Chapter 11
tOWL: Integrating Time in OWL

Flavius Frasincar, Viorel Milea, and Uzay Kaymak

Abstract The Web Ontology Language (OWL) is the most expressive standard language for modeling ontologies on the Semantic Web. In this chapter, we present the temporal OWL (tOWL) language: a temporal extension of the OWL DL language. tOWL is based on three layers added on top of OWL DL. The first layer is the Concrete Domains layer, which allows the representation of restrictions using concrete domain binary predicates. The second layer is the Time Representation layer, which adds time points, intervals, and Allen's 13 interval relations. The third layer is the Change Representation layer which supports a perdurantist view on the world, and allows the representation of complex temporal axioms, such as state transitions. A Leveraged Buyout process is used to exemplify the different tOWL constructs and show the tOWL applicability in a business context.

11.1 Introduction

In its role as reference system, time is, beyond any doubt, one of the most encountered dimensions in a variety of domains. Naturally, dealing with time has been, and continues to be, one of the major concerns in different fields, including knowledge representation.

When including time in a knowledge representation language, one can choose to model linear time or branching time. Linear time uses a single line of time (one future), while branching time employs many time lines (possible futures). Based on the inclusion of time representations in the language, we distinguish between explicit and implicit approaches. In an explicit approach, time is part of the language, and in an implicit approach, time is inherent in the ordering of states. For an explicit temporal representation, we differentiate between time points and time

Flavius Frasincar, Viorel Milea, and Uzay Kaymak
Erasmus University Rotterdam, Burgemeester Oudlaan 50, 3062 PA Rotterdam, the Netherlands
e-mail: {frasincar, milea, kaymak}@ese.eur.nl

intervals. Also, the explicit representations can further be defined using an internal or an external view on time. In an external view, an individual has different states at different moments in time, and in the internal view, an individual is seen as collection of different parts, each one holding at a certain moment in time. In other words, the external view uses an enduratist view on the world, and the internal view uses a perdurantist view on the world.

For modeling time one has at least two options to consider: valid time and transaction time. Valid time denotes the time during which the data is true in the modeled world. Transaction time represents the time at which the data was stored. Another differentiation pertains to whether we model relative time as "next week" or absolute time as "24 May 2009 15:00 CEST".

The considerable and ever-increasing volume of data present on the Web today motivates a need to move from free-text representations of data to semantically rich representations of information. Endeavors in this direction are being undertaken under a common denominator: the Semantic Web [4]. The state-of-the-art tools and languages provided under this umbrella, such as RDF(S) [5, 11] and OWL [3], go beyond the Web and provide the means for data sharing and reuse outside this platform, i.e., in the form of semantic applications. Despite the omnipresence of time in any Web knowledge representation, the current RDF(S) and OWL standards do not support at language level temporal representations, failing thus to provide a uniform way of specifying and accessing temporal information.

Previous attempts [6] to represent time and change relate to RDF extensions that are able to cope only to a limited extent with the semantics of temporal representations. Also, these languages are difficult to use in practice as they do not have an RDF/XML serialization. Other solutions are based on proposing ontologies [9, 17] for modeling time and/or change. These approaches also present shortcomings when modeling temporal semantics as they are bound to the OWL expressivity power.

In this chapter, we present a temporal ontology language, i.e., tOWL, addressing the current limitations on representing temporality on the Semantic Web. We model valid time using an absolute time representation. By employing a similar approach, one can model also transaction time, and by determining the context of temporal expressions, it is also possible to use relative time by converting it internally to an absolute representation. Our language is able to represent *linear time* using an *explicit time* specification. It supports both *time points* and *time intervals*, and adopts an *internal* (*predurantist*) *view* on the world. The proposed language builds upon OWL, the most expressive Semantic Web standard in knowledge representation. The current contribution is focused around employing the tOWL language for the representation of business processes, the Leveraged Buyout process.

Section 11.2 presents the concrete domains and fluents notions needed in order to understand the tOWL language. In Sect. 11.3 we describe the tOWL language by providing its layered architecture, and the OWL schema representation in RDF/XML of its vocabulary. After that, in Sect. 11.4 we present the TBox and ABox of the tOWL ontology for a Leveraged Buyout example. Section 11.5 compares the related work with the tOWL approach. Last, in Sect. 11.6 we present our concluding remarks and identify possible future work.

11.2 Preliminaries

The language proposed in this paper builds on previous work on concrete domains in description logics, and fluents representation in Semantic Web languages. In Subsect. 11.2.1 we present a scheme for integrating concrete domains and their predicates in description logics. After that, in Subsect. 11.2.2 we present the 4D Fluents approach for modeling change in OWL DL.

11.2.1 Concrete Domains

Current DL-based languages as OWL DL are well-equipped for representing abstract concepts, but experience limitations when modeling concrete features as price, weight, age, etc. For this purpose, in [2], an approach is proposed for including concrete domains in the description logic $\mathcal{ALC}$. A concrete domain $\mathcal{D}$ is defined as as a set $\Delta_{\mathcal{D}}$, the domain of $\mathcal{D}$, and a set $pred(\mathcal{D})$, the predicate names of $\mathcal{D}$. Each predicate name is associated with a predicate of arity n, $P^{\mathcal{D}} \subseteq \Delta^n_{\mathcal{D}}$.

In order to maintain the decidability of the extended language, the concrete domains need to be admissible. A concrete domain is considered admissible if it satisfies three conditions: (i) the set of predicate names is closed under negation, (ii) the set of predicate names contains a name for $\Delta_{\mathcal{D}}$, and (iii) the satisfiability of conjunctions of the form:

$$\bigwedge_{i=1}^{k} P_i(\underline{x}^{(i)})$$

where $P_1, ..., P_k$ are predicate names in $pred(\mathcal{D})$ of arity $n_1, ..., n_k$, respectively, and $\underline{x}^{(i)}$ represents an n_i-tuple $(x_1^{(i)}, ..., x_{n_i}^{(i)})$ of variables, is decidable.

