A Framework for Temporal Representation and Reasoning in Business Intelligence Applications*

Hans-Ulrich Krieger, Bernd Kiefer, Thierry Declerck
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
{krieger,kiefer,declerck}@dfki.de

Abstract

This paper presents a framework for temporal representation and reasoning in the MUSING project (http://www.musing-project.eu) which is dedicated to the investigation of semantic-based business intelligence solutions. Temporal information is based on a diachronic representation of time. Since ontological knowledge in MUSING is encoded in OWL (Smith, Welty, & McGuinness 2004), extending binary relations with time is not that easy, due to the fact that OWL (or description logic in general) only provides unary and binary relations (Baader *et al.* 2003). To do so, we need the notion of a time slice (Sider 2001). Contrary to (Welty, Fikes, & Makarios 2005), we directly interpret the original entities as time slices in order to avoid a duplication of the original ontology and to prevent a knowledge engineer from ontology rewriting. We will see that this reinterpretation makes it easy to extend an arbitrary upper/domain ontology with the concept of 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 DERI, Innsbruck. We describe how the time ontology has been interfaced with PROTON and how it can be interfaced with OWL-Time (Hobbs 2004). In the last third of this paper, we explain our choices that have led to a specific reasoning architecture in MUSING, based on Pellet, OWLIM, and Jena, and backed up by Sesame.

The MUSING Project

MUSING is an R&D European project dedicated to the development of a new generation of Business Intelligence (BI) tools and modules based on knowledge and content systems. MUSING integrates Semantic Web and Human Language technologies and will combine declarative rule-based methods and statistical approaches for enhancing the technological foundations of knowledge acquisition and reasoning in BI applications. The impact of MUSING on semantic-based BI is being measured in three strategic vertical domains:

- **Finance.** Development and validation of next generation (Basel II and beyond) semantic-based BI solutions, with particular reference to credit risk management;
- **Internationalization.**[1] Development and validation of semantic-based internationalization platforms;
- **Operational Risk Management.** Development and validation of semantic-driven knowledge systems for measurement and mitigation tools, with particular reference to operational risks faced by IT-intensive organizations.

In all those domain-specific application scenarios, time is playing a crucial role. Much of the temporal information is being automatically extracted from structured (e.g., balance sheets) and unstructured (e.g., financial news wires) documents, using information extraction systems. A base for the detection of time expressions in texts has been defined by the TimeML framework (Pustejovsky *et al.* 2003) which is describing guidelines for the annotation of event and temporal expressions in textual documents. But as (Pratt-Hartmann 2005) has shown, the kind of annotation provided by TimeML is not enough for supporting logical operations on time. And we expect in MUSING that temporal reasoning is playing a crucial role in supporting decision procedure for the three application domains mentioned above. Thus we cannot remain at the level of the TimeML annotation framework for marking temporal expressions in MUSING documents (in German, English, and Italian).

This is the reason why we have opted for a temporal ontology, on top of which temporal reasoning procedures can be defined. Since the domain ontologies developed and used in MUSING are already represented in OWL, we opted for an ontological representation of time that is compatible with the restrictions imposed by OWL, viz., unary and binary predicates, in order to guarantee decidability of the usual inference problems.

*The authors would like to thank our three reviewers for their encouraging and detailed comments. The research described in this paper has been partially financed by the European Integrated Project MUSING under contract number FP6-027097.

[1] Internationalization refers to the process that allows an enterprise to evolve its business from a local to an international dimension, hereby focusing on the information acquisition work concerning international partnerships, contracts, and investments.

A Motivating Example

The problem with *synchronic* relationships is that they all refer to one (hidden) point in time. Here is an example:

> Dieter Zetsche ist der neue Vorstandsvorsitzende von DaimlerChrysler.
> *Dieter Zetsche is the new CEO of DaimlerChrysler.*

Assuming a triple-based RDF representation (Manola & Miller 2004), an information extraction system might compute the following set of triples:

```
<dz,rdf:type,Person>
<dc,rdf:type,Company>
<dc,hasCeo,dz>
```

`dc` and `dz` are individuals (of type `Company` and `Person`, resp.) that have been introduced in order to refer to *DaimlerChrysler* and *Dieter Zetsche*, resp.

