Representing information evolving in time in ontologies is a difficult problem to deal with. Temporal relations are in fact ternary (i.e., properties of objects that change in time involve also a temporal value in addition to the object and the subject) and cannot be handled directly by OWL. The standard solution to this problem is to map all temporal relations to a set of binary ones with new (intermediate) classes introduced by the temporal model applied. Nevertheless, ontologies then become complicated and difficult to handle by standard editors such as Protégé (e.g., property restrictions of temporal classes might refer to the new classes rather than to the classes on which they were meant to be defined). It also requires that the user be familiar with the peculiarities of the temporal representation. This is exactly the problem this work is dealing with.

We introduce CHRONOS Ed, a plug-in for Protégé that enables handling of temporal ontologies in Protégé the same way static ontologies are handled. It is implemented as a Tab plug-in for Protégé and can be downloaded from the Web.

**Keywords:** Protégé; SOWL; temporal ontology; reasoning.

1. Introduction

Ontologies offer the means for representing high level concepts, their properties and interrelationships. Dynamic or temporal ontologies, in addition, enable representation of time-evolving information in ontologies. Representation of dynamic features calls for mechanisms that allow uniform representation of the notions of time (and of properties varying in time) within a single ontology. OWL-Time provides a vocabulary for expressing time-related facts. 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 (e.g., events) in time. Existing methods...
for achieving this include, among others, temporal description logics\textsuperscript{1}, concrete domains\textsuperscript{2}, property labeling\textsuperscript{3}, versioning\textsuperscript{4}, named graphs\textsuperscript{5}, reification, N-ary relations\textsuperscript{6} and the 4D-fluents (perdurantist) approach\textsuperscript{6}. All result in complicated ontologies compared with their static counterparts where all relations do not change in time.

Representing temporal information in ontologies resorts to OWL\textsuperscript{c} (Web Ontology Language). However, the syntactic restriction of OWL to binary relations complicates representation of temporal properties, since a property holding for a specific time instant or interval is a relation involving three objects (an object, a subject and a time instant or interval). Binary relations simply connect two instances (e.g., the employee with the company) without any temporal information. Nevertheless, a representation using OWL is feasible, although complicated. In addition, reasoning over temporal information in OWL, as well as maintaining property and data semantics (e.g., cardinality restrictions) are also issues that need to be handled. SOWL (Spatiotemporal OWL)\textsuperscript{7} handles all these issues.

Ontology editors, such as Protégé\textsuperscript{d} are particularly well suited for crafting (creating, editing) static ontologies with binary relations but have no means for dealing with temporal entities and temporal (ternary) relations. As it is common in all known approaches for representing dynamic concepts (such as the N-ary relations or the 4D-fluents approach referred to above), ternary relationships are decomposed into sets of binary relations. 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.

Protégé is probably the most versatile and popular ontology editor, it is free, open-source, supports creation, visualization and manipulation of ontologies, as well as exporting ontologies in various representation formats including OWL. Furthermore, Protégé can be extended by means of a plug-in architecture and a Java-based Application Programming Interface (API) for building tools and applications.

We present CHRONOS Ed, a Tab widget plug-in for the Protégé editor that facilitates handling of temporal ontologies, such as definition of temporal classes and of temporal properties. It is portable and easy to use (i.e., handles temporal ontologies similarly to the way static ontologies are created and handled in Protégé) and does not require the user to be familiar with the peculiarities of the underlying representation model of temporal information (i.e., the N-ary relations model in this work). Temporal ontologies, can still be exported in OWL and handled (i.e., viewed or modified) by standard OWL editors (although much more difficult to handle in this case). CHRONOS Ed interface is consistent with the layout of the

\textsuperscript{1} \texttt{http://www.w3.org/TR/swbp-n-aryRelations/}
\textsuperscript{2} \texttt{http://www.w3.org/TR/owl-features/}
\textsuperscript{3} \texttt{http://protege.stanford.edu/}
default Protégé Tabs. We have made CHRONOS Ed available on the Web\(^6\).

CHRONOS Ed supports adding restrictions on temporal properties (e.g., “an employee can’t work for two different companies at the same time”), classes (e.g., “a company cannot employ more than 20 employees at the same time”) and on individuals. Notice that, if there are inconsistencies within a set of defined temporal relations, normally, these will not be detected by a conventional OWL reasoner (i.e., a reasoner for static ontologies such as Pellet in Protégé) or, an OWL reasoner might not compute all temporal inferences. The problem is that property restrictions defined on temporal classes now refer to the new classes introduced by the N-ary relations model rather than to the classes on which they were meant to be defined. Dealing with such issues calls for reasoners capable of handling temporal information in OWL with the N-ary relations model, such as SOWL \(^7\). SOWL is implemented in SWRL, guarantees soundness, completeness and tractability of reasoning. Any temporal ontology in CHRONOS Ed is handled by Pellet in Protégé and the SOWL reasoner.

Temporal relations in CHRONOS Ed can also be defined as qualitative (i.e., using lexical terms such as “before”, “after” etc.) or as quantitative (i.e., relations described using numerical values such as “10 min after” etc.). The motivation for using a qualitative approach is that it is considered to be closer to the way humans represent and reason about commonsense knowledge. Another motivation is that it is possible to deal with incomplete knowledge. The accompanying SOWL reasoner is also capable of handling qualitative temporal information.

