-In in Protégé 4.1 beta, which has the following desirable properties:

1. The design and the implementation of the Plug-In interface in a way consistent with the layout of the default Protégé tabs .

2. The tool is developed in a way easy to use, handles temporal ontologies as static ones, and does not require that the user be familiar with the peculiarities of the underlying representation of temporal information.

3. Supports reasoning over temporal information using the standard Pellet reasoner in Protégé. This is achieved by introducing a set of SWRL rules adding the required reasoning support to the temporal representation. The reasoner implements the method described in [14, 15].

1.3 Thesis outline

Background knowledge and related research are discussed in Chapter 2. A description of ontologies and of the 4-D perdurantist approach to handle time in ontologies is presented. The three first sections of Chapter 3 provide information for the Plug-In interface, like use cases diagrams, activity diagrams and details about the layout and its supported functionality. In section 3.6 the steps of the ontology conversion to temporal are discussed accordingly to 4D-fluents mechanism. In sections 3.7 and 3.8 is presented the structure of our source code and some significant methods we use. Finally, conclusions and issues for further research are discussed in Chapter 4.

Chapter 2

Background and Related Work

2.1 Ontologies

An ontology is a method of representing knowledge (i.e., ideas, facts, things) in a way that defines the relationships and classifications of concepts within a specific domain of knowledge [24]. They are well-suited for describing heterogeneous, distributed and semi-structured information sources that can be found on the Web. By defining shared and common domain theories, ontologies help both people and machines to communicate concisely, supporting the exchange of semantics and not only syntax. In recent years, ontologies been adopted in many business and scientific communities as a way to share, reuse and process domain knowledge. Now they are used in many applications such as scientific knowledge portals, information management and integration systems, electronic commerce, and semantic web services.

2.2 Representation of time in Ontologies

Dealing with information that changes over time is a critical problem in Knowledge Representation (KR). Representation languages such as OWL [3], RDF (description logics) [16], frame-based and object-oriented languages (F-logic) [17] are all based on binary relations. The fact is that binary relations may change over time making the representation of time a difficult matter to deal with, since binary relations simply connect two instances without any temporal information. Apart from language constructs for the representation of time in ontologies, there is a need for mechanism for the representation of the evolution of concepts (events) over time. The principal mechanisms in Semantic Web are Temporal Description logics [10] [11], Versioning [12], Reification [1].

Temporal Description Logics (TDL) extend Description Logics (DL) with additional time representation operators and semantics such as 'until' and'always in the past'. Many TDLs have been proposed [10] [11] with the most expressive of them being undecidable. Contrary to other approaches,

temporal description logics offer additional semantics and reasoning mechanisms and they don't suffer from data redundancy. All other approaches except TDLs require temporal semantics to be defined using an additional set of rules combined with a reasoning mechanism, as we did in this work. TDLs disadvantage is that they require extending OWL to represent time (by introducing additional operators and semantics), while the other approaches can be implemented directly using OWL.

Versioning [12] suggests that the ontology has a different version for every instance of time. When a change takes place, a new version is created. Versioning suffers from several disadvantages: (a) information redundancy, because a new version of the ontology is created when a change occur ,even on single attributes,(b) searching for events, occurred at time instances or during time intervals, requires exhaustive searches in multiple versions of the ontology,(c) it is not clear how the relation between evolving classes is represented. Furthermore, ontology languages such as OWL citeowl are based on binary relations (relations connecting two instances) with no time dimension.

Reification [1] is a general purpose technique for representing N-ary relations using a language such as OWL that permits only binary relations. Specially, an N-ary relation is represented as a new object that has all the arguments of the N-ary relation as attributes. For example if the relation R holds between objects A and B at time t, expressed as R(A,B,t), this is represented in OWL using reification as an object R with attributes A;B and t. Reification suffers from two disadvantages: (a) data redundancy, because a new object is created whenever a temporal relation has to be represented (this is a problem common to all approaches based on non temporal Description Logics such as OWL-DL) and (b) offers limited OWL reasoning capabilities.

4D-fluents (four-dimensionalist)[1] approach shows how temporal information can be represented effectively in OWL and is the one that we use in this work. Notice though that it still suffers from data redundancy. In order to better explain this mechanism, we should briefly describe the 3-D view.

In 3D view, the world is distinguished into two basic categories: the endurants (physical objects such as cars, companies, people) and the occurants (events such as buying a car). Endurants are supposed to exist at all times and have no time dimension, while occurants have temporal parts that exist during the times the entity exists. The main issue with this approach is that the diachronic identity (the identity that determines an entity over time) of endurants is addressed by identifying a set of properties that do not change over time. An entity (endurant) has a set of properties that do not change over time (e.g., a person's DNA) along with a set of properties that do change over time (e.g., the hair's color).

The 4D approach assumes that all entities are perdurants, making no distinction between endurants and occurants. The idea is that each every entity has a start and an end point. An entity can be seen as a four dimensional "space-time worm", with the slices of the worm being the temporal parts of the entity. With this approach the problem of diachronic

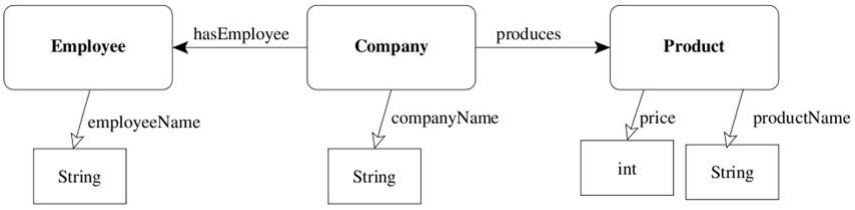

Figure 2.1: Static Enterprise Ontology

entity becomes trivial since an entity is four dimensional and has temporal parts. Changes occur on the properties of the temporal part keeping the entity as a whole unchanged.

The ontology of Figure 2.1 has three classes (concepts), namely *Company*, *Product* and *Employee*. Class *Company* has the data property *companyName* and the object properties *produces* and *hasEmployee*; class *Product* has the data properties *productName* and *price*, and class *Employee* has the data property *employeeName*.

Following the approach by Welty and Fikes [1], to add the time dimension to an ontology, classes *TimeSlice* and *TimeInterval* with properties *tsTimeSliceOf* and *tsTimeInterval* respectively are introduced, as shown in Figure 2.2. n this example, classes Company, Product have properties which may evolve in time (e.g., the name of the company may change, and similarly, the price or name of a product). Class *TimeSlice* is the domain class for entities representing temporal parts (i.e., 'time slices') and class *TimeInterval* is the domain class of time intervals. A time interval holds the temporal information of a time slice. Property *tsTimeSliceOf* connects an instance of class *TimeSlice* with an entity, and property *tsTimeInterval* connects an instance of class *TimeSlice* with an instance of class *TimeInterval*. Properties having a time dimension are called Fluent properties and connect instances of class *TimeSlice*.

2.3 OWL-Time Ontology

OWL-Time [7] is an ontology of temporal concepts. As all the Web services have temporal information, OWL-Time is a complete tool for describing the temporal content of Web pages and the temporal properties of Web services. It provides a vocabulary for expressing facts about topological relations among instants and intervals, together with information about durations, and about date-time information. A simple example is: "Suppose someone has a medical examination scheduled for 6:00pm EST on November 5, 2006. You would like to meet him at 2:00pm PST on the same day for an hour. Will there be an overlap?" In this use case we can specify the facts about the examination and the meeting using our ontology in OWL that will allow a temporal reasoner to determine whether there is a conflict. Figure 2.3 illustrates a diagram of OWL-Time ontology.

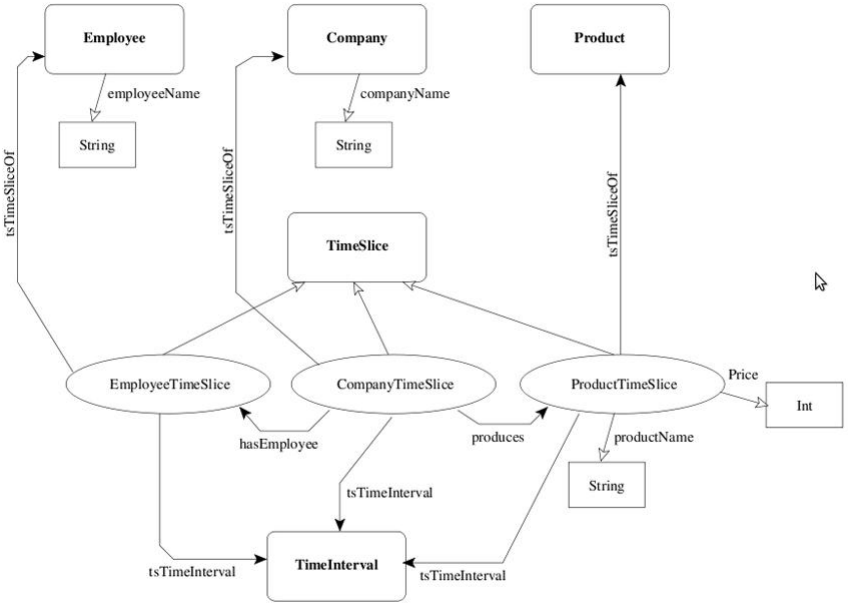

Figure 2.2: Dynamic Enterprise Ontology

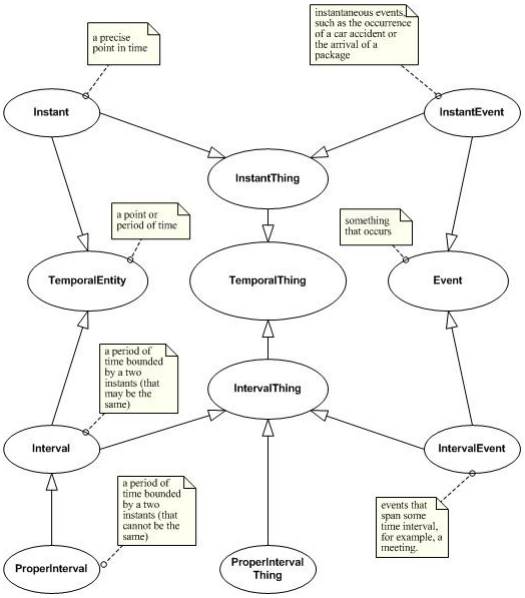

Figure 2.3: An OWL-Time diagram

2.4 Protégé editor

Protégé [4] is a free, open-source platform that provides tools to construct domain models and knowledge-based applications with ontologies and it supports the creation, visualization, and manipulation of ontologies in various representation formats. Protégé can be extended by way of a plug-in architecture and a Java-based Application Programming Interface (API) for building knowledge-based tools and applications.