However, most relationships are *diachronic*, i.e., they vary with time. Take, for instance, the following example:

> Jürgen Schrempp gibt bekannt, daß er zum 31. Dezember 2005 als Vorstandsvorsitzender von DaimlerChrysler ausscheiden wird.
> *Jürgen Schrempp announces that he will resign by 31st December 2005 from DaimlerChrysler as its CEO.*

Given this, our information extraction system might discover the following "tensed" facts (the new individual `js` refers to *Jürgen Schrempp*):

t_1 = 2005-12-31: `<js,resignsFrom,dc>`
$? \leq t_2 \leq$ 2005-12-31: `<js,ceoOf,dc>`

Let us continue this example. Assume that we find more information about Jürgen Schrempp in another document:

> 1995 gab Edzard Reuter den Vorstandsvorsitz der Daimler Benz AG an Schrempp ab.
> *In 1995, Edzard Reuter handed over the CEOship of Daimler Benz AG to Schrempp.*

This will lead to the following facts:

t_1 = 1995: `<er,precedes,s>`
$? \leq t_2 \leq$ 1995: `<er,ceoOf,db>`
1995 $\leq t_3 \leq$?: `<s,ceoOf,db>`

Given these two text snippets, identity tracking of individuals now becomes an important point, i.e., we need to identify entities that are refered to by different textual referential expressions, e.g., *Jürgen Schrempp*, *Schrempp* (named entities), *he* (third singular masculine personal pronoun), or *[the] CEO of DaimlerChrysler* (nominal phrase).

OWL, in fact, can help us here in that we identify individuals that have already been introduced so far. This can be achieved using the `owl:sameAs` construct (`s` = *Schrempp*, `db` = *Daimler Benz AG*):[2]

```
<js,owl:sameAs,s>
<dc,owl:sameAs,db>
```

[2]It should be noted here that many information extraction systems already perform the task of identifying co-referent textual expressions at the level of linguistic analysis. At the same time, we can not be sure whether such a processing step is complete.

For explanatory purpose, we assume here that *DaimlerChrysler* refers to the same company as *Daimler Benz AG* refers to. Let us conclude this example by taking into account the following textual information:

> Er **ist** bei der Allianz AG und bei Vodafone Mitglied des Aufsichtsrats.
> *He* **is** *a member of the supervisory board of Allianz AG and Vodafone.*

This gives us two more triples associated with very *underspecified* temporal information, and again a `sameAs` identification between `e1` = *er* and `js` = *Jürgen Schrempp* (`a` = *Allianz AG*, `v` = *Vodafone*):

$t_1 \leq t \leq t_2$: `<e1,memberOfSupBoard,a>`
$t_3 \leq t \leq t_4$: `<e1,memberOfSupBoard,v>`
`<e1,owl:sameAs,js>`

Assuming a document that contains this sentence and having publication date t, we might impose the assertion
$t_1 = t_2 = t_3 = t_4 = t$
to have a safe time point where the above proposition holds. Note that this heuristics only works if our information extraction system recognizes that the above sentence is in *present tense*.

We have already noticed that most relationships are diachronic, i.e., they embody the possibility to vary with time. With this in mind, we make the following two fundamental observations:

1. The value of a property is only valid within a certain, possibly underspecified time interval.
   Example: CEOship above.
2. A property need not hold for each subinterval (a.k.a. *subinterval inheritance*).
   Example 1: Die Deutsche Bank steigerte ihre Ergebnisse vor Steuern in 2005 um 58%. (*Deutsche Bank raised their profits before taxes in 2005 by 58%*). Here, we cannot assume that this raise is continuous through the whole year.
   Example 2: *Yesterday we drove west.* Clearly there have probably been subintervals during the travel, in which the driving direction was not west.

Diachronic Identity

Diachronic identity refers to the problem of identifying individuals that "look" different at different times, but refer to the same entity.

Endurants vs. Perdurants

Basically, there are two different approaches to diachronic identity: the *three-dimensional* or *endurantist* view and the *four-dimensional* or *perdurantist* view:

- In the 3D view, a distinction is made between *endurants* and *occurrants*. Endurants are wholly present, whereas occurrants have temporal parts. The problem of diachronic identity then reduces to the identification of finding the *essential* properties that always hold over some period of time. Thus hair color is obviously not an essential property of a person if we would talk over periods of time that might last 50 years or so.