Background knowledge and related research are discussed in Sec. 2. CHRONOS Ed is discussed in Sec. 3. Dealing with cardinality constraints and property restrictions requires particular attention and is discussed separately in Sec. 4. CHRONOS Ed’s architecture and user interface are discussed in Sec. 5 and Sec. 6 respectively. An application of CHRONOS Ed is discussed in Sec. 7, followed by conclusions and issues for further research in Sec. 8.

2. Background and Related Work

In the following, we discuss on models for representing information evolving with time in ontologies.

Temporal Description Logics (TDLs) \(^1\) extend standard description logics (DLs) that form the basis for semantic Web standards with additional constructs such as “always in the past”, “sometime in the future”. TDLs offer additional expressive capabilities over non temporal DLs but they require extending OWL syntax and semantics with the additional temporal constructs. Representing information concerning specific time instants requires support for concrete domains. Concrete Domains \(^2\) relies on the idea of introducing new datatypes and operators in OWL. Notice though, in our work we are opted for an approach that relies on existing

\(^{http://www.intelligence.tuc.gr/prototypes.php}\)
OWL standards and tools. This is a basic design decision in our work. TOWL is an approach combining 4D-fluents with concrete domains but didn’t support qualitative relations, path consistency checking (as this work does) and is not compatible with existing OWL editing, querying and reasoning tools (e.g., Protégé, Pellet, SPARQL).

Versioning suggests that the ontology has different versions as time evolves. When a change takes place, a new version is created. Versioning suffers from several disadvantages: (a) changes even on single attributes require that a new version of the ontology be created leading to information redundancy, (b) searching for events requires exhaustive searches in multiple versions of the ontology, (c) it is not clear how the relation between evolving classes is represented.

Named Graphs represent the temporal context of a property by inclusion of a triple representing the property in a named graph (i.e., a subgraph into the RDF graph of the ontology specified by a distinct name). The default (i.e., main) RDF graph contains definitions of interval start and end points for each named graph, so that a temporal property is represented by the start and end points corresponding to the temporal interval that the property holds. Named graphs are neither part of the OWL specification (i.e., there are not OWL constructs translated into named graphs) nor they are supported by OWL reasoners.

Reification is a general purpose technique for representing $n$-ary relations using a language such as OWL that permits only binary relations. Specifically, an $n$-ary relation is represented as a new object that has all the arguments of the $n$-ary relation as objects of properties. For example, if the relation $R$ holds between objects $A$ and $B$ at time $t$, this is expressed as $R(A,B,t)$. In OWL, this is expressed as a new object with $R, A, B$ and $t$ being objects of properties. Fig. 1 illustrates the relation $\text{WorksFor(Employee, Company, TimeInterval)}$ representing the fact that an employee works for a company during a time interval. The extra class “ReifiedRelation” is created having all the attributes of the relation as objects of

\footnote{http://www.w3.org/TR/owl2-syntax/}

Fig. 1. Example of reification.
properties. Reification suffers mainly from two disadvantages: (a) a new object is created whenever a temporal relation has to be represented (this problem is common to all approaches based on OWL) and (b) offers limited OWL reasoning capabilities. Because relation $R$ is represented as the object of a property, OWL semantics over properties (e.g., inverse properties) are no longer applicable (i.e., the properties of a relation are no longer associated directly with the relation itself).

The 4D-fluent (perdurantist) approach shows also how temporal information and the evolution of temporal concepts can be represented in OWL. To add the time dimension to an ontology, classes $\text{TimeSlice}$ and $\text{TimeInterval}$ with properties $\text{TimeSliceOf}$ and $\text{TimeInterval}$ are introduced. Properties having a temporal dimension are called fluent properties and connect instances of class $\text{TimeSlice}$ (e.g., properties “employs” and “worksFor” in Fig. 2). Dotted arrows in Fig. 2 represent object properties while, solid lines represent “isA” relations. Class $\text{TimeSlice}$ is the domain class for entities representing temporal parts (i.e., “time slices”) and class $\text{TimeInterval}$ is the domain class of time intervals. Time instances and time intervals are represented as instances of a $\text{TimeInterval}$ class. A temporal property does not hold between the static entities but between their temporal parts. The time slices of an entity have a specific lifetime, that is the time interval of the relation they participate in.

Fig. 2. Example of 4D-fluents representation.

Fig. 3. Example of N-ary relations representation.
The N-ary relations approach suggests representing an n-ary relation as two properties each related with a new object (rather than as the object of a property, as reification does). This approach requires only one additional object for every temporal relation. A temporal property between two individuals (e.g., “Employee works for Company”) holds as long as that event endures. The n-ary property is represented as a class rather than as a property. Instances of such classes correspond to instances of the relation. Additional properties introduce additional binary links to each argument of the relation. For properties that change in time, their domains and ranges have to be adjusted taking into account the classes of intermediate objects representing the relation. For example, the worksFor relation in Fig. 3 is no longer a relation having as object an individual of class Company and subject of class Employee as they are now related to the new object EmploymentEvent. The new domain is the union of the old domain with the class that represents the N-ary property (Event class). Likewise, the new range is a union of the old one with the Event class.

