A General Methodology for Equipping Ontologies With Time*

Hans-Ulrich Krieger

German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
krieger@dfki.de

Abstract

In the *first* part of this paper, we present a framework for enriching arbitrary upper or domain-specific ontologies with a concept of time. To do so, we need the notion of a time slice. Contrary to other approaches, we directly interpret the original entities as time slices in order to (i) avoid a duplication of the original ontology and (ii) to prevent a knowledge engineer from ontology rewriting. The diachronic representation of time is complemented by a sophisticated time ontology that supports underspecification and an arbitrarily fine granularity of time. As a showcase, we describe how the time ontology has been interfaced with the PROTON upper ontology. The *second* part investigates a temporal extension of RDF that replaces the usual triple notation by a more general tuple representation. In this setting, Hayes/ter Horst-like entailment rules are replaced by their temporal counterparts. Our motivation to move towards this direction is twofold: firstly, extending binary relation instances with time leads to a massive proliferation of useless objects (independently of the encoding); secondly, reasoning and querying with such extended relations is extremely complex, expensive, and error-prone.

1. Introduction

The first part of this paper presents a framework for enriching arbitrary upper or domain-specific ontologies with a concept of time. The work reported here is part of an EU-funded project called MUSING which is dedicated to the investigation of semantic-based business intelligence solutions.

Temporal information in MUSING is based on a diachronic representation of time, on top of which temporal reasoning services are defined (Krieger et al., 2008a). Since ontological knowledge in MUSING is encoded in OWL (McGuinness and van Harmelen, 2004), extending binary relations with an additional time argument is not that easy, due to the fact that OWL (or description logic in general) only provides unary and binary relations.

In order to equip ontologies with time, we need the notion of a time slice, as explained, e.g., in (Sider, 2001). Contrary to (Welty and Fikes, 2006), we directly interpret the original entities as time slices in order to (i) avoid a duplication of the original ontology and (ii) to prevent a knowledge engineer from ontology rewriting.

We will see that this reinterpretation makes it easy to extend an upper/domain ontology with time. The diachronic representation of time is complemented by a sophisticated time ontology that supports underspecification and an arbitrarily fine granularity of time.

MUSING makes use of a general upper-base ontology called PROTON (http://proton.semanticweb.org) that has been extended mostly by the MUSING partners from STI (formerly DERI), Innsbruck. As a showcase, we describe how the time ontology has been interfaced with PROTON. The OWL implementation of the methodology reported here (plus a general time ontology) can be obtained freely from the author.

*The research described here has been financed by the European Integrated projects **MUSING** (Multi-Industry Semantic-Based Business Intelligence, http://musing.eu/) and **CogX** (Cognitive Systems that Self-Understand and Self-Extend, http://cogx.eu) under contract numbers FP6-027097 and FP7 ICT-215181. I would like to thank the three referees for their valuable comments.

Even though our approach keeps the original ontology, it leads to a massive proliferation of "container" objects, due to the fact that the underlying data structure is still the RDF *triple* (Klyne and Carroll, 2004). Furthermore and very important, reasoning and querying with such a representation is extremely complex, expensive, and error-prone. It is worth noting that all other approaches, as presented in section 3., do suffer from the same disadvantage.

In order to overcome this problem, we propose to add some kind of temporal annotation to an RDF triple, realized as further temporal arguments (starting and ending time). We describe an extension of Hayes/ter Horst-like RDFS/OWL entailment rules that are "sensitive" to temporal information. We show that only lightweight reasoning capabilities are needed when working with such information.

The work reported in the second part of this paper is an outcome of the lessons learned from the MUSING project and is actively used and extended in the CogX project, whose aim is to develop a unified theory of self-understanding and self-extension with a convincing instantiation and implementation of this theory in a robot.

The representation of generalized tuples, reasoning with them and querying them is realized through *HFC*, a forward chainer developed at DFKI that scales up to millions of tuples, which is reasonable fast and expressive enough to formulate the extended entailment rules.

2. A Motivating Example

The problem with so-called *synchronic* relationships is that they all refer to only *one*, potentially hidden point/period in/of time. Here is an example:

Tony Blair was born on May 6, 1953.

Assuming a RDF-based representation, an information extraction system might compute the following set of triples:

```
tb rdf:type Person .
tb hasName "Tony Blair" .
tb dateOfBirth "1953-05-06" .
```

However, most relationships are *diachronic*, i.e., they embody the possibility to vary with time. Take, for instance,

the following example:

Christopher Gent was Vodafone's chairman until July 2003. Later, Chris became the chairman of GlaxoSmithKline with effect from 1st Jan 2005.