- 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 *space-time worms* (Sider 2001), since a four-dimensional trajectory identifies a perdurant in time and space.

In MUSING, we have adopted the perdurantist view, taking only the time dimension into account, since we are not dealing with spatial information at the moment. To do so, we associate a perdurant with all its temporal parts. A temporal part is often referred to as a *time slice* when adopting the spacetime worm metaphor; see (Sider 2001).

Approaches to Diachronic Representation

There exist several well-known techniques for extending n-ary representations with additional arguments, or in our case, equipping binary relations with time. (Welty, Fikes, & Makarios 2005) present three of them—we add a fourth one and finally present a fifth way by reinterpreting the 4D view with respect to an upper or domain ontology. This reinterpretation is the basis for representing temporal information in the MUSING project.

Equip Relations With a Temporal Argument. This approach has been pursued in temporal databases and the logic programming community. Thus a binary relation, such as $hasCeo$ between a company c and a person p becomes a ternary or quaternary relation with a third/fourth temporal argument t:

$$hasCeo(c,p) \quad \longmapsto \quad \begin{array}{l} hasCeo(c,p,\underline{t}) \\ hasCeo(c,p,\underline{t_1,t_2}) \end{array}$$

Unfortunately, OWL and description logic (DL) in general only support unary (classes, concepts) and binary relations (properties, roles) in order to guarantee decidability of the usual inference problems. Thus the above extended relation is not directly expressible in OWL.[3]

Apply a Meta-Logical Predicate. McCarthy & Hayes' situation calculus (McCarthy & Hayes 1969), James Allen's interval logic (Allen 1984), and the knowledge representation formalism KIF (Genesereth 1991) use the meta-logical predicate $holds$. Hence, our $hasCeo$ relation becomes

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

However, the property language in OWL (and again, description logic in general) is weak. OWL only comes with two property forming operators, one inherited from RDF Schema: *rdfs:subPropertyOf* and *owl:equivalentProperty*. Neither complex property definitions nor complex relation arguments are allowed in OWL. Role composition, in fact, would be needed here:

$$holds \circ hasCeo$$

Next generation OWL 1.1 only comes up with a very restricted form of role composition in role axioms to guarantee decidability. Unfortunately, general role composition has been shown to be undecidable, even for the relatively weak sublanguage $\mathcal{ALR}$ of KL-ONE (Schmidt-Schauß 1988).

Reify the Original Relation. Reification as used in RDF refers to the ability to treat a whole statement as a resource (and so as an argument of a RDF triple). Reifying a relation instance leads to the introduction of a new object and four instantiated relations. In addition, a new class needs to be introduced for each reified relation, plus properties to access the original arguments. Furthermore and very important, relation reification *loses* the original relation and thus needs rewriting of the original ontology. Coming back to our *hasCeo* example, we get something like this (*HasCeo* is the newly introduced container class for *hasCeo*):

$$\begin{array}{l} \underline{hasCeo(dc,js,t)} \;\longmapsto\; \exists hc\,. \\ \quad type(hc,HasCeo) \wedge \\ \quad time(hc,t) \wedge \\ \quad company(hc,dc) \wedge \\ \quad person(hc,js) \end{array}$$

Wrap Range Arguments. This approach wraps some of the original arguments in a new object in order to reduce the arity of a given relation instance, thus sharing similarities to object wrapping in object-oriented programming languages, currying in functional programming, or skolemization in logic. However, we do not wrap the original arguments (first and second), but instead the arguments in the range of the relation. The reason for this comes from the fact that the relation argument in first place often serves as an anchor during reasoning/querying. Coming back to our example, the diachronic ternary relation becomes

$$\begin{array}{l} hasCeo(dc,\underline{js,t}) \;\longmapsto\; \exists et\,. \\ \quad type(et,EntityTime) \wedge \\ \quad hasCeo(dc,et) \wedge \\ \quad entity(et,js) \wedge \\ \quad time(et,t) \end{array}$$

The good thing about this approach is that it does *not* lose the original relation and only introduces one single general container class (called EntityTime) that is used throughout every wrapped argument pair.