2.1. SOWL

SOWL 7 is an ontology framework for representing and reasoning over spatio-temporal information in OWL. Building-upon well established standards of the semantic Web (OWL 2.0, SWRL) SOWL enables representation of static as well as of dynamic spatio-temporal information. Both 4D-fluents and the N-ary models for the representation of temporal information are supported. The user is opted between a point-based and an interval-based representation. Representing both qualitative temporal and spatial information (i.e., information whose temporal or spatial extents are unknown such as “left-of” for spatial and “before” for temporal relations) in addition to quantitative information (i.e., where temporal and spatial information is defined precisely) is a distinctive feature of SOWL.

A temporal relation can be one of the 13 pairwise disjoint Allen’s relations of Fig. 4. Definitions for temporal entities (e.g., instants and intervals) are provided by incorporating OWL-Time into the ontology. Each interval (which is an individual of the ProperInterval class) is related with two instants (individuals of the Instant class) that specify its starting and ending points using the hasBeginning and hasEnd object properties respectively. In turn, each Instant can be related with a specific date represented using the concrete dateTime datatype. One of the before, after or equals relations may hold between any two temporal instants with the obvious interpretation. In fact, only relation before is needed since relation after is defined as the inverse of before and relation equals can be represented using the sameAs OWL keyword applied on temporal instants. In this work, for readability we use all three relations. Notice also that, property before may be also qualitative when holding between time instants or intervals whose values or end points are not specified. This way, we can assert and infer facts beyond the ones allowed when only instants or intervals with known values (e.g., dates) or end-points
CHRONOS Ed: A Tool for Handling Temporal Ontologies in Protégé

are allowed.

Relation | Inverse Relation
---|---
Before(i,j) | After(j,i)
Meets(i,j) | MetBy(j,i)
Overlaps(i,j) | OverlappedBy(j,i)
Starts(i,j) | StartedBy(j,i)
During(i,j) | Contains(j,i)
Finishes(i,j) | FinishedBy(j,i)
Equals(i,j) 

Fig. 4. Allen’s temporal relations.

Reasoning in SOWL is realized by introducing a set of SWRL rules operating on temporal relations. The reasoner is capable of inferring new relations and checking their consistency, while retaining soundness, completeness, and tractability over the supported sets of relations. Reasoners that support DL-safe rules such as Pellet can be employed for inference and consistency checking over temporal relations. The SOWL reasoner is fully incorporated within the CHRONOS Ed.

3. CHRONOS Ed

CHRONOS Ed is a Tab plug-in for Protégé version 4.x that facilitates creating and editing of temporal ontologies in OWL 2.0. The user has the option to create a new temporal ontology (i.e., starting with an empty ontology) or, convert an existing OWL ontology to temporal. In the later case, the user can select classes to be converted to temporal (in which cases all data properties, object properties and individuals associated with this class are also converted to temporal). Each class is converted to temporal following the N-ary relations approach discussed in Sec. 2. However, the user need not be familiar with details of the model or of the conversion mechanism, nor this information is visible to the user (i.e., CHRONOS Ed displays temporal information similarly to static). In order to view all the details of the resulting temporal representation (including any intermediate classes added by the N-ary relations model) the user can select to view the ontology within the standard Protégé Tab. It is always possible to switch between the two viewing modes at any

---

\(^{8}\)http://www.w3.org/Submission/SWRL/

\(^{b}\)http://clarkparsia.com/pellet/
time (i.e., the standard Protégé Tab and CHRONOS Ed) and also continue working with the ontology with any mode. Nevertheless, working with temporal ontologies within the standard (static) Tab, although feasible, requires good knowledge of the N-ary model and is not recommended.

CHRONOS Ed allows the user to create a new individual related with a temporal property, add temporal object or data property assertions to existing individuals, or edit the time intervals during which temporal property assertions hold. The user does not have to intervene with the intermediate objects or with their relationships, making the manipulation of temporal relations between individuals as easy as the manipulation of the static ones. Static object or data properties can be created or edited as usual but, in addition, they can be easily converted to temporal. New temporal object or data properties can be added between individuals.

Temporal properties can be expressed qualitatively or quantitatively by specifying specific temporal values for time instants or intervals. Cardinality constraints and property restrictions on temporal properties can be checked for consistency as well or, new relations (static or temporal) can be inferred from existing ones simply by running Pellet (which in turn calls for the underlying reasoner implementation in SWRL of SOWL).

In order to implement the changes suggested by the N-ary Relations model, the following new objects are introduced into the ontology.

- **Event**: The class that represents the N-ary property.
- **during**: An object property that relates the event to the time interval during which it holds.
- **participatesIn**: An object property that relates the individuals that participate in an event, to that specific event individual. Object properties that are converted to temporal become sub-properties of this property.
- **overlaps**: An object property that relates two time intervals. This property implies that those time intervals, in some way overlap to each other.

### 3.1. Static to Dynamic

Converting an ontology from static to dynamic requires a lot of changes. Different types of entities (e.g., object properties, data properties) are handled in a different way. Particular emphasis is given on maintaining the semantics of these entities, when they are converted to temporal. In the following, we describe the way CHRONOS Ed handles different kinds of OWL entities:

#### 3.1.1. Object Properties

The representation of a temporal object property, according to the N-ary Relations model, suggests that an intermediate object (instance of the Event class) is introduced between the subject and the object of the static object property. This object appears as both, an object and a subject in two triples, whose predicate is the spe-
cific object property and together represent the temporal relationship. To make this possible, the static property’s domain and range have to be modified. The dynamic property’s domain/range will be the union of the static property’s domain/range and the Event class. Moreover, the converted object property is made a sub-property of the participatesIn property. The Event class is related to the time interval class (Interval) with the object property during. In the example of Fig. 3 the domain of the static object property worksFor is class Employee. After conversion, the domain of the dynamic object property will be the anonymous class (Employee OR Event), that represents the union of the classes Event and Employee. The static property’s range is modified respectively, from Company to (Company OR Event).