When applying the synchronic representation scheme from above, however, the resulting RDF graph mixes up the association between the fact and the temporal extend (two out of four possibilities are wrong):

```
cg isChairman vf .
cg isChairman gsk .
cg hasTime [????-??-??,2003-07-??]  .
cg hasTime [2005-01-01,????-??-??]  .
```

No longer is it clear whether `[????-??-??,2003-07-??]` belongs to `vf` or `gsk` (same holds for `[2005-01-01, ????-??-??]`).

3. Approaches to Diachronic Representation

Several well-known techniques of extending binary relations with additional arguments have been proposed in the literature. (Welty and Fikes, 2006) mention three of them and add a fourth one (4D or perdurantist view; see below), which we reinterpret w.r.t. an upper or domain ontology. This reinterpretation is the basis for representing temporal information in MUSING and one of the topics of this paper, since it opens a way to enrich arbitrary ontologies with the concept of time, without any ontology rewriting.

3.1. Equip Relations With a Temporal Argument

This approach has been pursued in temporal databases and the logic programming community. A binary relation, such as *hasCeo* between a company c and a person p becomes a ternary relation with a further temporal argument t (we limit ourself to one further argument encoding an interval, instead of two, representing the starting and ending time of an interval):

$$\mathit{hasCeo}(c,p) \longmapsto \mathit{hasCeo}(c,p,\underline{t})$$

Unfortunately, OWL and description logic in general only support unary (classes) and binary relations (properties) in order to guarantee decidability of the usual inference problems. Thus, forward chainers (such as OWLIM and Jena) as well as description logic reasoners (e.g., Racer or Pellet) are unable to handle such descriptions.

We note here that this approach is clearly the *silver bullet* of representation, since it is the easiest and most natural one, although a direct interpretation is incompatible with RDF and currently available reasoners. We will favor this kind of representation in the second part.

3.2. Apply a Meta-Logical Predicate

McCarthy & Hayes' situation calculus, James Allen's interval logic, and the knowledge representation formalism KIF use the meta-logical predicate *holds*. Hence, our *hasCeo* relation becomes

$$\mathit{hasCeo}(c,p) \longmapsto \underline{\mathit{holds}}(\mathit{hasCeo}(c,p),\underline{t})$$

McCarthy & Hayes call a statement whose truth value changes over time a *fluent* (McCarthy and Hayes, 1969). Thus the extended ternary relation from the previous subsection is a *relational* fluent. The *holds* expression here, however, embodies a *functional* fluent, meaning that $\mathit{hasCeo}(c,p)$ is assumed to yield a situation-dependent value. Such kinds of relations are not possible in OWL, since description logics limit themselves to subsets of function-free first order logic.

3.3. Reify the Original Relation

Reifying a relation instance leads to the introduction of a new object and four additional new relationships. In addition, a new class needs to be introduced for each reified relation, plus accessors to the original arguments. Furthermore and very important, relation reification loses the original relation, requiring a modification of the original ontology. Coming back to our *hasCeo* example, we get something like this (*HasCeo* is the newly introduced class):

$$\underline{\mathit{hasCeo}(c,p,t)} \longmapsto \exists e\,.\ \mathit{type}(e,\mathit{HasCeo}) \wedge \mathit{hasTime}(e,t) \wedge \mathit{company}(e,c) \wedge \mathit{person}(e,p)$$

3.4. Encode the 4D View in OWL

(Welty and Fikes, 2006) have presented an implementation of the 4D or perdurantist view in OWL, using so-called time slices (Sider, 2001), encoding the time dimension of spacetime.[1] Relations from the original ontology no longer connect the original entities, but instead connect time slices that belong to those entities. A time slice is merely a container for storing time. For a given ontology, such a representation requires a lot of rewriting:

$$\underline{\mathit{hasCeo}(c,p,t)} \longmapsto \exists \mathit{ts}_1,\mathit{ts}_2\,.\ \mathit{type}(\mathit{ts}_1,\mathit{TimeSlice}) \wedge \mathit{hasTimeSlice}(c,\mathit{ts}_1) \wedge \mathit{type}(\mathit{ts}_2,\mathit{TimeSlice}) \wedge \mathit{hasTimeSlice}(p,\mathit{ts}_2) \wedge \mathit{hasTime}(\mathit{ts}_1,t) \wedge \mathit{hasTime}(\mathit{ts}_2,t) \wedge \mathit{hasCeo}(\mathit{ts}_1,\mathit{ts}_2)$$

3.5. Reinterpret the 4D View

In MUSING, we have reinterpreted the perdurantist/4D view in that we have reinterpreted the original entries from the ontology. The basic idea can be summarized in the following slogan:

What has been an entity becomes a time slice.