The Protégé platform supports two main ways of modeling ontologies: the Protégé-Frames editor and the Protégé-OWL editor. The Protégé-OWL editor enables users to build ontologies for the Semantic Web, in particular in the W3C's Web Ontology Language (OWL). "An OWL ontology may include descriptions of classes, properties and their instances. Given such an ontology, the OWL formal semantics specifies how to derive its logical consequences, i.e. facts not literally present in the ontology, but entailed by the semantics. These entailments may be based on a single document or multiple distributed documents that have been combined using defined OWL mechanisms" [3].

2.5 OWL API

There is a wide range of OWL tools available that support the creation and editing of OWL ontologies, reasoning over ontologies, and using ontologies in applications. All of these tools require some kind of underlying API that allow ontologies to be loaded, manipulated and queried. OWL API [2] has been designed to meet the needs of people developing OWL based applications, OWL editors, such as Protégé 4 and OWL reasoners. It is a high level API that is closely aligned with the OWL 2 specification and supports ontology management, ontology change, ontology parsing and rendering, data structure storage and reasoner interfaces.

Chapter 3

4D-Fluents Tab

4D-Fluents Tab is a Protégé Tab Plug-In [6]. It supports the creation and editing of temporal ontologies in OWL 2.0 compatible format, so that with the addition of SWRL rules [14, 15], reasoning can be supported using the built-in Pellet reasoner [9]. Static object properties, data properties and individuals can be converted to temporal easily, without being familiar with the peculiarities of the 4D-fluents model used to represent temporal entities in the ontology. So, anyone who knows how to use Protégé editor and OWL can load a temporal ontology, convert an existing static ontology to dynamic, create a new or edit an existing one. The 4D-Fluents Plug-In will hide all the details of the model that may be complex and confusing to user not familiar with temporal ontology representation models.

3.1 The Difficulty of Handling Temporal Ontologies in Protégé

Protégé editor is particularly well suited for crafting (creating, editing) static ontologies by declaring classes and their instances and properties between them. But for the creation of a dynamic ontology, it seems to be inefficient.

As we explained in section 2.2, for the conversion of a static concept to temporal we have to insert temporal classes, object and data properties, create time interval instances, create time slice instances and connect them to the appropriate interval. The ontology entities increase dramatically, as well as the complexity of the relations between them. These factors are getting worse if we want to include qualitative relations among time intervals, i.e. procedure A occurs *before* procedure B. This, also means that the user must be familiar with representation of time in ontologies and especially with the 4D-fluents model.

The 4D-Fluents Plug-In hides the peculiarities of the mechanism adopted and automatically inserts all the extra information needed. The details of this mechanism are presented and discussed in the following sections, along with use descriptions, user interface and its functionality.

3.2 Use Cases

Because the 4D-Fluents Tab is a Plug-In which deals with temporal information, we need a tool for describing temporal concepts and relations between them. This information can be found in OWL-TIME. More specifically, OWL-TIME provided definitions for intervals, temporal relations, duration, time zones, date and time. So, when the user selects the 4D-Fluents Tab from the Protégé tabs, we check if the loaded ontology is already temporal or not. If the ontology is static, we insert the OWL-Time ontology so that we can use its features. If the loaded ontology is temporal, we do nothing. Next, we are going to discuss some use case tables.

Table 3.1 is the description of use case "Convert an object property to temporal". In this case, the user selects a static property from the "Object Property Hierarchy" and then the button "Convert" from the object property panel. In 4D-fluents model, when a property is temporal, all the individuals that are connected with it, must be temporal too. In this case, a pop-up window appears, in order to inform the user that the individuals connected with the last selected object property, will become temporal. If the user wants to continue, then the object property will be converted to temporal, the same as its associated individuals. If the user cancels the conversion, the object property and the individuals remain static. If the selected object property is a property of OWL-Time ontology that property will remain unchanged.

The use case "Convert a data property to temporal" is described in table 3.2. The user selects a data property from the "Data Property Hierarchy" and the button "Convert" from the data property panel. As we explained at the previous case, when a data property is temporal, all the relevant triplets must be converted. A similar pop-up window appears and the user may select to proceed to the conversion or not. Again, if the selected data property is a property of OWL-Time ontology that property will remain unchanged.

Table 3.3 describes the use case "Create an individual". According to 4D-fluents model, time is not represented in classes or instances, but in properties. In fact, temporal individualsmust be connected with a temporal property. So when the user wants to create an individual, the whole triplet has to be created, namely the domain individual, the property and the range individual. To return in this case, user selects a class and the "Create individual" button from "Individuals" panel. In the "Individuals" dialog appeared, he inserts a name for the new individual, selects a temporal object property and an individual as range. Notice that, only the temporal object properties are displayed along with individuals associated with it individuals. In the following, an interval can be defined in three possible ways:

- The user selects the "Specific dates" button and selects the date of the start point and the date of the end point of the interval.
- The user selects the "Non specific dates" button and selects the date of the start point and the date of the end point, but not both. One of the start or end point are left undefined.
    1. The user selects the "Qualitative Relations" check box and a relation, such as *intervalAfter, intervalContains etc* and one interval from the "Intervals" list.

Use Case 1	**Convert an object property to temporal**
Goal In Context	A user wants to convert a static object property to temporal.
Skope & Level	System, User Goal
Preconditions	The Temporal Tab is selected.
Success End Condition	The object property is temporal and the individuals that are connected with this object property.
Failed End Condition	The object property is static.No data have been modified.
Primary Actors	User
Trigger	User selects an object property and the "Convert" button from the "Object Property" panel.
Description	**Step** **Action** **1** The System displays the "Confirm Individuals Triplets" Dialog. **2** User selects the "Yes" button. **3** The System converts the selected object property to temporal. **4** The System converts the individual triplets to temporal. **5** The System displays the "Object Property" panel.
Extension	**Step** **Branching Action** **1** *User selects the "No" button.* **1.1** The object property remains static. **1.2** The System displays the "Object Property" panel. **2** *User selects an object property of the ontology time.owl.* **2.1** The object property remains static. **2.2** A message at the "Info" text area informs the user that this object property cannot be converted to temporal. **2.3** The System displays the "Object Property" panel.
Sub-Variations	**Step** **Variation**

Table 3.1: Use Case 1

Use Case 2	**Convert an data property to temporal**	
Goal In Context	The user wants to convert a static data property to temporal.	
Skope & Level	System, User Goal	
Preconditions	The Temporal Tab is selected.	
Success End Condition	The data property is temporal and the individuals that are connected with this data property.	
Failed End Condition	The data property is static.No data have been modified.	
Primary Actors	User	
Trigger	The user selects an data property and the Convert to Temporal button from the "Data Property" panel.	
Description	**Step**	**Action**
	1	The System displays the "Confirm Individuals Triplets" Dialog.
	2	The user selects the "Yes" button.
	3	The System converts the selected object property to temporal.
	4	The System converts the individual triplets to temporal.
	5	The System displays the "Object Property" panel.
Extension	**Step**	**Branching Action**
	1	*The user selects the "No" button.* **1.1** The object property remains static. **1.2** The System displays the "Object Property" panel.
	2	*The user selects an object property of the ontology time.owl.* **2.1** The object property remains static. **2.2** A message at the "Info" text area informs the user that this object property cannot be converted to temporal. **2.3** The System displays the "Object Property" panel.
Sub-Variations	**Step**	**Variation**

Table 3.2: Use Case 2

2. The user does not select the “Qualitative Relations”.

The user may add an interval property using the “Intervals of the selected triplet” list with the “Add Interval” button. Similarly, an interval can be deleted by selecting “Delete”. Only intervals confirmed with “Save” will be stored into the ontology. By selecting “Cancel” the changes will not be saved.

Table 3.4 describes the use case “Create an individual”. The user selects a class and the “Create individual” button from “Individuals” panel. In the “Individuals” dialog, he defines the name of the new individual, selects a temporal data property and inserts a proper data value. Only temporal data properties are displayed. The start and end points of a time interval will be defined similarly to the previous case.

Table 3.5 describes the use case “Edit an individual”. Temporal individuals can be edited only in our tab. By selecting a static one, nothing will happen. By selecting an individual and the “Edit Individual” button from “Individuals” panel. In the dialog appeared, the name of the individual is displayed and it is uneditable. The user can select a temporal object property and an individual as its range.

Table 3.6 describes the use case “Edit an individual”. Temporal individuals can be edited only (if a static one is selected, nothing will happen). The user selects an individual and the “Edit Individual” button from “Individuals” panel. In the dialog appeared, the name of the individual is displayed and is is uneditable. The user can select a temporal data property and insert a proper data value. The time interval can be defined similarly.

Table 3.7 describes the use case “Edit an object property assertion”. In fact this option lets the user edit only the interval of a triplet. So, the user selects an object property assertion from the “Property Assertion”, at the “Individuals” panel and then the button “Edit” from the same area. When the dialog appears, the name of the individual, the temporal object property and the range individual displayed are uneditable. Also, at the ‘The intervals of the selected triplet” list is displayed the interval of the triplet. The user’s choices are reduced to interval’s options only. As described before, when the user selects the “Save” button:

- In the “Intervals of the selected triplet” list there are two intervals (or more), the new and the old one. A new object property assertion will be created, and the old one will be retained.
- In the “Intervals of the selected triplet” list there is a new interval (the user has deleted the old interval), a new object property assertion will be created and the selected one will be deleted.
- In the “Intervals of the selected triplet” list there is no interval, the selected object property will be deleted.

Finally, the use case “Edit a data property assertion” is described at table 3.7. In this case the user’s choices are similar to the previous ones. The differences are that the user selects a data property assertion from the “Property Assertion”, at the “Individuals” panel and when the dialog appears, the name of the individual, the temporal data property and the data value are displayed and uneditable. Also, at the ‘The intervals of the selected triplet” list the interval of the triplet is displayed as explained before.