From a tOWL perspective we identify one concrete domain that has been proven in the literature [12] to be admissible, the set of rational numbers with the comparison operators $<$, $\leq$, $=$, $\neq$, $\geq$, and $>$. The set of time intervals with the 13 Allen operators *equal*, *before*, *after*, *meets*, *met-by*, *overlaps*, *overlapped-by*, *during*, *contains*, *starts*, *started-by*, *finishes*, and *finished-by* can be translated to operations on the previously identified concrete domain. The intervals concrete domain is not admissible based on the previous definition, but it is a well-behaved concrete domain that has been proven to maintain the decidability of the extended language [12].

In [2] the authors propose $\mathcal{ALC}(\mathcal{D})$, $\mathcal{ALC}$ extended with an admissible concrete domain $\mathcal{D}$. The extension is based on the following concrete domain constructor:

$$\exists u_1 ... u_n.P$$

where u_i are concrete feature chains, i.e., compositions of the form $f_1 ... f_m g$ where $f_1, ... f_m$ are abstract features, and g is a concrete feature. The semantics of the concrete domain constructor is defined as follows:

$$(\exists u_1 ... u_n.P)^{\mathcal{I}} = \{a \in \Delta_{\mathcal{I}} | \ \exists x_1, ..., x_n \in \Delta_{\mathcal{D}} : \\ u_i^{\mathcal{I}}(a) = f_1^{\mathcal{I}} ... f_{m_i}^{\mathcal{I}} g^{\mathcal{I}}(a) = x_i, 1 \leq i \leq n, (x_1, ..., x_n) \in P^{\mathcal{D}}\}$$

where $\Delta_{\mathcal{I}}$ is the abstract domain of the interpretation.

In the same paper, the authors prove that $\mathcal{ALC}(\mathcal{D})$ is decidable. In addition, it is proven that the union of two admissible domains yields an admissible domain, i.e., admissibility is closed under union. The previous concrete domain constructor has been generalized to roles in [7] as follows:

$$\exists u_1 ... u_n.P$$
$$\forall u_1 ... u_n.P$$

where u_i are concrete role chains, i.e., compositions of the form $r_1 ... r_m g$ where $r_1, ... r_m$ are abstract roles, and g is a concrete feature.

The semantics of the generalized concrete domain constructor is defined as:

$$(\exists u_1 ... u_n.P)^{\mathcal{I}} = \{a \in \Delta_{\mathcal{I}} | \ \exists x_1, ..., x_n \in \Delta_{\mathcal{D}} : \\ u_i^{\mathcal{I}}(a) = r_1^{\mathcal{I}} ... r_{m_i}^{\mathcal{I}} g^{\mathcal{I}}(a) = x_i, 1 \leq i \leq n, (x_1, ..., x_n) \in P^{\mathcal{D}}\}$$
$$(\forall u_1 ... u_n.P)^{\mathcal{I}} = \{a \in \Delta_{\mathcal{I}} | \ \forall x_1, ..., x_n \in \Delta_{\mathcal{D}} : \\ u_i^{\mathcal{I}}(a) = r_1^{\mathcal{I}} ... r_{m_i}^{\mathcal{I}} g^{\mathcal{I}}(a) = x_i, 1 \leq i \leq n, (x_1, ..., x_n) \in P^{\mathcal{D}}\}$$

The earlier decidability results have been generalized in [13] to the more powerful description logic $\mathcal{SHIQ}(\mathcal{D})$. As tOWL aims at extending OWL with temporal information based on the previously identified concrete domains, the decidable subset of tOWL is given by $\mathcal{SHIN}(\mathcal{D})$, i.e., OWL DL with two well-behaved concrete domains but without nominals.

11.2.2 4D Fluents

Most of the existing knowledge representation formalisms on the Semantic Web deal with the representation of static domains, failing to capture the semantics of dynamic domains. A notable exception is the 4D Fluents approach from [17], which is based on an OWL DL ontology for modeling a perdurantist view on the world. The abstract syntax of the proposed ontology is defined as follows:

```
Ontology(4dFluents
  Class(TimeSlice)
  DisjointClasses(TimeSlice TimeInterval)
  Property(fluentProperty
    domain(TimeSlice)
    range(TimeSlice))
  Property(tsTimeSliceOf Functional
    domain(TimeSlice)
    range(complementOf(TimeInterval))
  Property(tsTimeInterval Functional
    domain(TimeSlice)
    range(TimeInterval)))
```

The cornerstone of the ontology is given by fluents, i.e., object properties that change through time. The authors investigate several solutions for the fluents representation in OWL. The first solution is provided by adding an extra temporal dimension to RDF triples that leads to ternary predicates which are not supported by the OWL language unless reification is used. Unfortunately, reification has no clear semantics so this solution is discarded. A second solution is provided by adding a meta-logical predicate *holds* which makes triples valid at certain moments in time. This solution is also rejected as OWL DL does not support second order logic.

The authors propose to represent fluents as properties that have as domain and range timeslices. Timeslices stand for entities that are stretched through the temporal dimension (temporal worms). For timeslice representation two properties are used: *tsTimeSliceOf* to refer to the corresponding entity, and *tsTimeInterval* to point to the associated interval. For representing intervals, the 4D Fluents ontology imports the OWL-Time ontology [9]. The definition of *TimeInterval* in OWL-Time is given as follows:

```
Ontology(OWL-Time
  Class(TimeInterval)
  Class(InstantThing)
  Property(begins Functional
    domain(TimeInterval)
    range(InstantThing)
  Property(ends Functional
    domain(TimeInterval)
    range(InstantThing))
  Property(inCalendarClockDataType
    domain(InstantThing)
    range(xsd:dateTime))
  ...)
```

TimeInterval is defined as a class with two properties *begins* and *ends* that refer to *InstantThing*s. *InstantThing* has the property *inCalendarClockDataType* to refer to one time instant given using XML Schema *xsd*:*dateTime*. When using fluents to link two different timeslices (possibly belonging to two different entities), one needs to make sure that these two timeslices correspond to the same time interval. This constraint goes outside the OWL DL expressivity as it accesses information from two possibly different entities and needs to be enforced before further processing the ontology.