In the example above, c and p are no longer entities, but instead time slices of an entity (a perdurant), that explain the behavior of an entity within a certain extension or point in time (e.g., that c is a time slice talking about a company or p a time slice, dealing with a person).

This reinterpretation does not need any ontology rewriting and makes it easy to equip arbitrary upper/domain ontologies with the concept of time. Coming back to our example, we have

$$\mathit{hasCeo}(c,p,\underline{t}) \longmapsto$$

[1] In the 4D view, all entities (the *perdurants*) only exist for some period of time. Given this view, it does not matter whether we are talking about an accidental, perhaps infinitely-small event (say, the shooting of a pistol) or a very long time interval (e.g., the lifetime of our universe). Entities under this view are often referred to as *spacetime worms* (Sider, 2001), since a four-dimensional trajectory identifies a perdurant in time and space.

$hasCeo(c, p) \wedge hasTime(c, t) \wedge hasTime(p, t) \wedge hasTimeSlice(C, c) \wedge hasTimeSlice(P, p)$

Note that the former binary predicate *hasCeo* is still available and unchanged. But the argument classes, viz., Company and Person have been equipped with an additional relation called *hasTime*, defined on class *TimeSlice*, as we will see later. Given this representation, everything that is defined on c, such as the CEOship, the name, the address, or the number of employees of this company, is assumed to co-occur during time period t. I.e., different facts speaking about the same time interval of the same individual in the first place of the relation need *not* to be encoded in different time slices. Furthermore, the original entities p and c are linked to perdurants P and C which, however, only need to be created once.

The 4D reinterpretation is easier than Welty&Fike's original formulation, viewed from the standpoint of complexity. Let us have a look at the domain (D) and range (R) of the above *hasCeo* property, using abstract description logic syntax:

- **Welty & Fikes (2006)**
  (D) $\exists \mathsf{hasCeo} \,.\, \top \sqsubseteq \forall \mathsf{hasTimeSlice}^{-} \,.\, \mathsf{Company}$
  (R) $\top \sqsubseteq \forall \mathsf{hasCeo} \,.\, (\forall \mathsf{hasTimeSlice}^{-} \,.\, \mathsf{Person})$

- **4D reinterpretation**
  (D) $\exists \mathsf{hasCeo} \,.\, \top \sqsubseteq \mathsf{Company}$
  (R) $\top \sqsubseteq \forall \mathsf{hasCeo} \,.\, \mathsf{Person}$

As we have already noticed, this reinterpretation also makes it easy to interface arbitrary ontologies with existing time ontologies. We will see this in a moment.

4. The Perdurant Ontology

Given the above discussion, this section now presents the basic ontology for perdurants and time slices used in the MUSING project that is, however, directly applicable to other applications and projects that deal with changing relationships over time in RDF. Here is the overall picture:

Perdurant: hasTimeSlice
TimeSlice: timeSliceOf, hasTime
Time

Let us describe the three top-level classes that are only necessary. Objects whose properties change over time are called *perdurants*, as already explained above. Those objects possess a number of *time slices*, hence we need a property hasTimeSlice in order to access their time slices. A time slice specifies an extension in time through the functional property hasTime and is associated with a perdurant via timeSliceOf, the inverse relation to hasTimeSlice. A time slice "contains" those properties whose values stay constant over the specified period of time. The range of hasTime is exactly an object of class Time which will be described in a moment:

$\top \sqsubseteq \leq 1 \mathsf{hasTime} \sqcap \forall \mathsf{hasTime}.\mathsf{Time}$

We note here that this simple ontology is completely open to the choice of the *time ontology* (and open to the upper/domain ontology that is equipped with a concept of time). Thus it will be possible to interface the perdurant ontology with popular time ontologies, such as Hobbs&Pan's OWL Time (Hobbs and Pan, 2004). This is achieved through the above mentioned class Time, a simple placeholder that is interfaced with the corresponding class in the time ontology. Similarly, the placeholder TimeSlice needs to be interfaced with the corresponding concept(s) in the upper/domain ontology. This will be shown in section 5.

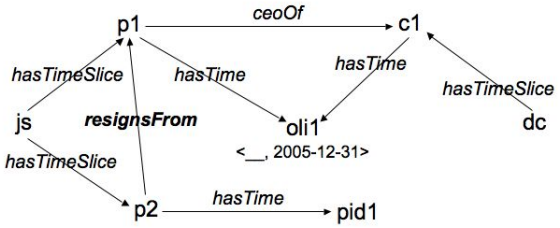
4.1. Flexible Semantic Representation

Let us focus on a natural language example and its (simplified) representation to see how things go together:

DaimlerChrysler's CEO Schrempp announces that he will resign by 31st December 2005.