Use Case 3	**Create an individual**	
Goal In Context	The user wants to create a temporal individual.	
Skope & Level	System, User Goal	
Preconditions	The Temporal Tab is selected and some property (object or data) is temporal.	
Success End Condition	A new temporal individual will be created.	
Failed End Condition	No individual will be created. No data have been modified.	
Primary Actors	User	
Trigger	The user selects a class and the Create Individual button from the Individuals panel.	
Description	**Step**	**Action**
	1	The System displays the "Individuals" Dialog.
	2	The user enters a name at the "Individual Name text area.
	3	The user selects the type of the property Object Property from the Property radio button.
	4	The user selects a property from the Properties combo box.
	5	The user selects an individual from the Range combo box.
	6	The user selects the Specific dates radio button.
	7	The user selects the date of the start and end point of the time interval.
	8	The user selects the Add Interval button.
	9	The System displays the time interval at the Intervals text area.
	10	The user selects the Save button.
Extension	**Step**	**Branching Action**
	1	*The user selects the "Cancel" button.* **1.1** No individual is created. **1.2** The System displays the "Individuals" panel.
	2	*User selects the "Non Specific dates" radio button.* **2.1** The user selects the date of the start OR end point of the time interval. **2.2** The user selects the "Add Interval" button. **2.3** The System displays the time interval at the "Intervals" text area. **2.4** The user selects the "Save" button.
	2	*The user selects the "Qualitative relations" check box.* **3.1** The user selects a property from the "Qualitative" combo box. **3.2** The user selects a time interval from the All Intervals list. **3.3** The user selects the "Add Interval" button.
Sub-Variations	**Step**	**Variation**

Table 3.3: Use Case 3

Use Case 4	**Create an individual**	
Goal In Context	The user wants to create a temporal individual.	
Skope & Level	System, User Goal	
Preconditions	The Temporal Tab is selected and some property (object or data) is temporal.	
Success End Condition	A new temporal individual will be created.	
Failed End Condition	No individual will be created. No data have been modified.	
Primary Actors	User	
Trigger	The user selects a class and the "Create Individual" button from the "Individuals" panel.	
Description	**Step**	**Action**
	1	The System displays the "Individuals" Dialog.
	2	The user enters a name at the "Individual Name" text area.
	3	The user selects the type of the property "Data Property" from the 'Property" radio button.
	4	The user selects a property from the "Properties" combo box.
	5	The user enters a value at the "Data Value" text area.
	6	The user selects the "Specific dates" radio button.
	7	The user selects the date of the start and end point of the time interval.
	8	The user selects the "Add Interval" button.
	9	The System displays the time interval at the "Intervals" text area.
	10	The user selects the "Save" button.
Extension	**Step**	**Branching Action**
	1	*The user selects the "Cancel" button.* **1.1** No individual is created. **1.2** The System displays the "Individuals" panel.
	2	*The user selects the "Non Specific dates" radio button.* **2.1** The user selects the date of the start OR end point of the time interval. **2.2** The user selects the "Add Interval" button. **2.3** The System displays the time interval at the "Intervals" text area. **2.4** The user selects the "Save" button.
	2	*The user selects the "Qualitative relations" check box.* **3.1** The user selects a property from the "Qualitative" combo box. **3.2** The user selects a time interval from the "All Intervals" list. **3.3** The user selects the "Add Interval" button.
Sub-Variations	**Step**	**Variation**

Table 3.4: Use Case 4

Use Case 5	**Edit an individual**	
Goal In Context	The user wants to edit a temporal individual.	
Skope & Level	System, User Goal	
Preconditions	The Temporal Tab is selected and the selected individual is temporal.	
Success End Condition	The individual will be edited.	
Failed End Condition	The individual will not be edited.	
Primary Actors	User	
Trigger	The user selects an individual and the "Create Individual" button from the "Individuals" panel.	
Description	**Step**	**Action**
	1	The System displays the "Individuals" Dialog.
	2	The system displays the name of the selected individual at the "Individual Name" text area.
	3	The user selects the type of the property "Object Property" from the "Property" radio button.
	4	The user selects a property from the "Properties" combo box.
	5	The user selects an individual from the "Range" combo box.
	6	The user selects the "Specific dates" radio button.
	7	The user selects the date of the start and end point of the time interval.
	8	The user selects the "Add Interval" button.
	9	The System displays the time interval at the "Intervals" text area.
	10	The user selects the "Save" button.
Extension	**Step**	**Branching Action**
	1	*The user selects the "Cancel" button.* **1.1** No individual is created. **1.2** The System displays the "Individuals" panel.
	2	*The user selects the "Non Specific dates" radio button.* **2.1** The user selects the date of the start OR end point of the time interval. **2.2** The user selects the "Add Interval" button. **2.3** The System displays the time interval at the "Intervals" text area. **2.4** The user selects the "Save" button.
	3	*The user selects the "Qualitative relations" check box.* **3.1** The user selects a property from the "Qualitative" combo box. **3.2** The user selects a time interval from the All Intervals list. **3.3** The user selects the "Add Interval" button.
Sub-Variations	**Step**	**Variation**

Table 3.5: Use Case 5

Use Case 6	**Edit an individual**	
Goal In Context	The user wants to edit a temporal individual.	
Skope & Level	System, User Goal	
Preconditions	The Temporal Tab is selected and the selected individual is temporal.	
Success End Condition	The individual will be edited.	
Failed End Condition	The individual will not be edited.	
Primary Actors	User	
Trigger	The user selects an individual and the "Create Individual" button from the "Individuals" panel.	
Description	**Step**	**Action**
	1	The System displays the "Individuals" Dialog.
	2	The system displays the name of the selected individual at the "Individual Name" text area.
	3	The user selects the type of the property "Data Property" from the "Property" radio button.
	4	The user selects a property from the "Properties" combo box.
	5	The user enters a value at the "Data Value" text area.
	6	The user selects the "Specific dates" radio button.
	7	The user selects the date of the start and end point of the time interval.
	8	The user selects the "Add Interval" button.
	9	The System displays the time interval at the "Intervals" text area.
	10	The user selects the "Save" button.
Extension	**Step**	**Branching Action**
	1	*The user selects the "Cancel" button.* **1.1** No individual is created. **1.2** The System displays the "Individuals" panel.
	2	*User selects the "Non Specific dates" radio button.* **2.1** The user selects the date of the start OR end point of the time interval. **2.2** The user selects the "Add Interval" button. **2.3** The System displays the time interval at the "Intervals" text area. **2.4** The user selects the "Save" button.
	3	*The user selects the "Qualitative relations" check box.* **3.1** The user selects a property from the "Qualitative" combo box. **3.2** The user selects a time interval from the All Intervals list. **3.3** The user selects the "Add Interval" button.
Sub-Variations	**Step**	**Variation**

Table 3.6: Use Case 6

Use Case 7	**Edit an object property assertion**	
Goal In Context	The user wants to edit the time interval of an individual.	
Skope & Level	System, User Goal	
Preconditions	The Temporal Tab is selected and the selected individual is temporal.	
Success End Condition	The time interval of the individual will be edited.	
Failed End Condition	The time interval of the individual will not be edited.	
Primary Actors	User	
Trigger	The user selects an object property assertion of the selected individual and the "Edit" button from the "Individuals" panel.	
Description	**Step**	**Action**
	1	The System displays the "Individuals" Dialog.
	2	The system displays the name of the selected individual at the "Individual Name" text area.
	3	The System displays the type of the property (object) from the "Property" radio button.
	4	The System displays the property from the "Properties" combo box.
	5	The System displays the individual from the "Range" combo box.
	6	The user selects the "Specific dates" radio button.
	7	The user selects the date of the start and end point of the time interval.
	8	The user selects the "Add Interval" button.
	9	The System displays the time interval at the "Intervals" text area.
	10	The user selects the "Save" button.
Extension	**Step**	**Branching Action**
	1	*The user selects the "Cancel" button.* **1.1** No changes at the selected object property assertion are made. **1.2** The System displays the "Individuals" panel.
	2	*The user selects the "Non Specific dates" radio button.* **2.1** The user selects the date of the start OR end point of the time interval. **2.2** The user selects the "Add Interval" button. **2.3** The System displays the time interval at the "Intervals" text area. **2.4** The user selects the "Save" button.
	3	*User selects the "Qualitative relations" check box.* **3.1** The user selects a property from the "Qualitative" combo box. **3.2** The user selects a time interval from the All Intervals list. **3.3** The user selects the "Add Interval" button.
Sub-Variations	**Step**	**Variation**

Use Case 8	**Edit an object property assertion**	
Goal In Context	The user wants to edit the time interval of an individual.	
Skope & Level	System, User Goal	
Preconditions	The Temporal Tab is selected and the selected individual is temporal.	
Success End Condition	The time interval of the individual will be edited.	
Failed End Condition	The time interval of the individual will not be edited.	
Primary Actors	User	
Trigger	The user selects an data property assertion of the selected individual and the "Edit" button from the "Individuals" panel.	
Description	**Step**	**Action**
	1	The System displays the "Individuals" Dialog.
	2	The system displays the name of the selected individual at the "Individual Name" text area.
	3	The System displays the type of the property (data) from the "Property" radio button.
	4	The System displays the property from the "Properties" combo box.
	5	The System displays the value at the "Data Value" text area.
	6	The user selects the "Specific dates" radio button.
	7	The user selects the date of the start and end point of the time interval.
	8	The user selects the "Add Interval" button.
	9	The System displays the time interval at the "Intervals" text area.
	10	The user selects the "Save" button.
Extension	**Step**	**Branching Action**
	1	*The user selects the "Cancel" button.* **1.1** No changes at the selected object property assertion are made. **1.2** The System displays the "Individuals" panel.
	2	*User selects the "Non Specific dates" radio button.* **2.1** The user selects the date of the start OR end point of the time interval. **2.2** The user selects the "Add Interval" button. **2.3** The System displays the time interval at the "Intervals" text area. **2.4** The user selects the "Save" button.
	3	*The user selects the "Qualitative relations" check box.* **3.1** The user selects a property from the "Qualitative" combo box. **3.2** The user selects a time interval from the All Intervals list. **3.3** The user selects the "Add Interval" button.
Sub-Variations	**Step**	**Variation**

3.3 User Interface design

The interface layout was inspired from the structure of standard Protégé tabs. Our attempt has been to retain the appearance and functionality of a standard Protégé tab, so users of Protégé wont experience difficulties in using it. We group different information in three different panels. The characteristics and the description of object properties are banded together in one panel as we can see in Figure 3.1. The characteristics and the descriptions of data properties are put together in one panel too (figure 3.2) and similarly the characteristics and the property assertions of individuals (figure 3.3). These panels are divided in four areas. In general, buttons for the conversion appear at the up and left side or a panel and text messages are displayed at the up and right side of a panel.