11.3 tOWL

This section presents the temporal OWL (tOWL) language [15], focusing on its abstract syntax and semantics. First, Subsect. 11.3.1 gives an overview of the tOWL language and its layer-based architecture. Then, Subsect. 11.3.2 introduces the tOWL OWL Schema in RDF/XML.

11.3.1 tOWL Overview

As given in Fig. 11.1 the tOWL language is composed of 4 layers, the bottom layer being the OWL DL layer.

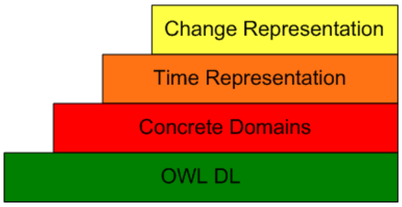

Fig. 11.1 tOWL layer cake

The first layer introduced by tOWL concerns the expressiveness of the language in a general, rather than in a strictly temporal sense. The *Concrete Domains* layer enables the representation of restrictions on property chains based on concrete domain predicates.

Partly enabled by the *Concrete Domains* layer, the *Temporal Representation* layer adds a temporal reference system to the language, in the form of concrete time and concrete temporal relations. For this purpose we use two concrete domains: the set of time instants given by XML Schema $xsd{:}dateTime$ (which is equivalent to the set of rational numbers) with the comparison predicates, and the set of time intervals with the Allen predicates. The $Interval$ is defined using the time instant concrete domain as follows:

$$Interval \equiv \exists\, start\ end. <$$

Upon enabling temporal reference in the language, the representation of change and state transitions is provided through the *Change Representation* layer. This extension enables the modeling of temporal parts of individuals that may have different property values at various moments in time. For this purpose we employ the fluents approach of the 4D Fluents ontology.

11.3.2 OWL Schema of tOWL

In this section we present an OWL schema of the tOWL language constructs. This is done in the same fashion as for OWL in the form of an "RDF Schema of OWL" [3]. The main purpose of this section is to provide a clear overview of the tOWL vocabulary.

The presentation of the OWL schema of tOWL starts off with the usual preliminaries in the form of namespace declarations as for any OWL ontology:

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
     <!ENTITY rdf  "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
     <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
     <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#">
     <!ENTITY owl  "http://www.w3.org/2002/07/owl#">
     <!ENTITY towl "http://www.towl.org/towl#">
     <!ENTITY towl_ "http://www.towl.org/towl">
   ]>
<rdf:RDF xmlns:rdf  = "&rdf;"
         xmlns:rdfs = "&rdfs;"
         xmlns:xsd  = "&xsd;"
         xmlns:owl  = "&owl;"
         xmlns      = "&towl;"
         xml:base   = "&towl_;">
```

The class *TimeSlice* is the superclass of all timeslices. This concept is introduced in the language by the *Change Representation* layer as follows:

```
<owl:Class rdf:ID="TimeSlice">
  <rdfs:label>TimeSlice</rdfs:label>
  <rdfs:subClassOf rdf:resource="&owl;Class"/>
</owl:Class>
```

Individuals of type *TimeSlice*, as presented in the previous paragraph, describe a regular individual over some period of time. Indicating which individual is described by an instance of type *TimeSlice* is achieved through the *timeSliceOf* functional property:

```
<owl:FunctionalProperty rdf:ID="timeSliceOf">
  <rdfs:label>timeSliceOf</rdfs:label>
  <rdf:type rdf:resource="&owl;ObjectProperty"/>
  <rdfs:domain rdf:resource="#TimeSlice"/>
  <rdfs:range>
    <owl:Class>
      <owl:complementOf>
        <owl:Class>
          <owl:unionOf rdf:parseType="Collection">
            <owl:Class rdf:about="#TimeSlice"/>
            <owl:Class rdf:about="#Interval"/>
            <owl:Class rdf:about="&rdfs;Literal"/>
          <owl:unionOf/>
        </owl:Class>
      </owl:complementOf>
    </owl:Class>
  </rdfs:range>
</owl:FunctionalProperty>
```

Another property describing individuals of type *TimeSlice* indicates the period of time for which these individuals hold. This is specified through the *time* functional property, that points to individuals of type *Interval*. This property, as well as the *Interval* class, are specified as follows:

```
<owl:FunctionalProperty rdf:ID="time">
  <rdfs:label>time</rdfs:label>
  <rdf:type rdf:resource="&owl;ObjectProperty"/>
  <rdfs:domain rdf:resource="#TimeSlice"/>
  <rdfs:range rdf:resource="#Interval"/>
</owl:FunctionalProperty>

<owl:Class rdf:ID="Interval">
  <rdfs:label>Interval</rdfs:label>
  <rdfs:subClassOf rdf:resource="&owl;Class"/>
</owl:Class>
```

Intervals are characterized by a starting point and an ending point, respectively. These bounds of an interval are represented as XML Schema *xsd*:*dateTime* datatypes, and connected to the respective interval through the *start* and *end* properties, respectively:

```
<owl:FunctionalProperty rdf:ID="start">
  <rdf:type rdf:resource="&owl;DatatypeProperty"/>
  <rdfs:domain rdf:resource="#Interval"/>
  <rdfs:range rdf:resource="&xsd;dateTime"/>
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:ID="end">
  <rdf:type rdf:resource="&owl;DatatypeProperty"/>
  <rdfs:domain rdf:resource="#Interval"/>
  <rdfs:range rdf:resource="&xsd;dateTime"/>
</owl:FunctionalProperty>
```

Indicating change for timeslices is achieved in the tOWL language through the use of fluents. A *FluentProperty* may come in one of two flavors, namely *FluentObjectProperty* or *FluentDatatypeProperty*. The first type links a timeslice to another timeslice, while the second type is used to indicate changing concrete values and thus links a timeslice to an XML Schema datatype. The fluent property types are defined as follows:

```
<owl:Class rdf:ID="FluentProperty">
  <rdfs:label>FluentProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdf;Property"/>
</owl:Class>