Consider that an information extraction system has find out that Jürgen Schrempp and DaimlerChrysler are named entities. Consequently, we introduce two perdurants *js* and *dc* for these entities (assuming that they have not already been introduced).

The fact that Schrempp was CEO of DC until 31st December 2005 is expressed by a time slice p_1 (of type Person) that contains an instance oli_1 of class OpenLeftInterval, whereas his resignation is encoded in a time slice p_2 (again of type Person) that is temporally anchored in an instance pid_1 which is of class ProperInstantDay, having value "2005-12-31".



Notice that Schrempp did not resign from DC, but instead resigned from DC's ceoship. Thus property resignsFrom points to p_1 that expresses Schrempp's ceoship with DaimlerChrysler.

4.2. Advantages of the Approach

Firstly, properties that do not change over time (e.g., birthdate) can be relocated from TimeSlice to Perdurant (no duplication of information). Time-varying information instead is kept in a series of time slice. If several properties of a perdurant are constant over the *same* period of time, we do not need several time slices.

Secondly, the subtypes of TimeSlice specify the *behavior* of a *perdurant* within a certain time interval (e.g., whether a perdurant *acts* as a company, a person, etc.). We will see in a moment how this can be achieved.

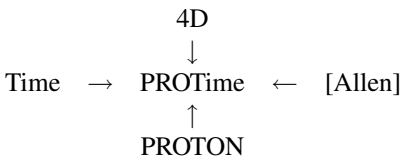
Thirdly, since hasTimeSlice is typed to TimeSlice, different slices of the same perdurant need *not* to be of the same type. For instance, the perdurant SRI might have a time slice for Company as well as a slice for AcademicInstitution, i.e., a perdurant can act in different ways.

Fourthly, representing modalities, such as *believe* can be achieved relatively easy. Representing space and movements in space can be modeled similarly.

Finally, Allen's 13 temporal topological interval relations (Allen, 1983) can be naturally extended to time slices.

5. Extending Ontologies With Time

As promised, we now describe how we have interfaced the 4D and the time ontology with an upper/domain ontology, in our case PROTON (http://proton.semanticweb.org).
Before going into the details, let us remark that our global ontology consists of concepts and properties that implement a 4D perdurantist view, but also deals with time in general, building on instants and intervals (and their subclasses). So we get the following picture for the merged ontology PROTime:



The 4D reinterpretation which we have presented so far says that the *original* entities should be regarded as *time slices*. To do so, one need to identify the most general classes in PROTON (or in another arbitrary upper/domain ontology) that are supposed to be extended by a temporal dimension—actually, we are interested in the domain/range classes of the time-varying properties. There is such a single, most general class in PROTON: psys:Entity. Thus we only need a single axiom, employing `owl:equivalentClass`:

fourd:TimeSlice $\equiv$ psys:Entity

In general, a new integrated ontology is constructed as follows:

1. **always use 4D**
   Perdurant: hasTimeSlice
   TimeSlice: timeSliceOf, hasTime
   Time
2. **choose time**
   an arbitrary time ontology (e.g., OWL Time)
3. **choose upper/domain ontology**
   the original ontology (e.g., PROTON)
4. **choose Allen (optional)**
   Allen relations over time slices

plus an equivalence statement of the above kind.

Note that the class Time in the 4D ontology is a simple placeholder used in hasTime $\subseteq$ TimeSlice $\times$ Time. When interfacing 4D with an arbitrary time ontology, one needs to say what is meant by Time, in our case:

fourd:Time $\equiv$ time:TemporalEntity

We will describe TemporalEntity and the time ontology in the next section.

In case there will be several maximal incompatible classes $c_1, \ldots, c_n$ that need to be extended by a temporal dimension, the above axiom clearly becomes

fourd:TimeSlice $\equiv c_1 \sqcup \ldots \sqcup c_n$

6. The Time Ontology

In this section, we will describe the time ontology that we have employed in MUSING. We have opted against OWL Time (Pan, 2007), a rich first-order axiomatization of time, since we have decided to model temporal underspecification in natural language and granularity of time through a subtyping hierarchy. The ontology described here, however, is fully compatible with OWL Time through the use of the class TemporalEntity as well as its subclasses Instant and Interval. Here is the overall picture:

TemporalEntity
  Instant
    NegativeInfinity
    PositiveInfinity
    ProperInstantYear: year
      ProperInstantMonth: month
        ProperInstantDay: day
          ProperInstantHour: hour
            ProperInstantMinute: minute
              ProperInstantSecond: second
                .....
  Interval: begins, ends
    OpenLeftInterval
      ClosedInterval
        Forever
        .....
    OpenRightInterval
      ClosedInterval
        Forever
        .....