At the left of the screen, we have four different views: *Class Hierarchy, Object Property Hierarchy, Data Property Hierarchy* and *Individuals by type*. For each one a different panel appears. Figure 3.1 illustrates the view of our tab, when the selected entity is an object property. At the right is the main panel, which looks like with the default tab "Object Properties" of Protégé. According to this default tab, we designed the "Characteristics" panel and the "Description" panel. In the first one, there are check boxes to indicate if an object property is *Functional, Inverse Functional, Transitive, Symmetric, Asymmetric, Reflexive* and *Irreflexive*. In the "Description" panel, there are jLists to show the *Domain, Range, Equivalent Properties, Super Properties, Inverse Properties* and *Disjoint Properties*, with the corresponding labels. Above these two panels, we see the "Convert to Temporal" button and the text area "Info". Some info-messages and some instructions to help the user are displayed in this area.

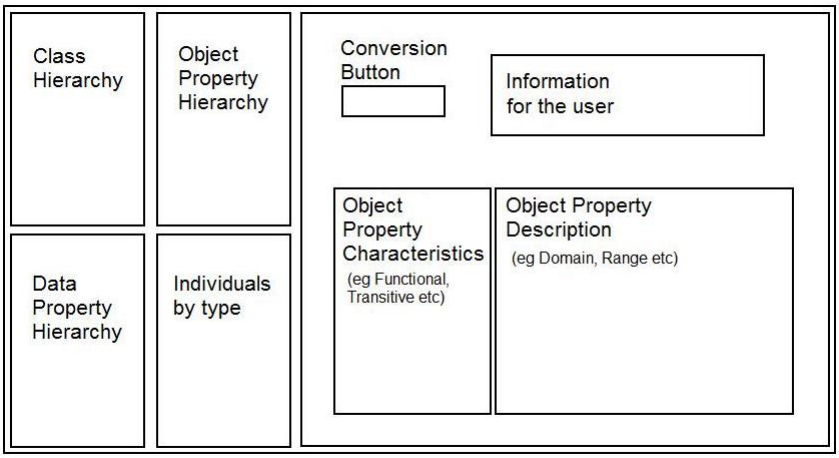

Figure 3.1: Object Property Panel

At Figure 3.2 we see how our tab looks like, when the selected entity is a data property. We designed it according to the default Protégé tab "Data Properties". Thus, these is the "Characteristics" panel, which includes the *Functional* check box and the "Description" panel with the jLists *Domain, Range, Equivalent Properties, Super Properties* and *Disjoint Properties*. Also, above these two panels, we see the "Convert to Temporal" button and the text area "Info".

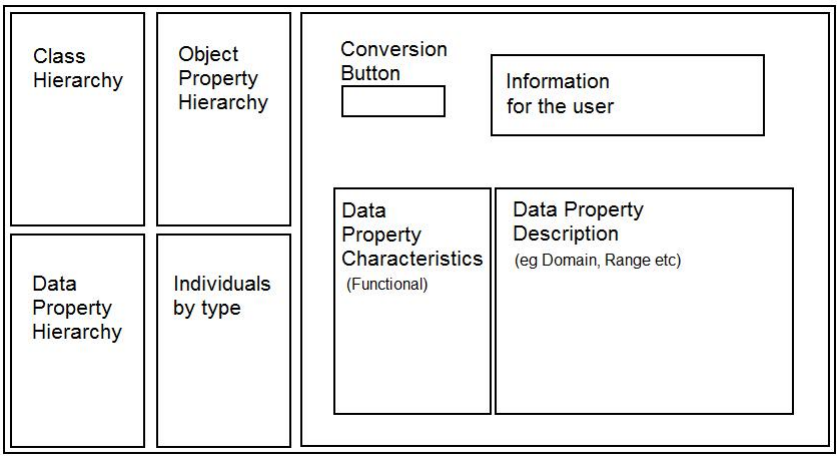

Figure 3.2: Data Property Panel

Figure 3.3 shows the view of our tab, when the selected entity is an individual or a class. According to the default "Individuals" tab of Protégé, we designed the "Description" panel and the "Property Assertions" panel. In the first one, there are check boxes to demostrate the individual *Type*, the *Same Individuals* and the *Different Individuals*. "Property assertions" panel has one jList for the *Object Property Assertions* and one for the *Data Property Assertions*, with the corresponding labels. These jLists respectively to their name, display the property assertions of the selected individual. Above these two panels, there are the "Create/Edit Individual" button and the text area "Info".

Note that in each panel, there are colorful details, such as colored labels and borders, depending on the type of the selected entity. If the selected entity is an object property we use the blue color, if it is a data property we use green and if it is an individual we use a hue of purple. The coloring rules that we use, are according to Protégé, so the user can easily understand and identify the type of entity.

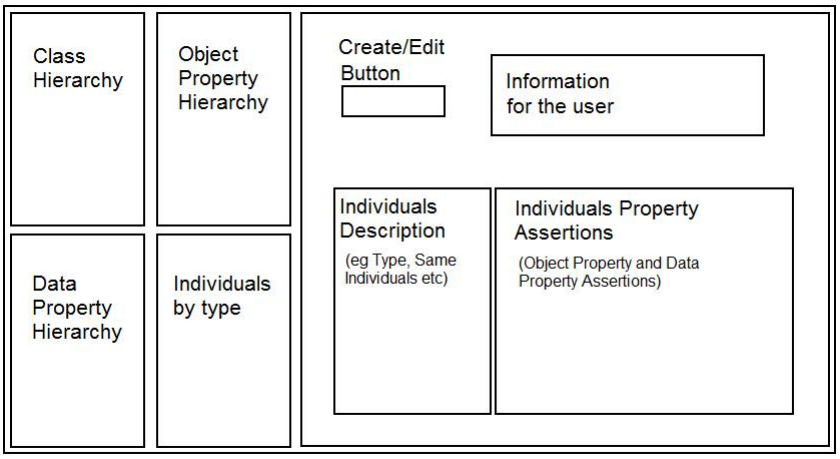

Figure 3.3: Individual Panel

3.4 Functionality

Below, we discuss the functionality that our Plug-In supports. Figure 3.4 is a screen-shot of Object Property View. This view is related to *Use Case 1: Convert an object property to temporal*. We see the *Object Property Hierarchy*, where the user can select an object property and the "Convert" button to make it temporal. When the user selects this button a pop-up window informs him for the relevant individuals triplets that are going to change. The window is shown in Figure 3.5.

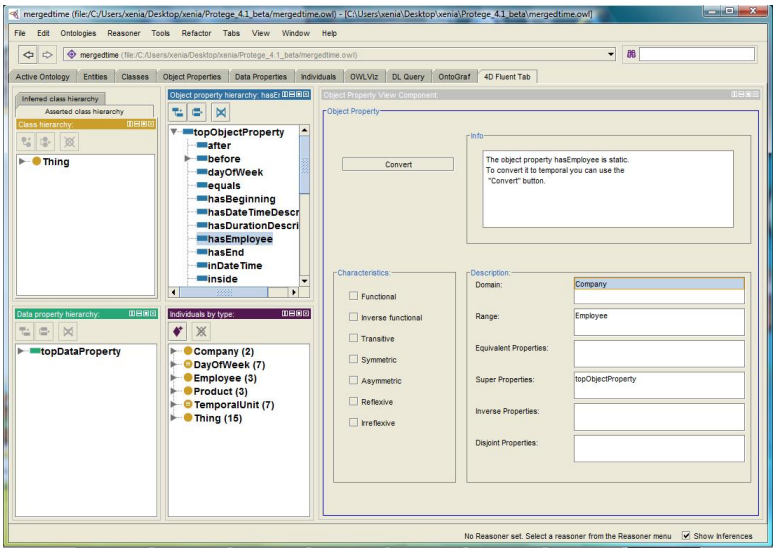

Figure 3.4: 4D-Fluents Tab: Object Properties View

Figure 3.6 is a screen-shot of Data Property View. This view is related to *Use Case 2: Convert a data property to temporal*. We also, see the *Data Property*

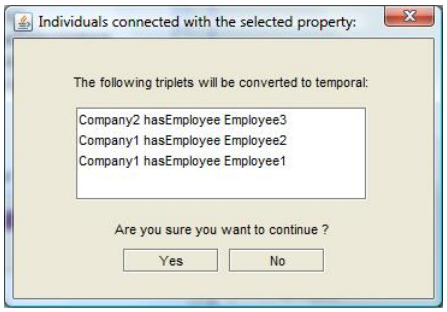

Figure 3.5: 4D-Fluents Tab: Confirm the conversion of an object property to temporal

Hierarchy, where the user can select a data property and the "Convert" button to make it temporal. In Figure 3.7 shows the pop-up window that informs the user for the individuals triplets that are going to change, when the "Convert" button is selected.

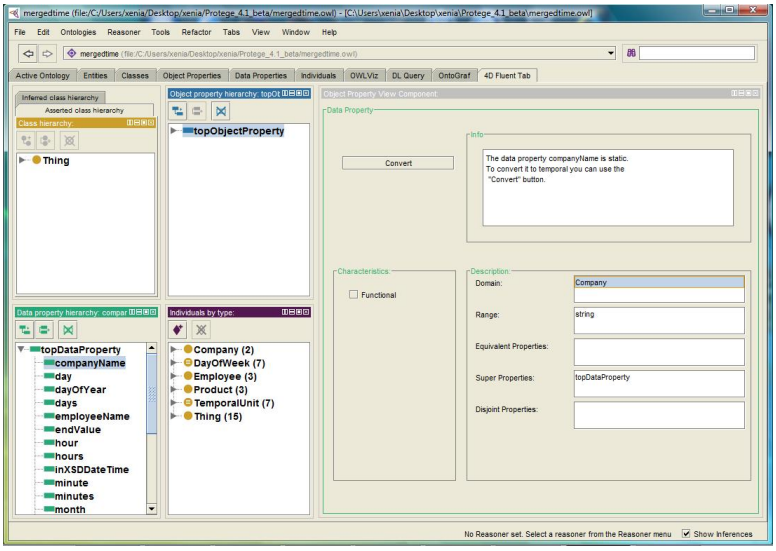

Figure 3.6: 4D-Fluents Tab: Data Properties View

Figure 3.8 is a screen-shot of Individual View. This view is related to *Use Case 3,4: Create an individual, Use Case 5,6: Edit an individual, Use Case 7: Edit an object property assertion, Use Case 8: Edit a data property assertion.* We see the *Individuals by type* view, where the user can select an individual or a class and the "Create/Edit" button to create or edit an individual. When the user selects this button the dialog "Create/Edit an Individual" appears. This dialog is shown in Figure 3.10 and is used in same use cases as the Individual view. Also, no individual can be created or edited, if there is no temporal property - object or data. This is indicated by the dialog showed in Figure 3.9.