Both relation reification and argument wrapping can clearly be applied to relations with more than three arguments. As is the case for reification, wrapping, of course, introduces a new object, but needs *less* ontology rewriting when compared to reification—remember, we do not lose the original relation. (Welty, Fikes, & Makarios 2005) have shown that some forms of built-in OWL reasoning are no

[3] Looking more closely at the extended relations, additional time arguments seem to behave more like an *annotation* than real individual arguments that need to be strongly considered during OWL reasoning. Only lightweight reasoning is probably needed here. Assume, we would have $ceoOf(js,dc,1995,2005)$ and $hasCeo$ has been defined as the *inverse* relation to *ceoOf*. Now it seems plausible to deduce that $hasCeo(dc,js,1995,2005)$ also holds (a similar argumentation also holds for *symmetric* and even *transitive* properties). As we said above, such temporal extensions to OWL are not avalable yet.

longer possible for relation reification. This fact also holds for the argument wrapping approach, but to a lesser extent.

Similar to an "untensed" representation while querying, we can still start from the original first argument to explore its properties. Consider, for example, the natural language question

Give me all CEOs of DaimlerChrysler.

This results in the following SPARQL (Prud'hommeaux & Seaborne 2007) queries for the "untensed" and the two diachronic representations:

(Untensed)
```
SELECT ?pers
WHERE {dc hasCeo ?pers}
```

(Reification)
```
SELECT ?pers
 WHERE {?ceo rdf:type HasCeo.
        ?ceo company dc.
        ?ceo person ?pers}
```

(Wrapping)
```
SELECT ?pers
 WHERE {dc hasCeo ?ent.
        ?ent entity ?pers}
```

Encode the 4D View in OWL (Welty, Fikes, & Makarios 2005) have given a compact implementation of the 4D or perdurantist view in OWL.[4] To do so, they define the notion of a time slice that encodes the time dimension of spacetime that is occupied by a spacetime worm. Relations from the original ontology no longer connect the original entities, but instead connect time slices that belong to those entities.

In fact, a time slice is merely a container for storing time only. For a given ontology, such a representation requires a lot of rewriting:

$$\underline{hasCeo(dc, js, t)} \longmapsto \exists ts_1, ts_2 \,.\, type(ts_1, TimeSlice) \wedge hasTimeSlice(dc, ts_1) \wedge type(ts_2, TimeSlice) \wedge hasTimeSlice(js, ts_2) \wedge time(ts_1, t) \wedge time(ts_2, t) \wedge hasCeo(ts1, ts2)$$

Reinterpret the 4D View In MUSING, we have reinterpreted the perdurantist/4D view in that we reinterpret the original entries:

What has originally been an entity now becomes a time slice.

In the example above, dc and js are no longer entities, but instead time slices, that explains the *behavior* of an entity within a certain extension or point in time (e.g., that dc is a time slice talking about a company).

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

$$hasCeo(dc, js, \underline{t}) \longmapsto hasCeo(dc, js) \wedge time(dc, t) \wedge time(js, t)$$

[4]The DOLCE OWL ontology (http://www.loa-cnr.it/DOLCE.html) as part of the WonderWeb infrastructure (http://wonderweb.semanticweb.org) has also implemented a 4D view using "time-snapshots" and makes a distinction between endurants and perdurants (Masolo *et al.* 2003).

In fact, this reinterpretation is easier than the original 4D formulation, viewed from the standpoint of complexity. Let us look at the domain (Dom) and range (Rng) of the above $hasCeo$ property, using DL abstract syntax:

- **(Welty, Fikes, & Makarios 2005)**

  (Dom) $\exists \mathsf{hasCeo} \,.\, \top \sqsubseteq \forall \mathsf{hasTimeSlice}^{-} \,.\, \mathsf{Company}$

  (Rng) $\top \sqsubseteq \forall \mathsf{hasCeo} \,.\, (\forall \mathsf{hasTimeSlice}^{-} \,.\, \mathsf{Person})$

- **4D reinterpretation**

  (Dom) $\exists \mathsf{hasCeo} \,.\, \top \sqsubseteq \mathsf{Company}$

  (Rng) $\top \sqsubseteq \forall \mathsf{hasCeo} \,.\, \mathsf{Person}$

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.