OWL *classes* start with uppercase letter characters; *properties* are written in lower case. Thus

Interval: begins, ends

means that properties begins and ends are defined on class Interval. *Indentation* expresses subtyping/subclassing. Subtyping also means that properties defined on superclasses are also available in subclasses. Hence, the properties year and month are also accessible in class ProperInstantDay.

Let us quickly describe the most top-level classes. We distinguish between two exhaustive partitioning and disjoint subclasses of TemporalEntity: Instant and Interval.

TemporalEntity $\equiv$ Interval $\sqcup$ Instant

Interval $\sqsubseteq \neg$ Instant

Instant is used to describe infinitely short events (i.e., instants), whereas Interval identifies measurable periods of time. Thus, Interval possesses two properties: begins and ends, both returning an instant. All classes above are expressed as OWL axioms.

We now give a more complex example—the definition of ClosedInterval:

ClosedInterval $\equiv$
  OpenLeftInterval $\sqcap$ OpenRightInterval $\sqcap$
  =1begins $\sqcap$ =1ends $\sqcap$
  $\exists$ begins.Instant $\sqcap$ $\exists$ ends.Instant

This definition says that begins and ends must be specified exactly once. begins and ends must furthermore be assigned an instance of (at least) type Instant.

ProperInstantYear, PositiveInfinity, and NegativeInfinity are declared as being mutually disjoint:

ProperInstantYear $\sqsubseteq \neg$ NegativeInfinity

ProperInstantYear $\sqsubseteq \neg$ PositiveInfinity
PositiveInfinity $\sqsubseteq \neg$ NegativeInfinity

Actually, saying that begins takes exactly one value is done in the direct superclass OpenRightInterval (same for ends and class OpenLeftInterval). begins and ends are being declared as functional on the very general Interval class. Functionality clearly means that a value need not to be present (as can be seen, e.g., for property ends in class OpenRightInterval):

$\leq$1begins $\sqsubseteq$ Interval
$\leq$1ends $\sqsubseteq$ Interval

begins and ends furthermore take objects of type Instant as values:

$\top \sqsubseteq \forall$begins.Instant
$\top \sqsubseteq \forall$ends.Instant

Given NegativeInfinity and PositiveInfinity, the definition for the time period Forever is easy:

Forever $\equiv$
ClosedInterval $\sqcap$
$\exists$begins.NegativeInfinity $\sqcap$
$\exists$ends.PositiveInfinity

ClosedInterval has further subclasses that we only mention here:

ClosedInterval
Day
Monday, Thuesday, ...
SpecialDay
Christmas, NewYearsEve
Month
January, February28, February29, ...
Quarter
FirstQuarter, SecondQuarter, ...
Season
Spring, Summer, ...
Year
Year365, Year366

Let us finally focus in this section on the definition of two of these classes in order to flesh out this framework, viz., Day and NewYearsEve:

Day $\equiv$
ClosedInterval $\sqcap$
$\exists$ begins.ProperInstantDay $\sqcap$
$\exists$ ends.ProperInstantDay

NewYearsEve $\equiv$
SpecialDay $\sqcap$
$\exists$begins.($\exists$month.{12} $\sqcap$ $\exists$day.{31}) $\sqcap$
$\exists$ends.($\exists$month.{12} $\sqcap$ $\exists$day.{31})

It is worth noting that even though we have specified a value for properties month and day, the definition of NewYearsEve misses the value for year. But this is correct and only get assigned in examples such as *New Year's Eve 2007* which will be modeled as an instance of class NewYearsEve, having value 2007 for property year. Otherwise, such an expression is underspecified w.r.t. to the value of year, as in the sentence *Over New Year's Eve I have visited the Eiffel Tower.*

Further subclasses of Instant and ClosedInterval help to deal with the *granularity* of time and the *underspecifiction* of time in natural language. We will address this in the next section.

7. Granularity and Underspecification

Granularity of time, i.e., the degree of how finely time is measured and the *temporal underspecification* of natural language expressions are closely related topics. Consider, for instance, the following example:

> *In 1995, Edzard Reuter handed over the CEOship of Daimler Benz AG to Schrempp.*

and assume that a year is the smallest amount of time that we want to measure. Thus the starting point for enriching the RDF triple

```
js ceoOf db .
```

is 1995 and this temporal information will be encoded via an instance of class ProperInstantYear–remember, we measure things no finer than a year. Since ProperInstantYear only possesses the property year and since this year is known, 1995 is a *fully specified* temporal expression, according to the measure we have applied.
Independent of the degree of measurement, one can clearly ask what is meant by *1995* here. Within the above context, 1995 probably does not refer to the instant 1995-01-01T00:00:00, assuming we would measure even seconds. Instead, 1995 expresses the fact that there *exists* an *interval* that *starts* somewhere in 1995 in which Schrempp started his CEOship with Daimler Benz. Since the temporal end point of the above fact is *not* known at this moment (but the starting point) and since the time of Schrempp's CEOship is probably not infinitely small, we encode this interval information in an instance of class OpenRightInterval.
This very simple example shows that temporal underspecification happens to appear on two levels:

1. instances of Instant might be underspecified in case not every property (year, month, day, ...) has been given a value;

2. instances of Interval might be underspecified in case its properties begins and/or ends have not been given a value *or* in case begins and/or ends are assigned a value (instances of Instant), this value is underspecified.