In the dialog of Figure 3.10, we see that in the upper half of the screen, the user can insert the name of the individual, select the property and the range.

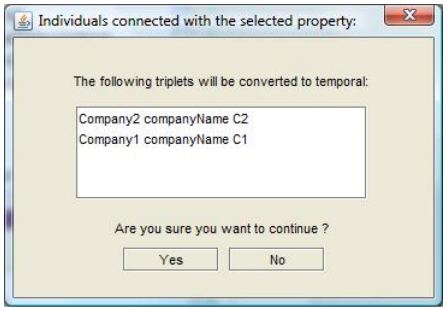

Figure 3.7: 4D-Fluents Tab: Confirm the conversion of a data property to temporal

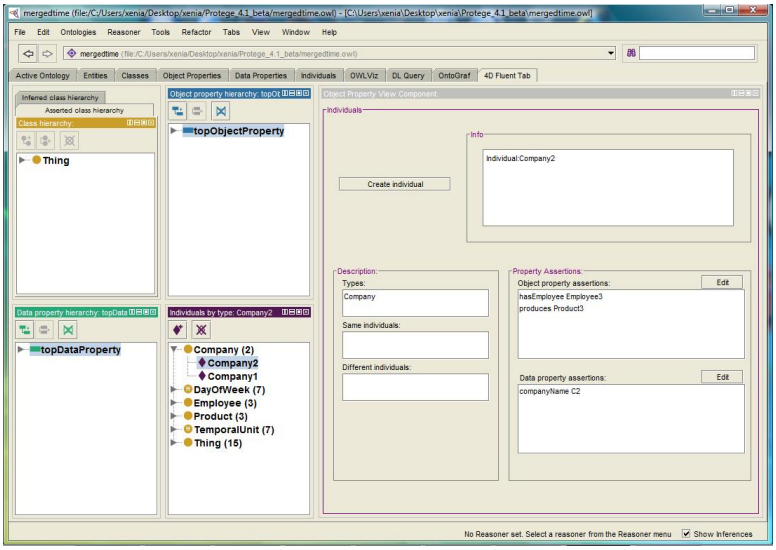

Figure 3.8: 4D-Fluents Tab: Individuals View

Figure 3.9: 4D-Fluents Tab: Dialog to indicate that there is no temporal property

Below this area, the interval with can be declared. The user can select the dates of the start and end point, the qualitative relations between the intervals and delete an interval. He can save the changes or abort the procedure.

An important extension that our team succeeded on that 4D-fluents model, is the support of qualitative relations. The main idea of these relations is that when we want to fit a fact in time, we may not know the exact dates of its endurance, but know that it happened before or after something else. For example, we know that Liza lived in Denmark from 2005 to 2009 and that her brother, Tom, visited her once as long as she was there, but we do not know exactly when. As we do not have the specific dates of his visit we could loose this information because we would n't have any way to represent it. Below we discuss the mechanism of qualitative relations and help us declare that the interval Liza lived there "contains" the interval of Tom's visit. More details are presented in section 3.6.

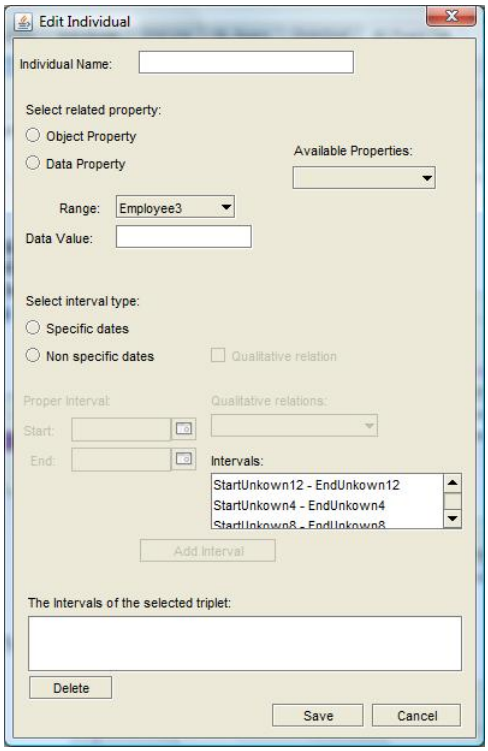

Figure 3.10: 4D-Fluents Tab: Individuals Dialog

3.5 Activity Diagrams

In this section we show the activity diagrams of the use cases in section 3.2. Each activity correlates with the use case with the same number. They illustrate the steps that a user has to follow for different procedures.