<owl:Class rdf:ID="FluentObjectProperty">
  <rdfs:label>FluentObjectProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="#FluentProperty"/>
</owl:Class>

<owl:Class rdf:ID="FluentDatatypeProperty">
  <rdfs:label>FluentDatatypeProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="#FluentProperty"/>
</owl:Class>
```

The 13 Allen relations [1] that describe any possible relation that may exist between 2 intervals are introduced in the language through the *TimeIntervalPredicate* class, limited to 13 individuals corresponding to the Allen predicates in a one-to-one fashion:

```
<owl:Class rdf:ID="TimeIntervalPredicate">
  <rdfs:label>TimeIntervalPredicate</rdfs:label>
  <owl:equivalentClass>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:ID="equal"/>
        <owl:Thing rdf:ID="met-by"/>
        <owl:Thing rdf:ID="meets"/>
        ...
      </owl:oneOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

In the same fashion we describe $<$, $\leq$, $=$, $\neq$, $\geq$, and $>$ relations that may hold between concrete $xsd{:}dateTime$ values that represent the end points of intervals:

```
<owl:Class rdf:ID="DateTimePredicate">
  <rdfs:label>DateTimePredicate</rdfs:label>
  <owl:equivalentClass>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:ID="dateTime-less-than"/>
        <owl:Thing rdf:ID="dateTime-greater-than"/>
        <owl:Thing rdf:ID="dateTime-equal"/>
        ...
      </owl:oneOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

Concrete features, i.e., functional properties over the concrete domain, are introduced in tOWL through the *ConcreteFeature* class:

```
<owl:Class rdf:ID="ConcreteFeature">
  <rdfs:label>ConcreteFeature</rdfs:label>
  <rdfs:subClassOf rdf:resource="&owl;FunctionalProperty"/>
  <rdfs:subClassOf rdf:resource="&owl;DatatypeProperty"/>
</owl:Class>
```

One of the novelties of the tOWL language consists of the introduction of chains of roles ending in a concrete feature. This is present in the language through the *ConcreteRoleChain* construct:

```
<owl:Class rdf:ID="ConcreteRoleChain">
  <rdfs:label>ConcreteRoleChain</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdf;List"/>
</owl:Class>
```

Introducing such chains has impact on the type of restrictions allowed in the language. For this purpose, the OWL construct *owl:onProperty* is extended with the *onPropertyChains* construct, thus allowing restrictions on tOWL chains:

```
<owl:ObjectProperty rdf:ID="onPropertyChains">
  <rdfs:label>onPropertyChains</rdfs:label>
  <rdfs:domain rdf:resource="&owl;Restriction"/>
  <rdfs:range rdf:resource="&tow;ConcreteRoleChains">
</owl:ObjectProperty>

<owl:Class rdf:ID="ConcreteRoleChains">
  <rdfs:label>ConcreteRoleChains</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdf;List"/>
</owl:Class>
```

In the same fashion, the OWL restrictions *allValuesFrom* and *someValuesFrom* are extended to include the *TimeIntervalPredicate* and *DateTimePredicate* classes:

```
<owl:ObjectProperty rdf:ID="dataAllValuesFrom">
  <rdfs:label>allValuesFrom</rdfs:label>
  <rdfs:domain rdf:resource="&owl;Restriction"/>
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#TimeIntervalPredicate"/>
        <owl:Class rdf:about="#DateTimePredicate"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="dataSomeValuesFrom">
  <rdfs:label>someValuesFrom</rdfs:label>
  <rdfs:domain rdf:resource="&owl;Restriction"/>
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#TimeIntervalPredicate"/>
        <owl:Class rdf:about="#DateTimePredicate"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
</owl:ObjectProperty>
```

11.4 A tOWL Ontology for the Leveraged Buyouts

In order to show the usefulness of the proposed language, in this section, we illustrate how one can model a business process for which the temporal aspects are essential. For this purpose, we decided to focus on Leveraged Buyouts (LBO), one of the most complex processes encountered in business acquisitions. First, in Subect. 11.4.1 we introduce the LBO example. Then, in Subsect. 11.4.2 we present the TBox of the LBO example. After that, in Subsect. 11.4.3 we give the ABox of the LBO example. Last, in Subsect. 11.4.4 we show some use cases for the tOWL in the context of the LBO example.

The focus of this section is to illustrate how the information regarding an LBO process can be represented in the tOWL language. For this purpose, we provide a representation of this example in tOWL abstract syntax and RDF/XML syntax. This is done for both TBox and ABox level representations.

11.4.1 Leveraged Buyouts

A Leveraged Buyout is a special type of an acquisition in which a company buys another company by using loans guaranteed with assets of the bought company. Figure 11.2 shows the activity diagram of an LBO process associated to one of the buyer/buyee companies. After each stage, but the last one, the system can go in the *Abort* stage, which ends the process without acquisition. The first state is the *Early Stage*. From this stage a transition can be made to the *Due Diligence* stage or the current state might be extended. After *Due Diligence* stage follows the *Bidding* stage, from which the process can optionally go to *Raise Bid* stage. Last, the process ends with *Acquisition*.

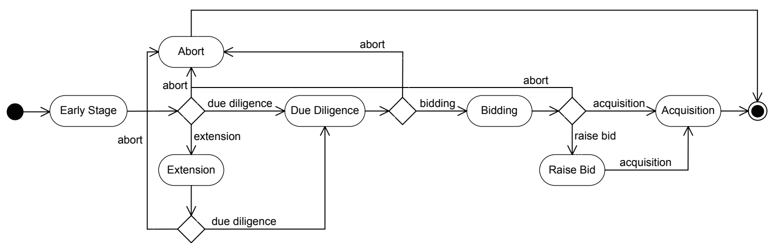

Fig. 11.2 The Leverged Buyouts process

The running example in this paper is based on the largest LBO acquisition in Europe. In 2007, two hedge funds did compete for the acquisition of a target company. From the two hedge funds, Kohlberg Kravis Roberts & Co and Terra Firma, the first won the bidding and acquired the target company Alliance Boots.

11.4.2 TBox

At TBox level we represent conceptual information that is known about LBO processes in general. In this context, two types of companies that take part in an LBO are known: *HedgeFund* and *Target*, which we define as subclasses of the *Company* class. In tOWL abstract syntax, this translates to the following:

```
Class(Company)
Class(HedgeFund partial Company)
Class(Target partial Company)
```

In tOWL RDF/XML, this can be represented as:

```
<owl:Class rdf:ID="Company"/>
<owl:Class rdf:ID="HedgeFund">
  <rdfs:subClassOf rdf:resource="#Company"/>
</owl:Class>
<owl:Class rdf:ID="Target">
  <rdfs:subClassOf rdf:resource="#Company"/>
</owl:Class>
```

The different stages of an LBO process are represented as subclasses of the *Stage* class, such as for example in the case of the *Bidding* stage. In tOWL abstract syntax, this is represented as:

```
Class(Bidding partial Stage)
```

The tOWL RDF/XML representation of the above expression takes the form:

```
<owl:Class rdf:ID="Stage"/>
<owl:Class rdf:ID="Bidding">
  <rdfs:subClassOf rdf:resource="#Stage"/>
</owl:Class>
```

All stages are pairwise disjoint, which can be represented in tOWL abstract syntax as follows:

```
DisjointClasses(EarlyStage, DueDiligence, ..., Extension)
```

In tOWL RDF/XML, for each unique pair of stages their disjuction is expressed like:

```
<owl:Class rdf:ID="RaiseBid">
  <rdfs:subClassOf rdf:resource="#Stage"/>
  <owl:disjointWith rdf:resource="#Acquisition"/>
</owl:Class>
```

We define the class of all timeslices of an LBO process as follows, in tOWL abstract syntax:

```
Class(LBOProcess_TS complete
  restriction(timeSliceOf(someValuesFrom LBOProcess)))
```

The same representation takes the following form in tOWL RDF/XML:

```
<owl:Class rdf:ID="LBOProcess_TS">
  <owl:quivalentClass>
    <owl:Restriction>
       <owl:onProperty rdf:resource="&towl;timeSliceOf"/>
       <owl:someValuesFrom rdf:resource="#LBOProcess"/>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

In similar fashion, we define, for each stage, the class of all timeslices of that stage. For the *EarlyStage* this achieved as follows, in tOWL abstract syntax:

```
Class (EarlyStage_TS complete
   restriction(timeSliceOf(someValuesFrom EarlyStage)))
```

The tOWL RDF/XML serialization of this fact takes the form:

```
<owl:Class rdf:ID="EarlyStage_TS">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&towl;timeSliceOf"/>
      <owl:someValuesFrom rdf:resource="#EarlyStage"/>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

For each stage, we define a functional property that links a particular LBO process timeslice to the timeslice of the stage belonging to it. Please note that this property is not a fluent as it links timeslices corresponding to different temporal intervals and it does not change in time. In tOWL abstract syntax, this is represented as:

```
ObjectProperty(earlyStage
  domain(LBOProcess_TS)
  range(EarlyStage_TS)
  Functional)
```

Representing the same fact in tOWL RDF/XML resumes to the following expression:

```
<towl:ObjectProperty rdf:ID="earlyStage">
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  <rdfs:domain rdf:resource="#LBOProcess_TS"/>
  <rdfs:range rdf:resource="#EarlyStage_TS"/>
</towl:ObjectProperty>
```

Next, we move on to define the *inStage* fluent, that for each timeslice of a company points to the stage in which the company finds itself. In tOWL abstract syntax it is represented as:

```
FluentObjectProperty(inStage
  domain(
    restriction(timeSliceOf(someValuesFrom Company)))
  range(
    restriction(timeSliceOf(someValuesFrom Stage)))
```

In tOWL RDF/XML serialization the previously fluent is described as:

```
<towl:FluentObjectProperty rdf:ID="inStage">
  <rdfs:domain>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&towl;timeSliceOf"/>
        <owl:someValuesFrom rdf:resource="#Company"/>
      </owl:Restriction>
  </rdfs:domain>
  <rdfs:range>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&towl;timeSliceOf"/>
      <owl:someValuesFrom rdf:resource="#Stage"/>
    </owl:Restriction>
  </rdfs:range>
</towl:FluentObjectProperty>
```

Timeslices of an LBO process are defined by the sequence of stages that a company may follow in this process. Representing such sequences relies on concrete role chains, and reduces to assessing the order of the intervals associated with the different stages.

For example, representing that the *EarlyStage* always starts an LBO process can be represented in tOWL abstract syntax as follows:

```
Class(LBOProcess_TS complete
  intersectionOf(
    restriction(
      dataSomeValuesFrom((earlyStage time) time starts))
    ...))
```

For the RDF/XML serialization of the above type of restriction we need two types of lists: lists for representing each concrete role chain, and a list that stores the property chains on which the binary concrete domain predicate is applied. Please note that for concrete features that stand for concrete feature chains used in restrictions, the list construct is not needed.

Serializing the previous axiom in tOWL RDF/XML results in the following:

```
<towl:ConcreteRoleChain rdf:ID="iEarlyStageChain">
  <rdf:first rdf:resource="#earlyStage"/>
  <rdf:rest>
    <towl:ConcreteRoleChain>
      <rdf:first rdf:resource="#time"/>
      <rdf:rest rdf:resource="&rdf;nil"/>
    </towl:ConcreteRoleChain>
  </rdf:rest>
</towl:ConcreteRoleChain>