The recursive part of this definition for temporal underspecification is applied in the following sentence:

> *Between 1995 and 2005, Schrempp was the CEO of DC.*

Now assume our fineness of time is measured in terms of days, thus we generate two instances of ProperInstantDay that fill the slots begins and ends of an instance of ClosedInterval. Even though this interval is closed, its beginning and end points are underspecified, hence this closed interval is regarded as being underspecified. If we, however, had measured time in terms of years, the above natural language description would have led to a totally specified

closed interval. It should be clear that further textual information might close an open-left/open-right interval. Textual information might even make a partially underspecified instant or interval total.

The above examples are fully compatible with the property restrictions imposed on begins, ends, year, month, day, etc., viz., being functional properties (0 or 1 value). In case we want to enforce a property to be instantiated, e.g., that begins and ends are "present" on ClosedInterval, we have applied a local number restriction on this specific class (see description logic axioms above).

We finally note that our approach to underspecification is a result of the subclass hierarchy of proper instants which applies a more finer measuring system when moving down the classes. An alternative, albeit less satisfying approach to underspecification would apply 0/1 cardinality constraints to the properties year, month, etc. in order to "switch them off/on", depending on the predefined granularity of time.

8. An Application

Let us focus on an application that uses the above time ontology and the methodology to represent temporally changing information: *imprint monitoring*. The monitoring system described in (Federmann and Declerck, 2010) extracts imprints (and other information) from a large number of companies on a regular temporal basis. Imprints specifies, e.g., the name of a company, the postal address, its legal form, authorized executives, etc. This information and its change over time is interesting for rating agencies (such as Creditreform).

In case the imprint of a perdurant `perd` changes at time `t` (w.r.t. information recorded in the ontology), the latest time slice `old` of `perd` is closed, using `t` (actually its time interval `oldint`). A new time slice `new` (of type `Company`) is also added to the ontology, storing the new imprint. Since `new` contains the latest information whose temporal ending point is unknown, `t` is stored as the starting point of an `OpenRightInterval`. Not only new triples are build up here, but also new individuals/URIs: besides `new`, an interval object `newint` is generated. More formally, we construct the following RDF triples:

```
old fourd:hasTime oldint .
oldint rdf:type time:ClosedInterval .
oldint time:ends t .
perd fourd:hasTimeSlice new .
new rdf:type Company .
new fourd:hasTime newint .
newint rdf:type time:OpenRightInterval .
newint time:begins t .
.....    // add imprint info to new
```

Information from the ontology can be queried using the SPARQL query language, as is used to obtain the latest time slice. The ontology, the reasoning and querying services are realized by the CROWL system (Combining Rules and OWL). CROWL consists of several publicly available reasoners (viz., Pellet (Sirin et al., 2007), OWLIM (Kiryakov, 2006), and Jena (Reynolds, 2009)), running in a fixpoint loop, and is extended by a template language to implement complex aggregation rules (Krieger et al., 2008b).

9. Problems: An Example

As we indicated in the introduction, even though our approach keeps the original ontology, it leads to a massive proliferation of objects, making reasoning and querying unnecessarily complex, expensive, and error-prone. This is due to the underlying data structure, the RDF triple, and the approaches presented in section 3. do suffer from the same problem.

Let us present an example to see how complexity builds up, even for a relatively easy task. This example will then be used in the next section when a solution is presented. The task we want to achieve is the following:

> *Compute maximal intervals, given a property, e.g.,* `ceoOf`*, between time slices* `?p` *and* `?c`*.*