A Note on Temporal Description Logics

Synchronic relations, like `dateOfBirth`, "store" temporal information *directly* as the range value of a relation instance. Since there is only *one* date of birth, this works perfectly well and uniquely captures the intended meaning. The deeper, albeit simple reason why this works comes from the fact that synchronic relations are simply functional *and* the temporal extent is directly stored as the range value of the synchronic relation.

Diachronic relationships as shown above, however, vary with time, i.e., in general, they are *no* longer functional. By imposing a synchronic temporal representation scheme on a diachronic relationship, the association between the changing value and its temporal extend gets lost.

Temporal description logics (TDLs), such as (Lutz 2004), are great when we aim at representing *synchronic* relations. TDLs are essentially description logics extended with a concrete domain (Baader & Hanschke 1991), where either $\mathbb{Q}$ or $\mathbb{R}$ are used as its temporal structure. Temporal features (and other abstract features or attributes) are functional relations that can be employed to define paths which in turn can be used to define temporal concepts through the use of constructors, such as $<, \leq, =, \neq, >$, and $\geq$. These descriptive devices help to impose further constraints on concepts which ordinary description logics are not capable of.

TDLs as such are *not* capable of representing diachronic relationships directly. However, TDLs can complement our 4D representation scheme in that they are able to impose further constraints on temporal concepts (e.g., to say that the date of birth of a mother is *before* the date of her children). To the best of our knowledge, no current state-of-the-art DL reasoner (e.g., FaCT++, Racer, Pellet, KAON2) is equipped with such descriptive means.

A Time Ontology for MUSING

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

Perdurant: hasTimeSlice
TimeSlice: timeSliceOf, hasTemporalEntity

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 an uppercase letter characters; properties are written in lower case. Thus

TimeSlice: timeSliceOf, hasTemporalEntity

means that properties timeSliceOf and hasTemporalEntity are defined on class TimeSlice. 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 other top-level classes. Objects whose properties change over time are called perdurants (class: Perdurant), 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 property hasTemporalEntity and is associated with a perdurant via timeSliceOf, the inverse relation to hasTimeSlice, containing exactly those properties that changes over the specified period of time. The range of hasTemporalEntity is exactly an object of class TemporalEntity.

We distinguish between two exhaustive partitioning subclasses of TemporalEntity: Instant and Interval. 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:

$$\begin{aligned}&\mathsf{ClosedInterval} \equiv\\ &\quad \mathsf{OpenLeftInterval} \sqcap \mathsf{OpenRightInterval} \sqcap\\ &\quad =1\mathsf{begins} \sqcap =1\mathsf{ends} \sqcap\\ &\quad \exists\, \mathsf{begins.Instant} \sqcap \exists\, \mathsf{ends.Instant}\end{aligned}$$

This definition says that begins and ends must be specified exactly once and that they are assigned instances of (at least) type Instant.

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

$$\begin{aligned}&\mathsf{ProperInstantYear} \sqsubseteq \neg\, \mathsf{NegativeInfinity}\\ &\mathsf{ProperInstantYear} \sqsubseteq \neg\, \mathsf{PositiveInfinity}\\ &\mathsf{PositiveInfinity} \sqsubseteq \neg\, \mathsf{NegativeInfinity}\end{aligned}$$

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):

$$\begin{aligned}&\leq 1\mathsf{begins} \sqsubseteq \mathsf{Interval}\\ &\leq 1\mathsf{ends} \sqsubseteq \mathsf{Interval}\end{aligned}$$

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

$$\begin{aligned}&\mathsf{Forever} \equiv \mathsf{ClosedInterval} \sqcap\\ &\quad \exists \mathsf{begins.NegativeInfinity} \sqcap \exists \mathsf{ends.PositiveInfinity}\end{aligned}$$