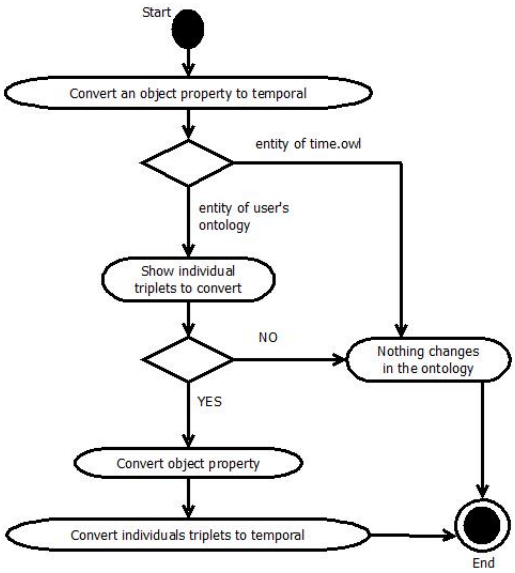

Figure 3.11: Activity Diagram 1

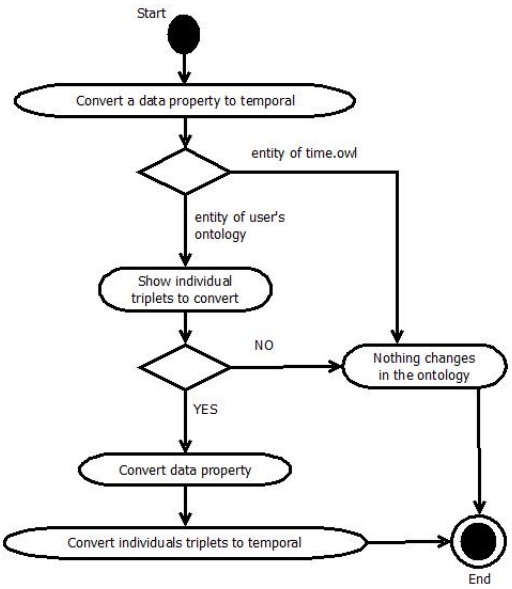

Figure 3.12: Activity Diagram 2

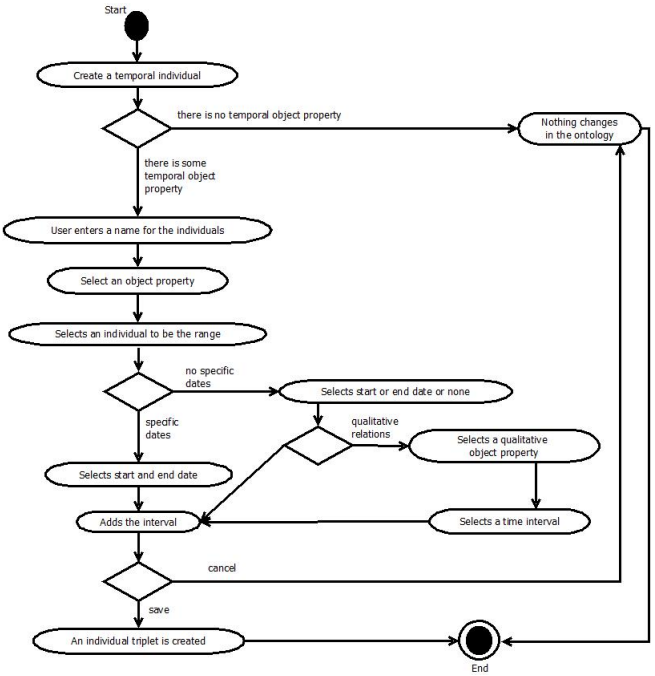

Figure 3.13: Activity Diagram 3

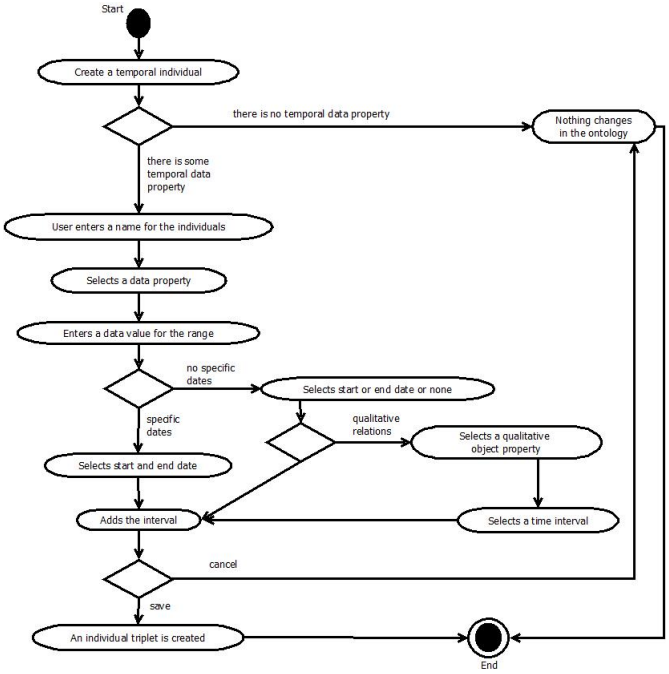

Figure 3.14: Activity Diagram 4

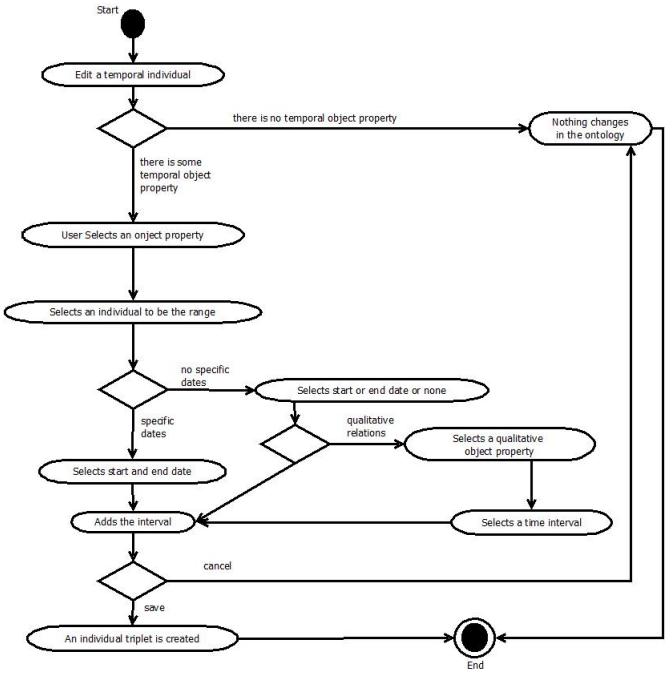

Figure 3.15: Activity Diagram 5

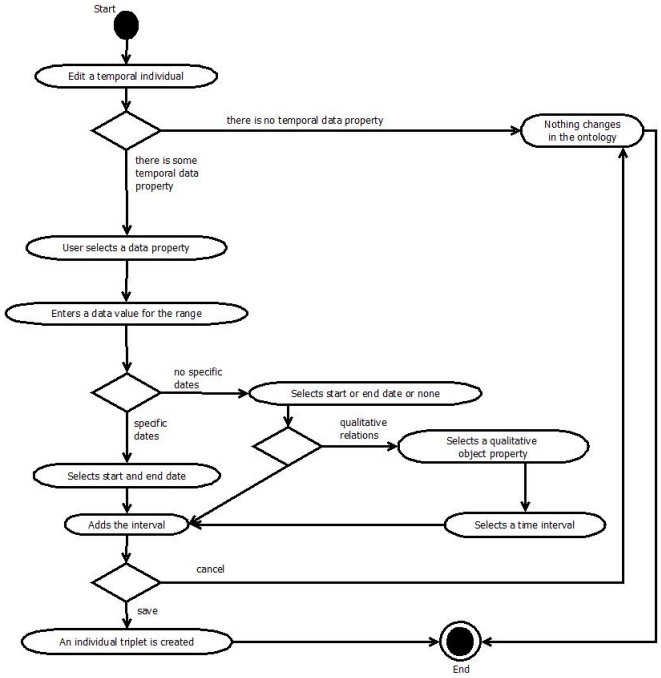

Figure 3.16: Activity Diagram 6

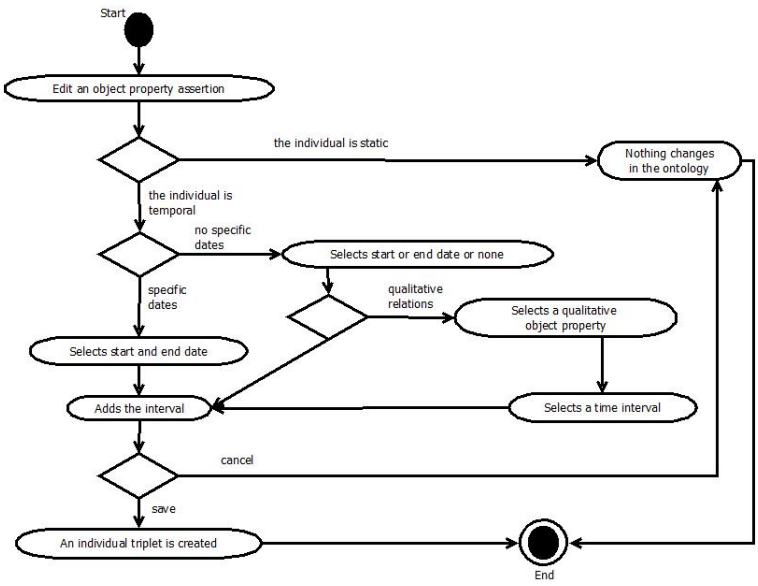

Figure 3.17: Activity Diagram 7

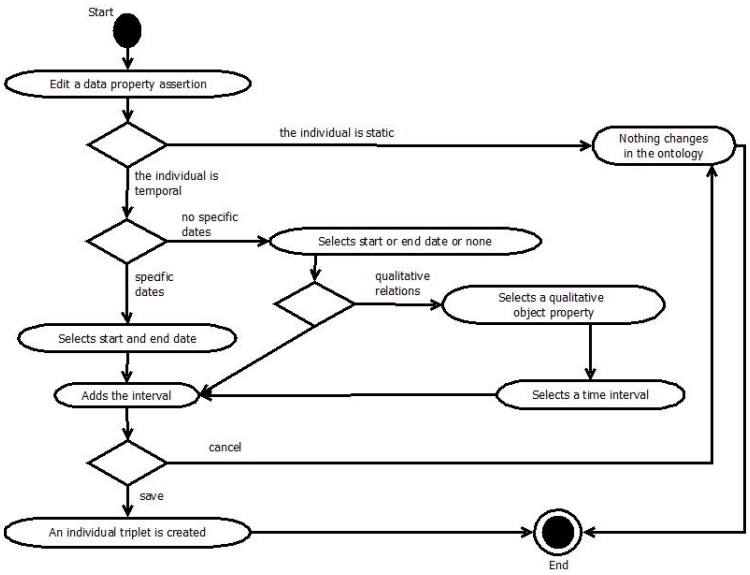

Figure 3.18: Activity Diagram 8

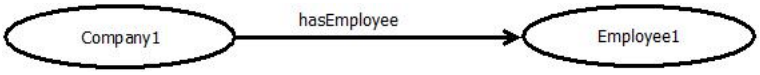

Figure 3.19: A static object property assertion

3.6 Ontology Changes According to 4D-fluents model

Whenever an entity is converted to temporal lots of changes occur in the underlying ontology.

When the user selects our tab from Protégé, immediately the loaded ontology is merged with OWL Time (time.owl). This is a needful step, as *time.owl* contains all the appropriate information to represent time. Using OWL API functions we join these two ontologies and we save them as a new one. We make the new ontology, active in Protégé and we delete the old one and the *time.owl*, so they do not confuse the user. Among others, the classes *TimeSlice*, *ProperInterval*, the object properties *tsTimeSliceOf*, *tsTimeInterval*, *hasBeginning*, *hasEnd* and the data property *xsdDateTime* are inserted, which are necessary in 4D-fluents model. So now, we are ready to handle the conversion of an entity to temporal.

That's why our Plug-In won't let the user change anything at the individuals, before he makes at least one property temporal. Accordingly to the 4D-fluents model, when the user selects to convert a property, in fact what is changing is the domain and the range of the property. The new domain and range are time slices of the existing ones. For example, we want to convert the object property *hasEmployee*, with domain the class *Company* and range the class *Employee*. After the conversion the property will remain, but the domain will be *timeSliceOf* **only** *Company* and the range *timeSliceOf* **only** *Employee*. The same stands for the data properties, except that the range does not change.

The conversion of an individual is more complicated, particularly because of the interval that must be represented here. Figure 3.19 illustrates an object property assertion when there is no time sequence. Two individuals *Company1* and *Employee1* are connected with the object property *hasEmployee*. If the user decides to add time information in this triplet, the following changes are made: first, two new instances of *TimeSlice* class are created, *Company1TimeSlice1* and *Empolyee1TimeSlice1*, and are connected with the *tsTimeSliceOf* object property to the individuals *Company1* and *Employee1* respectively. Then the *hasEmployee* property is introduced between them.

The representation of the time interval, that the user wants to add to this triplet, is implemented in steps: An instance of *ProperInterval* class is created and is connected with the *tsTimeInterval* object property, with the two time slices. This individual, *ProperInterval1*, "holds" the time frame. It is connected with two instances of *Instant* class, the *StartInstant* and *EndInstant*, with the *hasBeginning* and the *hasEnd* object properties correspondingly. If the start or end point of the interval is undefined, the *StartInstant* or the *EndInstant* are named *StartUnknown* and *EndUnknown*, and nothing else is added. But, if the user wants to add specific dates, the *StartInstant* and *EndInstant* are connected with the *xsdDateTime* data property to a DateTime data type. The converted

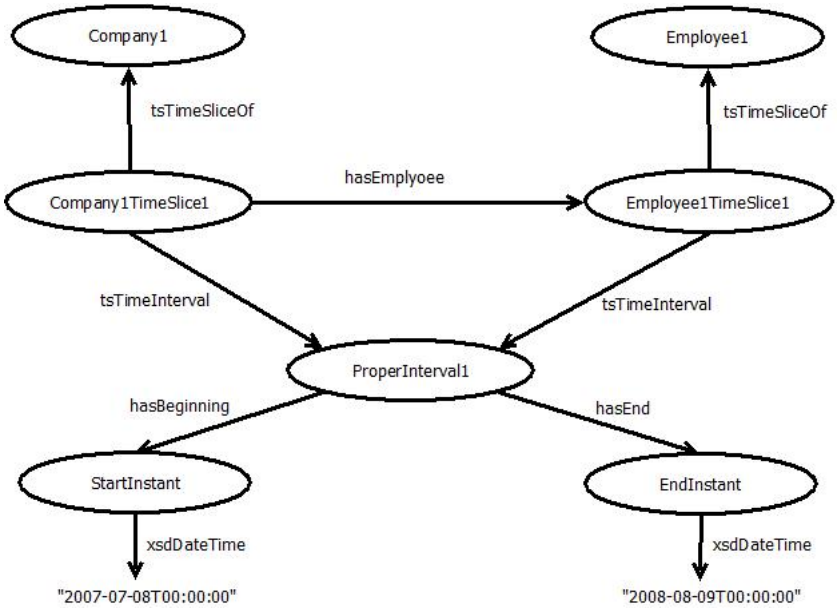

Figure 3.20: A temporal object property assertion

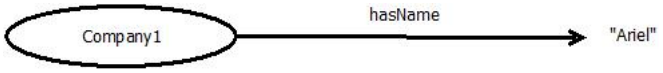

Figure 3.21: A static data property assertion

object property, as we described it above, shows Figure 3.20. In this example we suppose that the dates are specific.