Such queries often arise in practice when temporally-anchored facts need to be extended by further incoming information. Our approach, as described in section 3.5., would require a "lengthy" Jena-like heuristic rule to solve this task, impossible to formulate in OWLIM or Pellet, since it employs two aggregates, as realized by the functions `Min2` and `Max2`:

```
?p rdf:type fourd:Perdurant
?p fourd:hasTimeSlice ?ts1
?p fourd:hasTimeSlice ?ts2
?ts1 ceoOf ?obj1
?ts1 rdf:type ?tstype
?obj1 fourd:timeSliceOf ?q
?obj1 rdf:type ?objtype
?ts2 ceoOf ?obj2
?obj2 fourd:timeSliceOf ?q
?ts1 fourd:hasTime ?i1
?ts2 fourd:hasTime ?i2
?i1 time:begins ?b1
?i1 time:ends ?e1
?i2 time:begins ?b2
?i2 time:ends ?e2
->
?ts rdf:type ?tstype
?p fourd:hasTimeSlice ?ts
?ts ceoOf ?obj
?obj fourd:timeSliceOf ?q
?obj rdf:type ?objtype
?ts fourd:hasTime ?i
?obj fourd:hasTime ?i
?i rdf:type time:ClosedInterval
?i time:begins ?min
?i time:ends ?max
?i time:ends ?max
@test
?ts1 != ?ts2
@action
?min = Min2 ?b1 ?b2
?max = Max2 ?e1 ?e2
```

Independent of the underlying approach, we immediately feel that such a rule is hard to manage and expensive, both in terms of time (when matching clauses) and space (when introducing new objects/URIS, bound to `?ts`, `?obj`, `?i`, `?min`, and `?max`.

10. A Solution

The solution we propose in this section has been realized in the reasoning engine *HFC*, developed at DFKI. The idea here is to move from RDF triples to tuples in order to extend relation instances with further (temporal) arguments, as already described in section 3.1.

To achieve this goal, we also need to conservatively extend RDFS and OWL entailment rules, as originally described in (Hayes, 2004) and (ter Horst, 2005), i.e., to make these rules sensitive to temporal information. Here are three instantiated examples that show how things are supposed to work.

Assuming that *hasCeo* is the *inverse* of *ceoOf* and that our ontology has been populated with the fact that Jürgen Schrempp was DC's CEO from 1995 until 2005, represented as $ceoOf(js, dc, 1995, 2005)$, we would then like to deduce that $hasCeo(dc, js, 1995, 2005)$ also holds.

The fact that Angelina Jolie was married with Billy Bob Thornton from 2000 until 2003 is represented by $marriedWith(aj, bbt, 2000, 2003)$. Given that *marriedWith* is a *symmetric* property, the following should also be the case: $marriedWith(bbt, aj, 2000, 2003)$.

Given that my office is part of the DFKI building, i.e., $contains(dfki, room{+}1.26, 1990, 2010)$ and that my old office chair was replaced in 2002, i.e., $contains(room{+}1.26, chair42, 2002, 2010)$, we are allowed to infer that new chair is (at least) inside the DFKI since 2002 ($contains(dfki, chair42, 2002, 2010)$), due to the *transitivity* of the containment relation.

Such behavior can be formalized through temporally-extended entailment rules, quite similar to the "untensed" version described in (Hayes, 2004) and (ter Horst, 2005). As we indicated above, the temporal arguments are attached to the original triples, thus we end up in quintuples, assuming that we have a starting and ending time. Here are some examples:

- *?p is inverse of ?q*

```
?p owl:inverseOf ?q
?s ?p ?o ?t1 ?t2
->
?o ?q ?s ?t1 ?t2
```

- *?p is a symmetric property*

```
?p rdf:type owl:SymmetricProperty
?s ?p ?o ?t1 ?t2
->
?o ?p ?s ?t1 ?t2
```

- *?p is a transitive property*

```
?p rdf:type owl:TransitiveProperty
?x ?p ?y ?t1 ?t2
?y ?p ?z ?t3 ?t4
->
?x ?p ?z ?t5 ?t6
@action
?t5 = Max2 ?t1 ?t3
?t6 = Min2 ?t2 ?t4
```

- *copy subject for owl:sameAs*

```
?x owl:sameAs ?y
?x ?p ?z ?t1 ?t2
->
?y ?p ?z ?t1 ?t2
```

- *enforce domain restriction*

```
?p rdfs:domain ?dom
?s ?p ?o ?t1 ?t2
->
?s rdf:type ?dom
```

- *universal instantiation*

```
?i rdf:type ?c ?t1 ?t2
?c rdfs:subClassOf ?d
->
?i rdf:type ?d ?t1 ?t2
```

Note that only relation instances from the ABox are (usually) extended with temporal information—at the moment, we do not think that terminological knowledge needs to be equipped this way (e.g., that the domain/range restrictions of a property or the subtype relation between two classes only hold for some period of time).

Let us now come back to the example from the previous section that tries to build a contiguous interval from its two input intervals. Here is the new version:

```
?p ceoOf ?c ?b1 ?e1
?p ceoOf ?c ?b2 ?e2
->
?p ceoOf ?c ?min ?max
@test
?b1 != ?b2
?e1 != ?e2
@action
?min = Min2 ?b1 ?b2
?max = Max2 ?e1 ?e2
```

This is clearly much simpler and extremely intuitive: the only two clauses in the antecedent deal with the CEOship of a person with a company at different times and the single consequent extends the CEOship to a larger time span.