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

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:

$$\begin{aligned}&\mathsf{Day} \equiv \mathsf{ClosedInterval} \sqcap\\ &\quad \exists\, \mathsf{begins.ProperInstantDay} \sqcap\\ &\quad \exists\, \mathsf{ends.ProperInstantDay}\end{aligned}$$

$$\begin{aligned}&\mathsf{NewYearsEve} \equiv \mathsf{SpecialDay} \sqcap\\ &\quad \exists \mathsf{begins.}(\exists \mathsf{month.}\{12\} \sqcap \exists \mathsf{day.}\{31\}) \sqcap\\ &\quad \exists \mathsf{ends.}(\exists \mathsf{month.}\{12\} \sqcap \exists \mathsf{day.}\{31\})\end{aligned}$$

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 modelled 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.*

Convex Intervals. Intervals in our ontology need *not* to be *convex*, i.e., they might contain *holes*—remember the *Yesterday we drove west* example. The interpretation of this sentence during the time interval of a (whole) day expresses that we mostly drove west, so that a propostion expressed

in a time slice must not necessary hold for subintervals between begins and ends. The above example could even indicate that there were moments when the car did not move altogether (e.g., during a service station stop). In general, many (most?) relations do not satisfy the *subinterval inheritance property*, but this interpretation is clearly up to an application. W.r.t. a specific domain, it makes perfectly sense to classify relations according to this dimension, viz., those that satisfy the subinterval inheritance property, and those that do not, using `rdfs:subPropertyOf`. Clearly, rules that operate on top of a temporally-enriched domain ontology can be made sensitive to this distinction in their antecedent.

Instants vs. Intervals. The time slice of a perdurant either refers to an instant or an interval (interval and instant are disjoint). This is why the (functional) property hasTemporalEntity is typed to TemporalEntity:

$$\mathsf{TemporalEntity} \equiv \mathsf{Interval} \sqcup \mathsf{Instant}$$

$$\mathsf{Interval} \sqsubseteq \neg\, \mathsf{Instant}$$

$$\top \sqsubseteq \leq 1 \mathsf{hasTemporalEntity} \sqcap \forall \mathsf{hasTemporalEntity}.\mathsf{TemporalEntity}$$

Here are two example, showing that in fact both instants and intervals are legal temporal concepts to which a time slice might refer.

> On January 1, 2002 [00:00:00], the Euro was officially introduced. (*instant*)
> Deutsche Bank raised their amounts before taxes in 2005 by 58%. (*interval*)

Further subclasses of Instant and ClosedInterval help to deal with the *granularity* of time and the *underspecifiction* of time in natural language. We come to this in a moment.

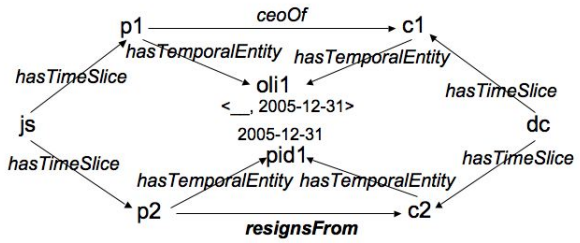
It is worth noting that we do *not* regard instants as degraded intervals, where begins = ends. In fact, stating that property begins has exactly the same value/refers to the same object than ends would require our description logic (OWL) to have role-value maps (and in general also role chains/composition):

$$\mathsf{Instant} \equiv \mathsf{Interval} \sqcap (\mathsf{begins}) \downarrow (\mathsf{ends})$$

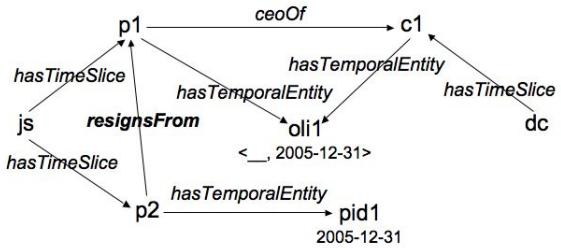
However, as (Schmidt-Schauß 1988) has shown, even the sublanguage $\mathcal{ALR}$ of KL-ONE is in general undecidable.[5] Furthermore, since we want to be compatible to OWL-Time (that uses the names Interval and Instant), we instead make Interval and Instant disjoint (see above) and solely define begins and ends on Interval. In doing this, we can simply read off the class of an individual (via `rdf:type`) to see whether we are looking at an interval or an instant.