### 3.1.2. Data Properties

Data properties are handled by CHRONOS Ed similarly to object properties. The dynamic data property’s domain is the union of the static property’s domain and the Event class. The main difference with the case of object properties is that the range cannot be the union of a data type and the Event class\(^1\). To overcome this problem, we create an object property named by the data property followed by “OP”. This object property will relate the static data property’s domain to the Event class. This is also made a sub-property of the participatesIn object property. The data property with the modified domain, will connect the event to the data type. As in object property conversion, the Event is related to the Interval with the during object property. Fig. 5 illustrates an example showing how the temporal data property hasPrice is converted to temporal (e.g., in the case where a produce the price of a product changes later in time).

\[^1\text{In OWL DL there is a distinction between OWL Data Types and OWL Classes. OWL Full allows the union of data types and classes. Protégé and OWL API support OWL DL in full but not OWL Full expressiveness.}\]
3.1.3. Individuals

Individuals represent objects in the domain of interest. For example, “John” is an individual of the class “Person”. The statement “John has lived from 1920 to 1998” does not require a temporal property for its representation. However, when a property evolves with time, such as in the statement “John has lived in Athens from 1950 to 1985”, the property livesIn is a temporal property that holds during a specific time interval. John still lived after the year 1985, but in a different place. In this case, temporal relations are defined in terms of relations between individuals rather than temporal individuals (i.e., temporal properties). When a property is converted to temporal, all the triples that contain this property are converted too. For each triple in the ontology, a new instance of the Event class is created and introduced between the subject and the object, as explained in subsections 3.1.1 and 3.1.2. This event individual is connected to a TimeInterval instance with the during object property. The TimeInterval individual is related to two Instant individuals, one that represents the starting point of the interval and one that represents the ending point of the interval. Each of these Instant individuals are connected to a dateTime data type with the data property inXSDDateTime. Fig. 6 illustrates the individuals and their relations for a temporal object property “worksFor” holding during a certain temporal interval.

![Diagram of temporal object property](image)

Fig. 6. Example of a temporal object property between individuals.

3.1.4. Classes

Classes provide an abstraction mechanism for grouping resources with similar characteristics. Every OWL class is associated with a set of individuals, called the class extension. The individuals in the class extension are called the instances of the class.
A class has an intensional meaning (the underlying concept) which is related but not equal to its class extension. Thus, two classes may have the same class extension, but still be different classes. Similarly to individuals, classes cannot be converted to temporal. So, when we use the term “convert a class”, we refer to the object and data properties that relate to this class. More specifically, when a class is converted to temporal in CHRONOS Ed the entities that are affected are: (a) the object and data properties that relate members of the selected class, (b) the object and data properties where this class appears as a Domain and (c) the restrictions where one of these object or data properties appear in. Summarizing “class conversion” is just a more convenient way to convert multiple object and data properties together to temporal.

3.2. Dynamic to Static

CHRONOS Ed facilitates conversion of entities from dynamic to static. This function enables easy recovering from errors during ontology construction or modifying existing ontologies to match user needs. The procedure followed is the reverse of that described in sec. 3.1. In the following, for ease of discussion, conversion from dynamic to static is discussed separately for each entity type.

3.2.1. Object Properties

The intermediate object that represents the ternary relationship is removed and the subject is directly connected with the object property to the object. The property’s temporal domain is changed back to static\(^1\) by removing the Event class. Moreover, the property participatesIn is no longer a super-property of the converted object property.

3.2.2. Data Properties

Data properties are handled similarly to object properties. The intermediate object is removed along with the object property that was introduced to connect it to the property’s subject (as described in 3.1.2). The subject is directly connected with the data property to the data type. The Event class is removed from the property’s domain.

3.2.3. Individuals

When a property is converted to static, all the temporal triples that contain it are converted too. All the subject individuals that participate in temporal triples are directly connected to the respective object individuals. The instances of the Event

\(^1\)As described in sec. 3.1.1 the domain/range of a temporal object property is the union of a class with the Event class.
class that used to represent instances of the ternary relationship are removed from the ontology.

3.2.4. Classes

As in the case of conversion to temporal, the conversion of a class to static is a convenient way to convert multiple objects to static. More specifically, the objects that are affected are: (a) the object and data properties that relate members of the selected class, (b) the object and data properties where this class appears as a Domain, (c) the restriction where one of these object and data properties participate in.

3.3. Deletions

An instance of a temporal property is represented by an Event individual. It is related to the property’s subject, object and the interval during which the temporal property holds true. By deleting a temporal property, all event individuals representing instances of this property are deleted too. CHRONOS Ed facilitates deletion of temporal properties by removing automatically all Event individuals representing instances of the deleted property.