Such a rule can even be generalized to arbitrary properties which persist through time in a similar way. Assuming that such properties are characterized as subproperties of `ContinuousProperty`, the above rule becomes

```
?r rdfs:subPropertyOf ContinuousProperty
?p ?r ?c ?b1 ?e1
?p ?r ?c ?b2 ?e2
->
?p ?r ?c ?min ?max
.....
```

Even though the old rule is extremely complex, both rules only "look" at two intervals. Now, assuming that we want to glue n intervals together, both rules require $n-1$ iterations to compute the maximal interval. The number of rule applications is even larger: $(n-1) \times 2 \times \sum_{i=1}^{n-1} i$.

In order to overcome this last obstacle, we need *aggregation rules* that differ from ordinary rules in that variables do not bind only one individual at a time, but all individuals, satisfying the left-hand side constraints and the tests. This is quite similar to aggregates as used in query languages

(e.g., in SQL), except that the queried information is used to instantiate further tuples which are then added to the ontology.

HFC provides us with such aggregation rules. The above rule even becomes more simple; the important point, however, is that one rule application immediately yields the maximal interval. Note the different arrow sign => to indicate that the below rule aggregates information through `MinN` and `MaxN`:

```
?p ceoOf ?c ?b ?e
=>
?p ceoOf ?c ?min ?max
@action
?min = MinN ?b
?max = MaxN ?e
```

11. Conclusion

In this paper, we have presented two approaches that are able to enrich arbitrary ontologies with a concept of time.

The first approach implements a 4D or perdurant view on temporally-changing information, complemented by a sophisticated time ontology that permits temporal underspecification. This approach keeps the original ontology and does not leave the territory of RDF. This approach was used in the MUSING project.

The lessons, we learned in MUSING, have led us to a second approach that is much simpler, more expressive, and more efficient, but requires to move from RDF triples to general tuples. Temporal information here is directly attached to the relation instance. We have indicated how RDFS and OWL entailment rules can be conservatively extended to make them sensitive to temporal information. This approach is currently employed in the CogX project.

12. References

James F. Allen. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843.

Christian Federmann and Thierry Declerck. 2010. Extraction, merging, and monitoring of company data from heterogeneous sources. In *Proceedings LREC 2010*.

Patrick Hayes. 2004. RDF semantics. Technical report, W3C.

Jerry Hobbs and Feng Pan. 2004. An ontology of time for the Semantic Web. *ACM Transactions on Asian Language Processing (TALIP)*, 3(1):66–85.

Atanas Kiryakov. 2006. OWLIM: balancing between scalable repository and light-weight reasoner. Presentation of the Developer's Track of WWW2006.

Graham Klyne and Jeremy J. Carroll. 2004. Resource description framework (RDF): Concepts and abstract syntax. Technical report, W3C. 10 February.

Hans-Ulrich Krieger, Bernd Kiefer, and Thierry Declerck. 2008a. A framework for temporal representation and reasoning in business intelligence applications. In *AAAI 2008 Spring Symposium on* AI Meets Business Rules and Process Management, pages 59–70. AAAI.

Hans-Ulrich Krieger, Bernd Kiefer, and Thierry Declerck. 2008b. A hybrid reasoning architecture for business intelligence applications. In *8th International Conference on Hybrid Intelligent Systems, HIS-2008*, pages 843–848. IEEE.

John McCarthy and Patrick J. Hayes. 1969. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press.

Deborah L. McGuinness and Frank van Harmelen. 2004. OWL Web Ontology Language Overview. Technical report, W3C. 10 February.

Feng Pan. 2007. *Representing Complex Temporal Phenomena for the Semantic Web and Natural Language*. Ph.D. thesis, University of Southern California.

Dave Reynolds. 2009. Jena 2 inference support (version 1.40). http://jena.sourceforge.net/inference/index.html.

Theodore Sider. 2001. *Four Dimensionalism. An Ontology of Persistence and Time*. Oxford University Press.

Evren Sirin, Bijan Parsia, Bernardo Cuenca-Grau, Aditya Kalyanpur, and Yarden Katz. 2007. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2).

Herman J. ter Horst. 2005. Combining RDF and part of OWL with rules: Semantics, decidability, complexity. In *Proceedings of the International Semantic Web Conference*, pages 668–684.

Christopher Welty and Richard Fikes. 2006. A reusable ontology for fluents in OWL. In *Proceedings of Fourth International Conference on Formal Ontology in Information Systems (FOIS)*, pages 226–236.