Finding the Right Semantic Representation

Again, we use an example here to make things more clear. Recall the sentence from the beginning of this paper: *Jürgen Schrempp announces that he will resign by 31st December 2005 from DaimlerChrysler as its CEO.* Assume that an information extraction system finds 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 another *time slice* p_2 (again of type Person) that is temporally anchored in an instance pid_1 which is of class ProperInstantDay. Hence, we get the following picture:



Although our basic intuition says that this is a proper representation, the propositional content of the natural language sentence expresses something different—Schrempp did not resign from DC, but instead resigned from DC's ceoship (*Vorstandsvorsitzender von Daimler Chrysler*). Our approach is clearly capable of expressing this fact in that property resignsFrom no longer points to the original company time slice c_2, but instead to time slice p_1 that expresses Schrempp's ceoship:

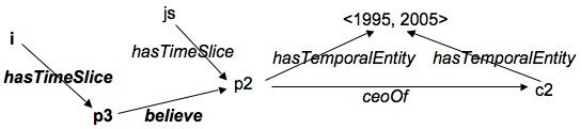


We note here that the above representation framework makes the representation of (higher-order) modalities, such as *believe*, easy. This is due to the fact that the properties that undergo a temporal change (those that are defined on TimeSlice and all its subclasses) again lead to time slices. In fact, a modality such as *believe* is similar—it is a property that connects special *time slices*. Consider the following example:

> I *believe* [that] Jürgen Schrempp was the CEO of DC between 1995 and 2005.

The following RDF graph depicts the application of the property believe to the propositional content that Schrempp was CEO from 1995 until 2005. Time slice p_3 of perdurant i (I) is related to time slice p_2 via property believe:

[5] At the same, role-value maps over functional properties have shown to be decidable (Nebel & Smolka 1989) (and begins and ends are functional). We would like to see this descriptive device in future extensions of OWL. These special role-value maps (in the TBox) are called coreference/agreement constraints or reentrancies in unification-based grammars (Shieber 1986).



Since p_3 is the time slice for the believe affair, it can even be equipped with temporal information that will ususally differ from the time of the nucleus sentence (here: 1995–2005).

Granularity and Underspecification

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

> 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 an explicit 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, assuming our fineness of time is measured in terms of days:

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

We thus 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.

Note that the above examples are fully compatible with the property restrictions imposed on begins, ends, year, month, 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).

Advantages of the Approach

Let us summarize the advantages of our approach. Firstly, properties that do *not* change over time (e.g., year of birth) can be relocated from TimeSlice to Perdurant (no duplication of information), thus a perdurant can even encode essential "endurantist" properties. Time-varying information instead is kept in a time slice. If several properties of a perdurant change over the *same* period of time, we do not need several time slices, but encode the change in a single slice.

Secondly, the subtypes of TimeSlice (e.g., Company, Person) 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 when the time ontology is getting interfaced with an upper-base and/or domain-specific ontology.

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/entity can act in different ways.

Fourthly, representing modalities, such as *believe* is achieved without any effort, as we have already seen above.

Finally, the approach makes it easy to accommodate 3-D space by simply adding a further property, say hasLocation, that encodes a 3-D trajectory that happens to take place during the temporal movement.

ProTime: Interfacing Time and PROTON

As promised, we now describe how we have interfaced the 4D time ontology with an upper/domain ontology, in our case PROTON (http://proton.semanticweb.org) and domain-specific extensions, relying on PROTON. These extensions were implemented by our colleagues from DERI Innsbruck.

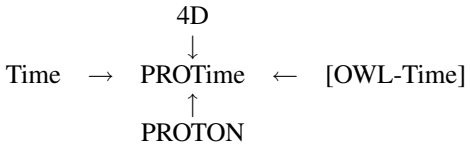
Before going into the details, let us remark that our time ontology consists of concepts and properties that implement a 4D perdurantist view, but also deals with time in general, building on instants and intervals (and their subclasses).