4. Dealing with Cardinality Constraints and Property Restrictions

OWL classes are described through ‘class descriptions’. A property restriction is a special kind of class description. It describes an anonymous class, namely a class of all individuals that satisfy the restriction. OWL distinguishes between two kinds of property restrictions: value constraints and cardinality constraints.

4.1. Value Constraints

4.1.1. owl:allValuesFrom

It is a built-in OWL property that links a restriction class to either a class description or a data range. It describes a class of all individuals for which all values of the property are either members of the class extension of the class description or are data values within the specified data range. In the case where the property is temporal, the form of the constraint is different. The restriction is used to describe a class of all individuals for which all values of the property under consideration are those members of the Event class that are connected with the concerned property to either members of the class extension of the class description or are data values within the specific data range. For example, a restriction on a class Company could be

employs only Employee

If the object property employs is temporal, the restriction becomes:
4.1.2. owl:someValuesFrom

This constraint links a restriction class to a class description or a data range. A restriction containing an owl:someValuesFrom constraint describes a class of all individuals for which at least one value of the property concerned is an instance of the class description or a data value in the data range. In the case of a temporal property, this constraint describes a class of individuals for which, at least one value of the property is an Event individual which is connected using the property with an instance of the class description or a data value in the data range. For example, the following defines a restriction on class Company:

\[
\text{employs some Employee}
\]

If the object property employs is temporal, the restriction becomes:

\[
\text{employs some (Event and (employs some Employee))}
\]

4.1.3. owl:hasValue

This constraint links a restriction class to a value \(V\), which can be either an individual or a data value. A restriction containing a owl:hasValue constraint describes a class of all individuals for which the property concerned has at least one value semantically equal to \(V\) (it may have other values as well). The temporal form of this restriction describes a class of all individuals for which the property concerned has at least one value semantically equal to \(V\), for each event that these individuals participate in. This temporal constraint is applied with the addition of an SWRL rule that to the ontology. For example, an owl:hasValue restriction on a class Company is:

\[
\text{employs value John}
\]

where “John” is an individual of the class Employee. The SWRL rule that applies this temporal constraint is:

\[
\text{Company(?x)\land participatesIn(?x, ?e)\land Event(?e) \rightarrow employs(?e, John)}
\]

meaning that for each event that an individual of the class Company participates in, that company individual is also related to the Employee ‘John’ with the temporal object property employs.
4.1.4. Cardinality Constraints

In OWL, like in RDF\(^k\) (Resource Description Framework), it is assumed that any instance of a class may have an arbitrary number (zero or more) of values for a particular property. To make a property required (at least one), or to allow only a specific number of values for that property, or to insist that a property must not occur, cardinality constraints can be used. OWL provides three constructs for restricting the cardinality of properties locally within a class context.

4.1.5. owl:maxCardinality

This constraint links a restriction class to a data value belonging to the value space of the XML Schema datatype nonNegativeInteger. A restriction containing an owl:maxCardinality constraint describes a class of all individuals that have at most \(N\) semantically distinct values (individuals or data values) for the property concerned, where \(N\) is the value of the cardinality constraint. Syntactically, the cardinality constraint is represented as an RDF property element with the corresponding rdf:datatype attribute. In CHRONOS Ed, maxCardinality constraint when interpreted as temporal property describes a class of all individuals that have at most \(N\) semantically distinct values at the same time, for the property concerned. This temporal constraint is applied with the addition of a SWRL rule to the ontology. For example, the owl:maxCardinality constraint in Manchester syntax is:

\[
\text{employs max 2 Employee}
\]

It implies that an individual of the class Company cannot be related to more than two individuals of the class Employee with the object property employs. The SWRL rule that is added to the ontology to apply the temporal version of the constraint is:

\[
\begin{align*}
\text{Event(?e0) \land Event(?e1) \land Event(?e2) \land Company(?x) \land Employee(?y0) \\
\land Employee(?y1) \land Employee(?y2) \land during(?e0, ?i0) \land during(?e1, ?i1) \\
\land during(?e2, ?i2) \land overlaps(?i0, ?i1) \land overlaps(?i0, ?i2) \land overlaps(?i1, ?i2) \land employs(?e0, ?y0) \land employs(?e1, ?y1) \land employs(?e2, ?y2) \\
\land employs(?x, ?e0) \land employs(?x, ?e1) \land employs(?x, ?e2) \land DifferentFrom(?y0, ?y1) \land DifferentFrom(?y0, ?y2) \land DifferentFrom(?y1, ?y2)
\rightarrow \text{Nothing(?x)}
\end{align*}
\]

This means that if there are three different individuals of the class Employee that relate through the temporal property employs to the same individual of the class Company and the time intervals associated with those temporal properties pairwise overlap.

\(^k\text{http://www.w3.org/RDF/}\)
4.1.6. owl:minCardinality

This constraint links a restriction class to a data value belonging to the value space of the XML Schema datatype nonNegativeInteger. A restriction containing an owl:minCardinality constraint describes a class of all individuals that have at least \( N \) semantically distinct values (individuals or data values), where \( N \) is the value of the cardinality constraint. Syntactically, the cardinality constraint is represented as an RDF property element with the corresponding rdf:datatype attribute.