<owl:Class rdf:ID="LBOProcess_TS">
  <owl:equivalentClass>
    <owl:intersectionOf parseType="Collection">
      <owl:Restriction>
        <towl:onPropertyChains>
          <towl:ConcreteRoleChains>
            <rdf:first rdf:resource="#iEarlyStageChain"/>
            <rdf:rest>
              <towl:ConcreteRoleChains>
                <rdf:first rdf:resource="#time"/>
                <rdf:rest rdf:resource="&rdf;nil"/>
              </towl:ConcreteRoleChains>
            </rdf:rest>
          </towl:ConcreteRoleChains>
        </towl:onPropertyChains>
        <towl:dataSomeValuesFrom rdf:resource="#starts"/>
      </owl:Restriction>
      ...
    </owl:intersectionOf>
  </owl:equivalentClass>
</owl:Class>
```

Similarly, *meets* is used between the other stages of the LBO process, while the last stage, i.e., *Acquisition finishes* the LBO process.

11.4.3 ABox

At ABox level we represent particular information that is known about the specific LBO process presented in this section. We start off by instantiating the relevant individuals that are known to play a role in the LBO process.

First, we represent the participating companies. In tOWL abstract syntax this is represented as follows:

```
Individual(iAllianceBoots Target)
Individual(iKKR HedgeFund)
Individual(iTerraFirma HedgeFund)
```

Similarly, we represent the same in tOWL RDF/XML:

```
<Target rdf:ID="iAllianceBoots"/>
<HedgeFund rdf:ID="iTerraFirma"/>
<HedgeFund rdf:ID="iKKR"/>
```

For each of the hedgefunds involved, we instantiate a process and define its stages, such as in the case of the TerraFirma. In tOWL abstract syntax, we define the following:

```
Individual(iLBOProcess1 type(LBOProcess)
  value(earlyStage iEarlyStage1)
  value(dueDiligence iDueDiligence1)
  value(bidding iBidding1)
  value(abort iAbort1))

Individual(iLBOProcess_TS1 type(LBOProcess_TS)
  value(timeSliceOf iLBOProcess1))}
```

The tOWL RDF/XML representation of the above two individuals takes the following form:

```
<LBOProcess rdf:ID="iLBOProcess1">
  <earlyStage rdf:ID="iEarlyStage1"/>
  <dueDiligence rdf:ID="iDueDiligence1"/>
  <bidding rdf:ID="iBidding1"/>
  <abort rdf:ID="iAbort1"/>
</LBOProcess>

<LBOProcess_TS rdf:ID="iLBOProcess_TS1">
  <towl:timesliceOf rdf:ID="iLBOProcess1"/>
</LBOProcess_TS>
```

Next, we represent the information contained by the individual news messages associated with the LBO process. We illustrate this by employing the first news message that describes the hedgefund *TerraFirma* entering the *EarlyStage* phase. This is described in the following news message:

Buyout firm Terra Firma mulls Boots bid
Sun Mar 25, 2007 8:42am EDT
This news message signals the beginning of the LBO, mentioning that Terra Firma is considering a bid for Alliance Boots (*EarlyStage*).

For representing the information contained in the news message we create a timeslice for the hedgefund and the target, respectively, a time interval associated to the stage, and employ the *inStage* fluent to associate the companies to the stage. In tOWL abstract syntax, this resumes to the following:

```
Individual(t1 type(Interval))
Individual(iEarlyStage1 type(EarlyStage))
Individual(iEarlyStage1_TS1 type(TimeSlice)
  value(timeSliceOf iEarlyStage1)
  value(time t1))
Individual(iAllianceBoots_TS1 type(TimeSlice)
  value(timeSliceOf iAllianceBoots)
  value(time t1)
  value(inStage iEarlyStage1_TS1))
Individual(iTerraFirma_TS1 type(TimeSlice)
  value(timeSliceOf iTerraFirma)
  value(time t1)
  value(inStage iEarlyStage1_TS1))
```

The tOWL RDF/XML representation of this news message is formulated as follows:

```
<towl:Interval rdf:ID="t1"/>
<EarlyStage rdf:ID="iEarlyStage1"/>
<towl:TimeSlice rdf:ID="iEarlyStage1_TS1">
  <towl:timeSliceOf rdf:ID="iEarlyStage1"/>
  <towl:time rdf:ID="t1"/>
</towl:TimeSlice>
<towl:TimeSlice rdf:ID="iAllianceBoots_TS1">
  <towl:timeSliceOf rdf:ID="iAllianceBoots"/>
  <towl:time rdf:ID="t1"/>
  <inStage rdf:ID="iEarlyStage1_TS1"/>
</towl:TimeSlice>
<towl:TimeSlice rdf:ID="iTerraFirma_TS1">
  <towl:timeSliceOf rdf:ID="iTerraFirma"/>
  <towl:time rdf:ID="t1"/>
  <inStage rdf:ID="iEarlyStage1_TS1"/>
</towl:TimeSlice>
```

Finally, it should be remarked that although the representation proposed here is not an exhaustive one with regard to an arbitrary LBO process, i.e., it does not cover the whole process, it is sufficient for illustrating the fulfillment of the two objectives it set out to achieve: i) illustrating the power and use of the tOWL language constructs in a temporal context, and ii) how an LBO process can be modeled by employing the tOWL language.

11.4.4 Use Cases

The usefulness of the tOWL representation of the LBO process is explained by means of three use cases depicted in Fig. 11.3: (a) historical analysis, (b) stock prediction, and (c) regulatory conformance.

In the historical analysis use case it can be determined in which stage the LBO process is, given a certain time instant. In this way the LBO process evolution can be analyzed at every moment in time. In the example from Fig. 11.3, the LBO process is in *Early* stage at time t_1 and in *Due Diligence* at time t_2.

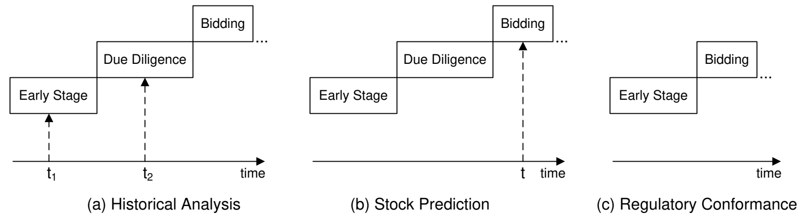

Fig. 11.3 Three use cases for the tOWL representation of the LBO process

In the stock prediction use case it can be estimated what is the impact on the stock price of a certain company given its stage in the LBO process. For instance knowing that the LBO process is in an advanced stage has a positive effect on the price of the target company stocks. In the example from Fig. 11.3, the LBO process is in *Bidding* stage at time t, which means that the *Alliance Boots*' stock price will possibly increase.

In the regulatory conformance use case it can be checked if a given LBO process (ABox) obeys its temporal obligations from the regulatory specification (TBox). In the example from Fig. 11.3, the *Bidding* stage comes immediately after the *Early Stage* which conflicts with the LBO process regulation that says that after *Early Stage* should follow the *Due Diligence* stage.

11.5 Related Work

In this section we compare our approach with related work for representing time and change on the Semantic Web. In Subsect. 11.5.1 we analyse Temporal RDF, an RDF extension to represent both time and change. Then, in Subsect. 11.5.2 we discuss OWL-Time, an OWL ontology able to represent time. Last, in Subsect. 11.5.3 we relate to the 4D Fluents ontology able to represent time and change.

11.5.1 Temporal RDF

Temporal RDF [6] extends RDF with temporal information. The approach is based on temporal graphs which are sets of triples with time intervals or instants (temporal labels) associated to them. The time intervals represent the time in which a triple holds true, and have $intial$ and $final$ properties defined. The authors focus on valid time, while stating that transaction time can be defined in a similar way. The proposed approach extends the RDF semantics with a temporal semantics and defines temporal entailment of RDF graphs.

An interesting feature of temporal RDF is that it supports anonymous time which represents a time variable instead of a constant. One such temporal variable introduced in the language is *NOW* which stands for the current time. This variable is a place holder for the time at which the corresponding triple is evaluated. Temporal RDF has a temporal query language, and it is proven that the temporal labeling of triples does not introduce any complexity overhead in query answering.

Despite using reification for associating time intervals to triples, the lack of semantics of reification is overcame by defining temporal rules that provide for equivalent representations. Nevertheless, from a practical perspective, the authors do not show how reification can be avoided from the serialization of RDF graphs. In addition, as XML is the lingua franca on the Semantic Web, temporal RDF doesn't have an RDF/XML serialization which makes this approach difficult to use in practice.

Temporal RDF and tOWL are able to specify both temporal instants and intervals. In tOWL the reification problem is avoided by employing a perdurantist view on the world, extending objects in a temporal dimension. In addition, we target OWL instead of RDF, which allows a more precise definition (e.g., functional properies, restrictions, etc.) of the provided vocabulary. We also make use of concrete domains which allows a more accurate definition of intervals (the start time point of an interval has to be before the end time point of the same interval) as well as employing the Allen calculus for defining temporal relationships between object states. Differently than temporal RDF, tOWL makes use of existing standards, when possible, for representing temporal information (e.g., $xsd{:}dateTime$ for representing time points).

11.5.2 OWL-Time

One of the first OWL ontologies seeking to represent time is OWL-Time [9]. The initial purpose of OWL-Time was to describe the temporal content of Web pages and services. It later grew to a reference time ontology able to represent time, duration, clock, calendar, and temporal aggregates in may domains. Some of the applications of OWL-Time are information retrieval and question answering. OWL-Time is currently a W3C working draft [10].

The root node of OWL-Time ontology is the $TemporalEntity$ which is refined in two types: $Instant$ and and $Interval$. Any $TemporalEntity$ has a *begins* and *ends* property which refer to $InstantThing$s. In addition, the ontology defines $CalendarClockDescription$, $DurationDescription$, and $TemporalUnit$. OWL-Time provides two alternative ways to represent time points $CalendarClockDescription$ and $xsd{:}dateTime$. The advantage of the first representation is that it allows one to express more information (e.g., the first day of a week, i.e., Sunday), while the second one is based on a standard which has a wider acceptance and usage.

OWL-Time defines the *inside* relation between instants and intervals, and the Allen temporal relations between intervals. The Allen relations can be specified based the transitive $before$ relation between begin and end points. Unfortunately, this translation scheme goes beyond the OWL expressivity and thus needs to be encoded separately from the OWL inference rules. From the 13 Allen relations six have inverses (*equal* doesn't have an inverse) which can be easily expressed based on OWL semantics.

For time representations that are not based on *xsd*:*dateTime*, OWL-Time defines a time zone ontology that allows to define time zones (e.g., +1 hour from Greenwich Mean Time) and link them to geographical locations (e.g., the Netherlands). Additionally OWL-Time allows the representation of temporal aggregates [16]. For this purpose, it builds upon the the inherent ordering of temporal entities. For example in OWL-Time one is able to specify "every other Friday in 2009" which takes in consideration the days ordering in a specific context, i.e., "2009".

The semantics of the introduced concepts is given in first order logic and to some extent also in second order logic (quantifying over predicates for the definition of temporal aggregates) and thus goes beyond the expressivity power of OWL DL. It is not clear which set of the proposed vocabulary provides for a decidable language. Also, some of the introduced primitives as *xsd*:*duration* have ambiguous semantics (e.g., a duration of 1 month can represent 28 days, 29 days, 30 days, and 31 days) and for this reason they have been left out from the XML Schema datatypes included in RDF and OWL [8].