If the user wants to declare a qualitative relation between two intervals, he can choose one from the following object properties: *intervalAfter, intervalContains, intervalDuring, intervalEquals, intervalFinishedBy, intervalFinishes, intervalMeets, intervalMetBy, intervalOverlappedBy, intervalOverlaps, intervalStartedBy, intervalStarts*. The interval that he is creating will be the domain of the qualitative property, and as range he selects one from the list with all the intervals of the ontology.

When the user wants to convert a data property assertion we follow the exact same steps. The only thing that changes from the object property assertion case, is that the range remains the same. The data type, whatever it is, is not connected with the instance of *ProperInterval* class that is created. Figure 3.21 and figure 3.22 show the conversion of a data property assertion.

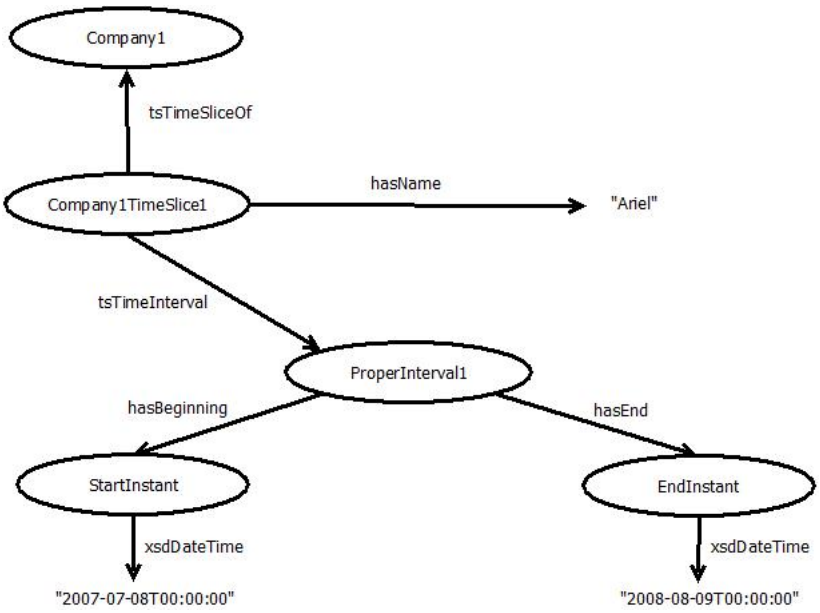

Figure 3.22: A temporal data property assertion

3.7 Code Structure

We organized our source code in several packages, in order to be more flexible and some parts of it reusable. Especially the part which implements the interface of the Plug-In. As we designed the tab, to display three different panels depending on the selected entity, that are very similar to the default ones, this code can be used easily in other tabs for multiple purposes.

objectPropertyPanels, dataPropertyPanels, individualsPanels : We created a package for every panel. Every package has four classes and each one implements a part of the panel. There is one class to create the *Button* and to handle its events, one to create the *Text area* and display the appropriate messages, one to create the *Description* panel with its lists and labels and one to create the *Characteristics* with its check boxes. These classes are similar for all the packages, but they have little differences to cover all the functionalities we need for every entity. For example, in the *individualPanels* package instead of the description area, there is the the *Property Assertions* panel, which shows the object property and data property assertions for every individual and has two buttons for their editing, as we saw in Figure 3.8.

Additionally, in the *individualsPanels* and *objectPropertyPanels* packages there two more classes, *AskIndDialog* and *CreateIndDialog* respectively. These classes implement the dialogs we see in Figure 3.5 and 3.10. Mainly the second one is very significant as it handles all the temporal individuals. In this class, we manage the creation and the editing of the individuals, their property assertions, and their intervals and all the qualitative relations among them.

tab : All these panels are used in the *tab* package. There are three classes *ObjectPropertyMainPanel*, *DataPropertyMainPanel* and *IndividualsMainPanel*, that combine them and illustrates the complete panel. In the same package there is the *TimeFactory* class. This a very important class, as it implements all the methods that are necessary for the conversion of properties and individuals, handles the intervals and their deletion, handles the property assertions of the individuals and maintains the ontology correct accordingly to 4D-fluents model. Some of these methods will be discussed in the section 3.8.

In the *tab* package also exists the *ViewComponent* class which extends the *AbstractOWLViewComponent* and inherits from Protégé all the necessary features for the creation of the tab. In this class we take the current ontology loaded in Protégé and we check if it is already temporal in order to merge it with *time.owl* or not.

utils : Because of the complexity of the 4D-fluents model and OWL-API, it is difficult to manage the temporal ontology. There are many times we need an complex entity, a "time slice" individual for example, and we follow several properties and axioms of the ontology to find it. It would not be efficient to repeat this procedure every time we needed it. Hence the need for the *utils* package. This package contains twelve classes and each one implements special structures that keep the information we need. Without them the management of the temporal ontology and its editing would be a difficult matter to deal with.

Figure 3.23 illustrates the class diagram of our project.

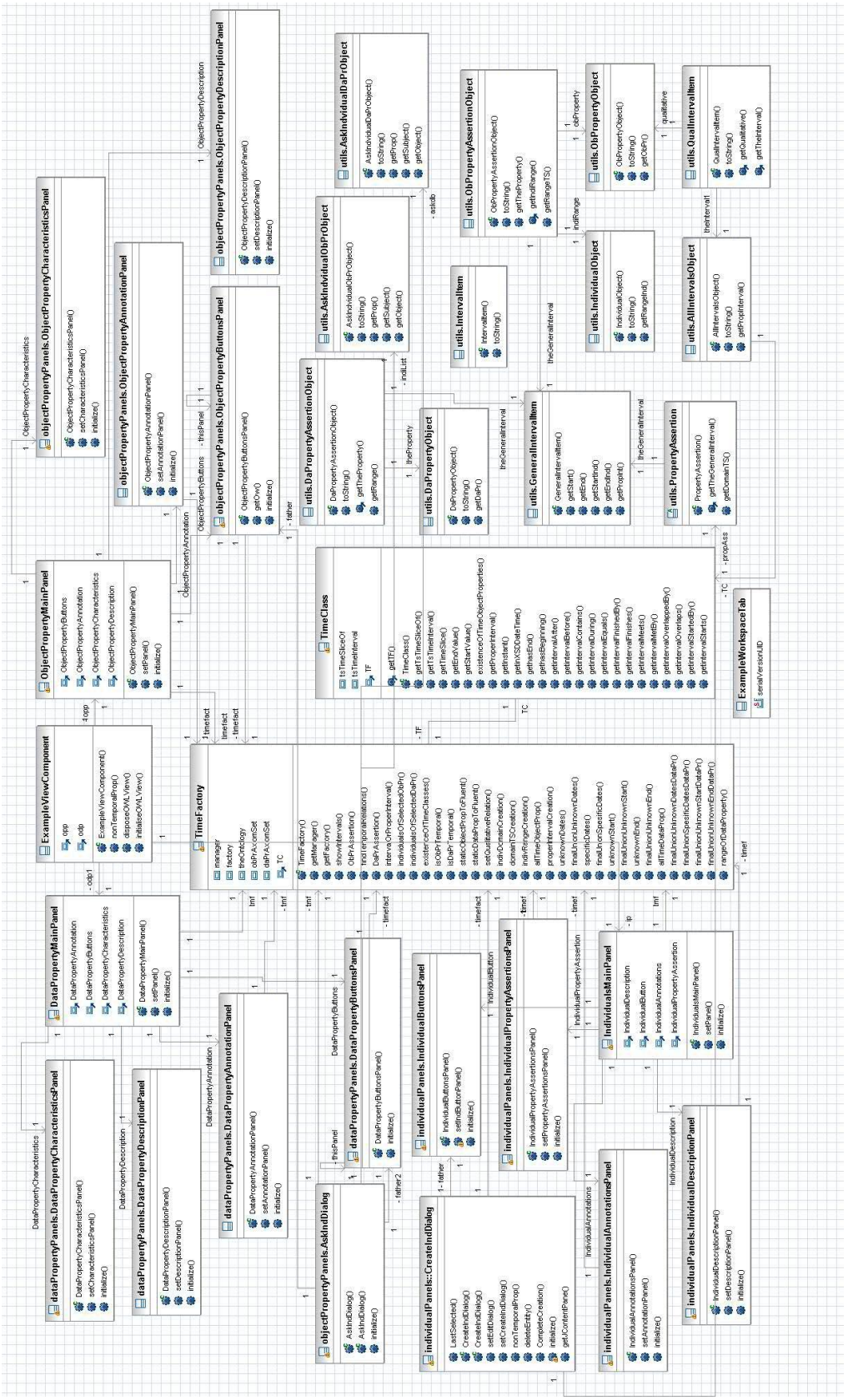

Figure 3.23: Class Diagram

3.8 Plug-In Documentation

In this section, we discuss some important methods that maintain ontology consistency accordingly to 4D-fluents model. Such methods are created in *TimeFactory* class of *tab* package.

At first, there are the methods that manage the **properties**:

- `staticOblectPropToFluent(OWLObjectProperty)` and `staticDataPropToFluent(OWLDataProperty)`, with arguments object property and data property respectively. They find the domain and the range of the property, change them and delete the old ones.
- `allTimeObjectProp()` and `allTimeDataProp()`, which iterates all the properties of the ontology and returns only those which are already temporal.
- `isObPrTemporal(OWLObjectProperty)` and `isDaPrTemporal(OWLDataProperty)`, with arguments object property and data property respectively. They check if the input is temporal or static.
- `individualsOfSelectedObPr(OWLObjectProperty)` and `individualsOfSelectedDaPr(OWLDataProperty)`, with arguments object property and data property respectively. They find all the individuals that are related with the input and returns a vector filled with *AskIndvidualObPrObject* or *AskIndvidualDaPrObject* objects.