In CHRONOS Ed, the temporal version of this constraint describes a class of all individuals that are connected to at least \( N \) members of the Event class with the property at hand. The event individuals have at least one value for the property concerned. OWL adopts the open world assumption, thus if a member of a class restricted with a minCardinality constraint has less than \( N \) distinct values for the concerned property, no inconsistency will result. An example of an owl:minCardinality constraint would be:

\[
\text{employs min } 2 \text{ Employee}
\]

The temporal version of that minCardinality constraint is:

\[
\text{employs min } 2 \text{ (Event and (employs some Employee))}
\]

Actually this interpretation of the constraint does not imply that the individuals of the class Company have at least 2 employees at the same time, but just that they have two employees during their existence, connected with the temporal property employs. This is a less strict application of the right constraint which would require all the company individuals to have at least 2 employees at the same time. This was the only temporal interpretation of the minCardinality constraint that we were able to implement, given the current expressiveness of property restrictions and SWRL rules.

4.1.7. owl:cardinality

This constraint links a restriction class to a data value belonging to the range of the XML Schema datatype nonNegativeInteger. A restriction containing an owl:cardinality constraint describes a class of all individuals that have exactly \( N \) semantically distinct values (individuals or data values) for the property at hand, where \( N \) is the value of the cardinality constraint. Syntactically, the cardinality constraint is represented as an RDF property element with the corresponding rdf:datatype attribute. The temporal version of this constraint in CHRONOS Ed describes a class of all individuals that are related to Event individuals with the property concerned and those event individuals have exactly \( N \) semantically distinct values for the property concerned. An example of an owl:cardinality constraint would be:

\[
\text{employs exactly } 2 \text{ Employee}
\]
It implies that an individual of the class Company can be related to exactly two individuals of the class Employee with the object property employs. The SWRL rule that would be added to the ontology to apply the temporal version of the constraint would be:

\[
\text{Event}(?e0) \land \text{Event}(?e1) \land \text{Event}(?e2) \land \text{Company}(?x) \land \text{Employee}(?y0) \\
\land \text{Employee}(?y1) \land \text{Employee}(?y2) \land \text{during}(?e0, ?i0) \land \text{during}(?e1, ?i1) \\
\land \text{during}(?e2, ?i2) \land \text{overlaps}(?i0, ?i1) \land \text{overlaps}(?i0, ?i2) \land \text{overlaps}(?i1, ?i2) \land \text{employs}(?e0, ?y0) \land \text{employs}(?e1, ?y1) \land \text{employs}(?e2, ?y2) \\
\land \text{employs}(?x, ?y0) \land \text{employs}(?x, ?y1) \land \text{employs}(?x, ?y2) \\
\land \text{employs}(?x, ?e1) \land \text{employs}(?x, ?e2) \\
\land \text{DifferentFrom}(?y0, ?y1) \land \text{DifferentFrom}(?y0, ?y2) \land \text{DifferentFrom}(?y1, ?y2) \\
\rightarrow \text{Nothing}(?x)
\]

This rule would result in inconsistency if an individual of the class Company was related to more than two instances of the class Employee, with the temporal object property employs, but not if it was related to only one Employee individual. In that case, the behavior of the temporal versions of the maxCardinality and the cardinality constraint coincides.

### 4.2. Global Cardinality Constraints on Properties

#### 4.2.1. owl:FunctionalProperty

A functional property is a property that can have only one (unique) value \( y \) for each instance \( x \), i.e. there cannot be two distinct values \( y_1 \) and \( y_2 \) such that the pairs \( (x, y_1) \) and \( (x, y_2) \) are both instances of this property. Both object properties and datatype properties can be declared as “functional”. For this purpose, OWL defines the built-in class `owl:FunctionalProperty` as a special subclass of the RDF class `rdf:Property`. A temporal functional property can have only one value in each time interval for which the property holds. This is realized by adding a SWRL rule to the ontology. For the temporal property `employs` to be functional, the rule is:

\[
\text{Event}(?e1) \land \text{Event}(?e2) \land \text{during}(?e1, ?i1) \land \text{during}(?e2, ?i2) \land \text{overlaps}(?i1, ?i2) \land \text{employs}(?e1, ?y1) \land \text{employs}(?e2, ?y1) \\
\land \text{employs}(?x, ?e1) \land \text{employs}(?x, ?e2) \rightarrow \text{SameAs}(?y1, ?y2)
\]

#### 4.2.2. owl:InverseFunctionalProperty

If a property is declared to be inverse-functional, then the object of a property statement uniquely determines the subject (some individual). More formally, if we state that \( P \) is an `owl:InverseFunctionalProperty`, then this asserts that a value \( y \) can only be the value of \( P \) for a single instance \( x \), i.e. there cannot be two distinct instances \( x_1 \) and \( x_2 \) such that both pairs \( (x_1, y) \) and \( (x_2, y) \) are instances of \( P \). When a temporal property is inverse-functional the object uniquely determines the subject for each time instant: there can be two instances \( x_1, x_2 \) such that \( (x_1, y, \).
interval1) and (x2, y, interval2) are instances of P as long as the interval1 and interval2 do not overlap. The SWRL rule to make the temporal property worksFor inverse-functional is:

\[ \text{Event(?e1)} \land \text{Event(?e2)} \land \text{during(?e1, ?i1)} \land \text{during(?e2, ?i2)} \land \text{overlaps(?i1, ?i2)} \land \text{worksFor(?e1, ?y)} \land \text{worksFor(?e2, ?y)} \land \text{worksFor(?x1, ?e1)} \land \text{worksFor(?x2, ?e2)} \rightarrow \text{SameAs (?x1, ?x2)} \]