For tOWL we have identified that the decidable language subset is $\mathcal{SHIN}(\mathcal{D})$. Also, tOWL is able to better capture the semantics of intervals (e.g., impose an ordering between the start and end time points of an interval), and make use of Allen calculus for defining temporal relations at TBox level by employing the functionality offered by concrete domains (OWL-Time is able to do this only at ABox level). tOWL does not make use of durations avoiding thus their ambiguous semantics. Differently than OWL-Time, tOWL is able to represent the objects' dynamics by specifying object properties that change in time.

11.5.3 4D Fluents

As identified in Sect. 11.2, the 4D Fluents ontololgy [17] allows to represent both time and change. Unfortunately, as it makes use of OWL-Time ontology for the time representation, it suffers from the shortcomings previously identified for this approach. While 4D Fluents defines timeslices and fluents as an ontology, tOWL brings these primitives as first class citizens of the new language enabling anyone devising a tOWL ontology to make use of these constructs in modeling the domain dynamics. In addition, this approach allows a tOWL reasoner to translate the Allen relations between intervals as temporal relations between their end points, and enforce that two timelisces connected by the same fluent correspond to the same interval.

The 4D Fluents solution for representing a dynamic world suffers from the proliferation of objects. One fluent requires two timeslices, each timeslice referring to one entity and one temporal interval, in total 7 triples (including the triple in which the fluent is employed) need to be created in the dynamic domain for 1 triple in the static domain. In order to alleviate this problem, tOWL differentiates between two types of fluents FluentObjectProperty and FluentDatatypeProperty, FluenDataype-Property has as range datatypes which means that one timeslice and its two required relationships are not needed. This yields a reduction from 7 triples to 4 triples in the case of FluentDatatypeProperty.

Differently than 4D Fluents, tOWL allows the use of the Allen calculus at TBox level as for expressing temporal relations between the different concept states. Also, by extending the OWL language with concrete domains in addition to the temporal concrete domains, one can add other concrete domains as for example the Region Connection Calculus (RCC8) for specifying spatial relationships, or the price concrete domain for representing price changes.

11.6 Conclusion

The tOWL language is an extension of OWL DL that enables the representation of time and change in dynamic domains. It comes to meet shortcomings of previous approaches, such as [6, 9, 17] that only address these representations to a limited extent. For this purpose, tOWL makes use of concrete domains and a perdurantist view on the world based on fluents. The expressivity power of the language is demonstrated by means of one of the most complex use cases known from business process modeling, i.e., a real world Leveraged Buyout (LBO) process. The knowledge regarding this LBO is modeled at TBox level by means of axioms describing the possible paths through such a process. At ABox level we are able to describe the actual process through the representation of the information contained in real news messages associated with this particular LBO.

Currently we are working towards providing a tOWL Protege plugin that would foster the tOWL usage by allowing users to build tOWL ontologies using a simple user interface. For this purpose, we plan to keep the new interface compatible with the Protege OWL interface so that the transition to the new language is made as smooth as possible for existing Protege users. As far as the reasoner is concerned, we would like to implement the $\mathcal{SHIQ}(\mathcal{D})$ reasoning algorithm from [13], or use a hybrid reasoner in which one of the existing DL reasoners (e.g., Pellet) is extended with a concrete domain reasoning box.

In order to further reduce the proliferation of objects in tOWL, we would like to investigate the merging of timeslices by coalescing their corresponding intervals. Also, we would like to investigate the decidability of tOWL (in this paper it is shown that tOWL without nominals is decidable) by considering what is the influence of nominals on $\mathcal{SHIQ}(\mathcal{D})$ decidability results. We also plan also to examine how to extend the tOWL language with a spatial dimension by including the RCC8 calculus. As Allen calculus, RCC8 has the jointly exhaustive and pairwise disjoint property that ensures the decidability of the extended DL language [14].

Acknowledgements The authors are supported by the EU funded IST-STREP Project FP6-26896: *Time-determined ontology-based information system for realtime stock market analysis* (TOWL). More information is available on the official website[1] of the TOWL project.

References

1. Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM **26**(11), 832–843 (1983)
2. Baader, F., Hanschke, P.: A scheme for integrating concrete domains into concept languages. In: 12th International Joint Conference on Artificial Intelligence (IJCAI 1991), pp. 452–457. Morgan Kaufmann (1991)
3. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL Web Ontology Language reference. W3C Recommendation 10 February 2004 (2004). http://www.w3.org/TR/owl-ref/
4. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American **284**(5), 34–43 (2001)
5. Brickley, D., Guha, R.: RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation 10 February 2004 (2004). http://www.w3.org/TR/rdf-schema/
6. Gutierrez, C., Hurtado, C.A., Vaisman, A.A.: Introducing time into RDF. IEEE Transactions on Knowledge and Data Engineering **19**(2), 207–218 (2007)
7. Hanschke, P.: Specifying role interaction in concept languages. In: Third International Conference on Pirnciples of Knowledge Representation and Reasoning (KR 2002), pp. 318–329. Morgan Kaufmann (1992)
8. Hayes, P.: RDF semantics. W3C Recommendation 10 February 2004 (2004). http://www.w3.org/TR/rdf-mt
9. Hobbs, J.R., Pan, F.: An ontology of time for the Semantic Web. ACM Transactions on Asian Language Information Processing **3**(1), 66–85 (2004)
10. Hobbs, J.R., Pan, F.: Time ontology in OWL. W3C Working Draft 27 September 2006 (2006). http://www.w3.org/TR/2006/WD-owl-time-20060927/
11. Klyne, G., Carroll, J.J.: Resource Description Framework (RDF): Concepts and abstract syntax. W3C Recommendation 10 February 2004 (2004). http://www.w3.org/TR/rdf-concepts/
12. Lutz, C.: Interval-based temporal reasoning with general tboxes. In: Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001), pp. 89–96. Morgan Kaufmann (2001)
13. Lutz, C.: Adding numbers to the SHIQ description logic: First results. Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR 2002) pp. 191–202 (2002)
14. Lutz, C., Milicic, M.: A tableau algorithm for description logics with concrete domains and general TBoxes. Journal of Automated Reasoning **38**(1-3), 227–259 (2007)
15. Milea, V., Frasincar, F., Kaymak, U.: Knowledge engineering in a temporal Semantic Web context. In: Eigth International Conference on Web Engineering (ICWE 2008), pp. 65–74. IEEE Computer Society (2008)
16. Pan, F., Hobbs, J.R.: Temporal aggregates in OWL-Time. In: 18th International Florida Artificial Intelligence Research Society Conference (FLAIRS 2005), pp. 560–565. AAAI Press (2005)
17. Welty, C.A., Fikes, R.: A reusable ontology for fluents in OWL. In: Fourth International Conference on Formal Ontology in Information Systems (FOIS 2006), *Frontiers in Artificial Intelligence and Applications*, vol. 150, pp. 226–336. IOS Press (2006)

[1] http://www.semlab.nl/towl