Likewise, there are functions to manage the **individuals**:

- `indivDomainCreation(String, OWLClass)` and `indivRangeCreation-(OWLNamedIndividual)` with arguments a string and a class. It creates an instance of the input class and names it with the input string.
- `properIntervalCreation()` creates an instance of *ProperInterval* class.
- `specificDates(String,String,OWLNamedIndividual)` with arguments two strings and an individual that is an instance of *ProperInterval* class. The strings are the start and end date that the user has selected for the interval. This function creates two instances of *Instant* class, one for the start point "StartInstant" and one for the end "EndInstant". It converts the input strings into DateTime data type and connects it with the instances. As well as the input "proper interval" individual with the data properties *hasBeginning* and *hasEnd* with the "StartInstant" and "EndInstant". The functions `unknownDates()`, `unknownStart()`, `unknownEnd()` are working in a similar way and we need the first when we do not know the interval ends, the second when we do know only the end point and the third when we only know the start point.
- `finalUnionSpesificDates(OWLObjectProperty, OWLNamedIndividual, OWLNamedIndividual, OWLNamedIndividual, String, String)` with arguments an object property, an instance of the domain class, an instance of the range class, an instance of the *ProperInterval* class and two strings. The two strings and the "proper interval" individual are used when the `specificDates()` is called. The object property connects the domain

individual and the range individual. These two are connected with the proper interval. So, a triplet is created that represents also the time interval in which it exists. The functions `finalUnionUnknownDates()`, `finalUnionUnknownStart()`, `finalUnionUnknownEnd()` are working in a similar way and are needed when we do not know the exact interval ends.

- `finalUnionSpesificDatesDataPr(OWLDataProperty,OWLNamedIndividual, String, OWLNamedIndividual, String, String)` with arguments a data property, an instance of the domain class, a string with the data value, an instance of the *ProperInterval* class and two more strings. The two strings and the "proper interval" individual are used when the `specificDates()` is called. The data property connects the domain individual with its proper data value. The domain individual is connected with the proper interval. So, a triplet is created that represents also the time interval in which it exists. The functions `finalUnionUnknownDatesDataPr()`, `finalUnionUnknownStartDataPr()`, `finalUnionUnknownEndDataPr()` are working in a similar way and are needed when we do not know the exact interval ends.

- `setQualitativeRelation(OWLNamedIndividual, OWLObjectProperty, OWLIndividual)` with arguments two instances of *ProperInterval* class and an object property. It connects the individuals with the property that defines a qualitative relation between intervals.

There are also functions that handle the **property assertions** and the **intervals** of the individuals.Some of them are:

- `ObPrAssertion(OWLNamedIndividual)` with arguments an individual, usually the selected one by the user. It is a significant function as it finds and keeps all the individuals that take part in a 4D-fluents triplet. To explain it really simply the steps that it follows are: starting from the input it finds its time slice, the object property, the range time slice individual and the range individual. From the domain time slice it finds the proper interval individual. To handle the interval we use the functions we are discussing later.

- `DaPrAssertion(OWLNamedIndividual)` with arguments an individual, usually the selected one by the user. The steps are similar to the previously mentioned function: starting from the input it finds its time slice individual, the data property and the data value. From the domain time slice it finds the proper interval individual.

- `showIntervals()` iterates all the intervals of all the individuals and returns a vector filled with "AllIntervalsObject" objects.

- `showIntervalsEnds()` finds the ends of an interval following the connections with the *Instants* and then with the *DateTime* data values.

- `intervalOrProperInterval(OWLNamedIndividual)` with arguments an instance of *ProperInterval*. This functions finds the interval ends of the input, but checks also if the input is connected with a qualitative relation with an other interval. If so, it finds the type of the qualitative relation, the second interval and its ends and returns a "GeneralIntervalItem" object.

In package *utils* there are some classes that help us keep and find the information we need directly, when we need it. An example is "GeneralIntervalItem" that we mentioned before. This class holds the proper interval individual and its intervals ends. "IntervalItem" and "QualIntervalItem" extend "GeneralIntervalItem" and inherit all the functions and the characteristics of the last one. "IntervalItem" overrides the function *toString()* for the proper interval. "QualIntervalItem" keeps the qualitative relation, the second proper interval individual and its ends, and overrides the *toString()* for this triplet.

Finally, in packages *objectPropertyPanels*, *dataPropertyPanels* and *individualsPanels* there are functions to manage the layout and the features to be displayed, as we explained at previous sections. For example, `setCharacteristicsPanel()` checks the characteristics of the property (functional, transitive, etc) and enables the appropriate check box. `setDescriptionPanel()` function displays the domain, the range, etc of a property at the appropriate list. At this point we have to mention, that these methods hide all the needless information that concerns the temporal parts, the time slice individuals, proper interval individuals etc. He sees the ontology like it was a simple one. For example, in Figure 3.20 the property domain is the individual *Compnay1TimeSlice1*, but our tab shows as domain the individual *Company1*.

Chapter 4

Conclusion and future work

We introduce 4D-Fluents Tab Plug-In, a tool for crafting temporal ontologies in Protégé. The temporal concepts are represented by means of the 4D-fluents [1] mechanism implementing events occurring or evolving in time. As such, temporal information cannot be handled directly by an ontology editor such as Protégé. The 4D-Fluents Plug-In facilitates the creation and editing of temporal ontologies. Moreover, it does not require that the user be familiar with the peculiarities of the temporal representation mechanism(the 4D-fluents approach in our work) adopted, making it easy to use by ordinary users of the semantic web.

Extending our mechanism to support temporal restrictions, that is a restriction holding on dynamic properties, is an interesting issue for future work. In this case, a restriction must progress over time in the same way the property does. Then, consistency must be ascertained not only once, but at every instance of time, i.e., for every timeSlice the property is associated with. The property restrictions that change when a property is dynamic, for object properties are *Functional*, *Inverse Functional* and *Transitive* and for data properties is *Functional*. The value constraints that differentiate are *OWL:allValuesFrom*, *OWL:someValuesFrom* and *OWL:hasValue* and the cardinality constraints are *OWL:maxCardinality*, *OWL:minCardinality* and *OWL:ExactCardinality*.

In addition, implementing reasoning mechanisms as a Plug-In to Protégé, to perform restriction checking, conclusion extracting from temporal relations, i.e, Allen algebra, and path consistency would offer additional useful functionality to our work. Future work also includes extending our Plug-In to support other time representation models, such as N-ary model which is a W3C recommendation.

Bibliography

[1] C. Welty, R. Fikes, and S. Makarios. "A Reusable Ontology for Fluents in OWL", Technical Report RC23755 (Wo510-142), IBM Research Division, T. Watson Research Center, Yorktown Heights, NY, October 2005.

[2] Matthew Horridge and Sean Bechhofer. "The OWL API: A Java API for Working with OWL 2 Ontologies", The University of Manchester, UK.

[3] D. L. McGuinness and F. VanHarmelen. "OWL Web Ontology Language Overview", W3C Recommendation, February 2004. http://www.w3.org/TR/owl-guide/

[4] http://protege.stanford.edu/

[5] http://www.w3.org

[6] http://protegewiki.stanford.edu/wiki/PluginAnatomy/

[7] http://www.w3.org/TR/owl-time/

[8] http://owlapi.sourceforge.net/documentation.html

[9] http://clarkparsia.com/pellet

[10] A. Artale, and E. Franconi. "A survey of temporal extensions of description logics", Annals of Mathematics and Artificial Intelligence, 30(1-4), 2001.

[11] C. Lutz,F. Wolter, and M. Zakharyaschev. "Temporal description logics: A survey", Proc. TIME08, IEEE Press, 2008.

[12] M. Klein and D. Fensel. "Ontology Versioning for the SemanticWeb", International Semantic Web Working Symposium (SWWS'01), pages 75-92, California, USA, July-August 2001.

[13] Evdoxios Baratis, Euripides G.M. Petrakis, Sotiris Batsakis, Nikolaos Maris and Nikolaos Papadakis. "TOQL: Temporal Ontology Querying Language", 11th International Symposium on Spatial and Temporal Databases (SSTD 2009), July 8-10, 2009, Aalborg, Denmark.

[14] Sotiris Batsakis, Euripides G.M. Petrakis. "Representing Temporal Knowledge in the Semantic Web: the Extended 4d-fluents Approach", 2nd International Workshop on Combinations of Intelligent Methods and Applications (CIMA' 2010), October 27-29, 2010, Arras, France.

[15] Sotiris Batsakis, Euripides G.M. Petrakis. "SOWL:Spatio-temporal Representation, Reasoning and Querying over the Semantic Web", 6th International Conference on Semantic Systems (I-SEMANTICS' 2010), Graz, Austria, September 1-3, 2010.

[16] Franz Baader, Ian Horrocks and Ulrike Sattler. "Description Logics", Chapter 3, Elsevier, 2007.

[17] Michael Kifer, Georg Lausen and James Wu."Logical Foundations of Object-Oriented and Frame-Based Languages". Journal of ACM, May 1995.

[18] Baratis Evdoxios. "TOQL: Querying temporal information in ontologies." Technical Report TR-TUC-ISL-02-2008.

[19] E. Baratis. "TOQL: Querying Temporal Information in Ontologies." Master's thesis, Techn. Univ. of Crete (TUC), Dept. of Electronic and Comp. Engineering, July. 2008.

[20] Ying Ding, Dieter Fensel, Michel Klein, Boris Omelayenko. "The semantic Web: yet another hip?", Elsevier, Data & Knowledge Engineering, 19 December 2001.

[21] A. Johannes Pretorius. "Ontologies - Introduction and Overview". Adapted from: PRETORIUS, A.J., Lexon Visualisation: Visualising Binary Fact Types in Ontology Bases, Chapter 2, Unpublished MSc Thesis, Brussels, Vrije Universiteit Brussel, 2004.

[22] Jennifer Golbeck, Amy Alford, James Hendler. "Organization and Structure of Information using Semantic Web Technologies". Semantic Web and Agents Project, Maryland Information and Network Dynamics Laboratory, University of Maryland, College Park.

[23] Kalliopi Zervanou, Evdoxios Baratis. "TOWL Time-determined ontology based information system for real time stock market analysis." Intelligent Systems Laboratory (InteLLigence), Dept. of Electronic and Computer Engineering, Technical University of Crete (TUC).

[24] Thomas C. Jepsen. "Just What Is an Ontology, Anyway?" IT Pro September/October 2009, Published by the IEEE Computer Society, 1520-9202/09/26.00 2009 IEEE.

[25] Claudio Gutierrez, Carlos Hurtado, and Alejandro Vaisman. "Temporal RDF". Department of Computer Science Universidad de Chile, Department of Computer Science Universidad de Buenos Aires.

[26] Michael Uschold. "Where are the Semantics in the Semantic Web?", Ontologies in Agent Systems workshop, Autonomous Agents Conference, Montreal, June 2001.