4.3. Negative property assertions

A negative property assertion is a feature introduced by OWL 2.0 and applies on individuals, not allowing them to have a specific value (individual or data value). More formally, a negative property assertion between the individual \(x\), the value \(y\), connected with the property \(P\) asserts that there cannot be an instance of the property such as \(P(x,y)\). A temporal negative property assertion restricts an individual \(x\) in a way that it cannot be connected with a temporal property \(P\) to a specific value \(y\) during a time interval \(interval1\), thus the \(P(x,y, interval1)\) cannot be an instance of the temporal property \(P\). The SWRL rule added to the ontology would be:

\[
\text{Event(?e) } \land \text{during(?e, ?i) } \land \text{overlaps(?i, interval1) } \land \text{employs(?e, John)} \\
\land \text{employs(Company1, ?e)} \\
\rightarrow \text{Nothing(Company1)}
\]

This rule forbids the individual “Company1” to employ “John” during any time interval that somehow overlaps with the specified time interval “interval1”.

4.4. Transitive Properties

A property defined as transitive means that, if a pair \((x, y)\) is an instance of \(P\) and the pair \((y, z)\) is an instance of \(P\) then, we can infer that the pair \((x, z)\) is also an instance of \(P\). The instances of the properties are considered to hold for a specific time interval. If a temporal property \(P\) is transitive and \((x, y, interval1)\) is an instance of \(P\) and \((y, z, interval2)\) is an instance of \(P\), then we can infer that \((x, z, interval1 \cap interval2)\) is also an instance of \(P\). The SWRL expressiveness though does not allow the creation of instances of classes. Thus, the creation of such a rule is not possible. In our implementation the transitivity between instances of a temporal property takes place only if those instances hold for same time intervals. The SWRL applying that effect on the temporal property \(\text{worksFor}\), would be:

\[
\text{Event(?e1) } \land \text{Event(?e2)} \land \text{during(?e1, ?i1)} \land \text{during(?e2, ?i2)} \land \text{worksFor(?e1, ?y)} \land \text{worksFor(?e2, ?z)} \land \text{worksFor(?x, ?e1)} \land \text{worksFor(?y, ?e2)} \\
\land \text{intervalEquals(?i1, ?i2)} \land \text{DifferentFrom (?y, ?z)} \\
\rightarrow \text{worksFor(?x, ?e2)}
\]
5. Architecture

Fig. 7 illustrates the architecture of CHRONOS Ed. It consists of several modules, the most important of them being the following:

(i) **Chronos Tab**: It is the main tab of CHRONOS Ed plug-in. Its main function is to display the ontology. It provides views for all OWL Entities of the ontology. The user can select an OWL entity to work on which is fed to the “Chronos View Component”.

(ii) **Chronos View Component**: Displays the attributes of the selected entity. Changes on a static OWL entity are applied to the ontology directly via the Ontology Manager (an interface for loading, creating and accessing ontologies) provided by OWL API. If the entity to be changed is a temporal one, it will be forwarded to the “Time Factory”.

(iii) **Time Factory**: This module is responsible for implementing the conversion of static OWL entities to dynamic. As explained in the previous sections this involves creation of additional OWL axioms. More specifically, this module implements:

- Conversion of an object property to temporal,
- Conversion of a data property to temporal,
- Creation of temporal object property assertions on individuals,
- Creation of temporal data property assertions on individuals,
- Creation of temporal negative object property assertions on individuals,
- Creation of temporal negative data property assertions on individuals,
• Creation of temporal property restrictions.

The changes made to the ontology are displayed back to the “Chronos View Component”. CHRONOS Ed has been implemented using OWL API\(^1\).

6. User Interface

As a use case, the following example will demonstrate the use of the Tab plug-in and the conversion from static to dynamic of an ontology containing classes *Company* and *Employee* and their relation *employs*. Initially, object property *employs* has domain the class *Company* and range the class *Employee*. The first time CHRONOS Ed tab is selected, it checks if the active ontology is merged with the OWL-Time ontology. If it is not, a pop-up window will appear, prompting the user to merge the ontology with the OWL-Time ontology. In the case where the user selects not to merge the active ontology with OWL-Time, CHRONOS Ed will only add to the ontology the OWL entities that are required for the representation of temporal relationships, but will not support reasoning over the temporal concepts of the ontology (i.e., inference of new entities or consistency checking) as a result of this decision.

CHRONOS Ed provides four different views on the left of the Tab screen, one for each entity type, referred to as “Class hierarchy”, “Object property hierarchy”, “Data property hierarchy” and “Individuals by type”. “Chronos View Component” is the main CHRONOS Ed’s panel. Depending on the type of the OWL Entity that has been selected, “Chronos View Component” displays:

- An object property’s characteristics and description,
- A data property’s characteristics and description,
- An individual’s description and its property assertions,
- A class’ restrictions, superclasses and equivalent classes.

Fig. 8 illustrates the layout of the views in CHRONOS Ed.

The user may then select to convert a class, an object property or a data property to temporal. A dialog window pops-up notifying the user about the triples that will be affected by this conversion. In this window the user is prompted to provide the interval during which all the converted triples hold. If this field is left blank, then each triple will be related to an unknown time interval. This dialog window is illustrated in Fig. 9. In this example there are three individuals of class *Company* connecting instances of class *Employee* with the *employs* object property. These object property assertions will be converted to temporal too.

Fig. 10 illustrates object property *employs* after it has been has been converted to temporal. Notice that, using the “Chronos View Component”, the changes made to the property according to the N-ary relations model, are hidden from the user.

\(^1\)http://owlapi.sourceforge.net
A graphical representation of the model is displayed at the bottom of the view component.

To specify the time intervals during which the temporal triples hold, in the “Individuals by type” view the user selects the individual that appears as the object in the triple of interest. The assertions associated with that individual are displayed in the respective panel (as illustrated in Fig. 11) and can be edited.
A dialog window, called “Individual Wizard”, pops-up and guides the user on the specification of the time interval. The user can either select one of the existing intervals, or create a new one. An interval can be defined either quantitatively (by specifying its starting and end points) or qualitatively by temporal intervals whose either the starting or end points are left unspecified. Alternatively, a temporal interval can be defined qualitatively using natural language terms (e.g., “before”, ...)
“after”) to define its relation with other temporal intervals. The “Individual Wizard” window is illustrated in Fig. 12.

Fig. 12. Creation of time interval.

A user may decide to add a temporal property restriction to Company class, on object property employs. In the following example a maximum 2 cardinality restriction is introduced, implying that a company cannot employ more than 2 employees at the same time. The creation of the temporal restriction using CHRONOS Ed is similar to the creation of a restriction on the default Protégé tab: First, the class Company from the “Class hierarchy view” is selected. In the “Class restrictions” panel, the user adds a superclass on the selected class and the “Class restriction editor” window is displayed. Then, the employs object property is selected as the restricted property, class Employee as the restriction filler, “Max” as the restriction type and cardinality number is set to 2. Fig. 13 illustrates the “Class restriction editor”.

Assume that the user decides to convert the employs object property back to static. In the “Object Property Hierarchy” view, the user selects the employs object

\[\text{Company} \rightarrow \text{Employe}e\]

The list of the restricted properties contains only the properties that have been converted to temporal.
A dialog window is displayed, illustrating temporal triples that will be affected by this conversion. Fig. 14 illustrates the “Affected Triples” dialog window.

Consistency checking on the derived temporal ontology can be applied using the built-in Pellet reasoner of Protégé. For details on the algorithms for converting static ontologies to temporal and for instructions on using CHRONOS Ed the reader is referred to 9.
7. SybillaTUC

Bipolar Disorder (BD) is a severe and recurrent mental illness related with high morbidity and evolves constantly in time. SybillaTUC \(^\text{10}\) is a prototype decision support system implementing best practice treatment guidelines for patients suffering from BD \(^\text{11,12,13,14}\). Its purpose is to alert clinicians on the possibility of a crucial incident (e.g., on the transition of the patient’s state from stable to unstable) according to a breakthrough depressive episode, a scenario that develops when a patient shows depressive symptoms during pharmaceutical treatment with lithium.

SybillaTUC represents patient’s and disease information as well as expert’s knowledge by means of an ontology. CHRONOS Ed provides the means for describing the temporal concepts in the ontology. Hence, the developers of SybillaTUC do not have to deal with the representation of temporal concepts, or with the peculiarities of the underlying temporal representation model. Decision making implements a reasoning system by means of SWRL which applies expert’s knowledge and clinical guidelines on patient’s data.

Fig. 15. SybillaTUC ontology.
Fig. 15 illustrates the temporal ontology representing both static (in grey span) and temporal concepts. Main classes (with the obvious meaning) are: PatientCase representing the current patient state and is associated with classes Patient record, PatientHistory, Therapy, Diagnosis, FunctionalTests, PatientState and Recommendation. Class PatientState is associated with class EpisodeInfo providing access to information about the type and the severity of an episode when a patient is in an episode state. Class FunctionalTests is associated with class StandardTests allowing the system to reason on whether a clinical test has optimal value or not. Class Therapy and PatientHistory are associated with class Medicine and this allows for reasoning using information on medication that a patient may receive or has received in the past. As an example, when a patient who is not diagnosed with rapid-cycling (i.e., with four or more mood episodes within a year) displays depressive symptoms while his serum lithium test value is optimal, the following rule issues a recommendation to the clinician to add antidepressant or second mood stabilizer.

\[
\text{PatientCase} \land \exists \text{caseIncludesDiagnosis.rapidCycling} = \text{false} \land \\
(\exists \text{caseIncludesFunctionalTests.functionalTestsValue} \geq 0.6 \land \\
\lnot < f > \exists \text{caseIncludesPatientState.state} = \text{true}) \rightarrow \\
\text{Recommendation "Add antidepressant OR Second mood Stabilizer"}
\]

8. Conclusions and Future Work
We introduce CHRONOS Ed a Tab plug-in for Protégé editor that facilitates the creation and editing of temporal OWL 2.0 ontologies. The temporal concepts as well as the properties that evolve over time are represented by means of the N-ary Relations model. CHRONOS Ed allows for defining restrictions on temporal properties and also suggests the meaning of restrictions defined on temporal properties. The user does not have to be familiar with the peculiarities of the temporal representation model, thus making the manipulation of temporal entities as easy as if they were static. Enhancing CHRONOS Ed with querying support on temporal ontologies is an interesting issue for future work.

References