From Jerry Hobbs' OWL-Time (Hobbs 2004), we have only borrowed the class names TemporalEntity, Instant, and Interval, plus the two properties begins and ends. This very coarse classification has then been extended by

us through further subclasses and properties as explained above (e.g., OpenLeftInterval, Forever, year, second, etc.). Through the use of these very general names, an import of OWL-Time would extend our time ontology by further temporal topological relations from Allen's interval algebra, such as before (we have *not* opted for this in MUSING at the moment).

From (Welty, Fikes, & Makarios 2005), we have adopted the 4D treatment with the special reinterpretation, as explained above. This has given us the concept TimeSlice and property timeSliceOf. The property timeInterval has been renamed to hasTemporalEntity, since a time slice in our treatment can either refer to an Interval or to an Instant, thus to a TemporalEntity (the class from the general time ontology).

The third ontology which has been merged is a modified version of PROTON. Hence we get the following picture for the merged ontology PROTime (= PROTON + Time):



The 4D reinterpretation which we have presented above now says that the original entities should be regarded as time slices. To do so, one need to identify the most general class(es) in PROTON (or in another arbitrary upper/domain ontology) that are supposed to be extended by a temporal dimension. Of course, the properties defined on this class and its subclasses are related to this temporal information. PROTON has such a most general class: psys:Entity. Thus we only need an additional axiom:

time:TimeSlice ≡ psys:Entity

In general, a new integrated diachronic ontology imports the reinterpreted 4D treatment, a general time ontology, and the original ontology, plus an axiom of the above kind.

A Unified Reasoning Architecture

Since MUSING has decided to use OWL as the primary knowledge representation language, it was important for us to find a rule language on top of or parallel to OWL (and RDF/RDFS) that permits the representation of rule knowledge that is outside the expressiveness of OWL. The Semantic Web community has defined a minimal standard to fill this gap, viz., SWRL, the Semantic Web Rule Language (Horrocks *et al.* 2004).

In the following, we hope to show that that OWL can be fruitfully married with an implemented hybrid reasoning architecture, consisting of several well-known reasoners, each restricted to a specific area in which it behaves best.

Evaluated Systems

Two freely avalable reasoners which we have finally chosen are much in the spirit of SWRL, viz., OWLIM (Kiryakov 2006) (http://www.ontotext.com/owlim/) and Jena (Reynolds 2006) (http://jena.sourceforge.net/)—we will present them in a moment. Overall, there are not many other (partial) implementations of SWRLish languages:

- The latest version of **Pellet** (Kolovski, Parsia, & Sirin 2006) (http://pellet.owldl.com/) which we will use for initial TBox checking states that

  > Pellet has a *preliminary* implementation of a direct tableau algorithm for a DL-safe rules extension to OWL-DL. ... The initial empirical results are *encouraging*. We think that the DL-safe implementation is practical for *small to mid-sized ontologies*. [largest ontology: 46 classes, 30 object properties, 94 individuals, 2 rules]

- The user's guide (Racer Systems 2005) from the latest version of **Racer** (Haarslev & Möller 2001) (http://www.racer-systems.com/) notes that

  > A *first experimental* SWRL implementation is part of RacerPro 1.9.

  We were not able to activate the rule engine with RacerPro v1.9 (December 8, 2005) in a test setting with the LT-World ontology that backs up the language technology portal http://www.lt-world.org (see below), since inital TBox and ABox consistency checking required more than 1 GB main memory and was canceled after a day.

- **Jess**, "the Rule Engine for the Java Platform" (http://herzberg.ca.sandia.gov/jess/) is a general rule engine, developed at the Sandia National Lab by Ernest Friedman-Hill. Jess does not provide a native built-in OWL support, but via the Protégé's SWRL Jess tab (O'Connor *et al.* 2005), a mostly meaning-preserving translation from SWRL rules to Jess rules is possible. In case that a Jess rule produces new OWL individuals, however, Jess does not have any means to "replay" TBox consistency checking or ABox realization, since this would require predicate variables in entailment rules in Jess, similar to (Hayes 2004) or (ter Horst 2005).

- **KAON2** is a reasoning framework, implemented mainly by Boris Motik that supports most of OWL-DL, DL-safe SWRL rules, and F-logic. Contrary to tableaux-based systems such as Racer, Pellet, or FaCT++, KAON2 translates OWL ontologies into disjunctive datalog programs. KAON2 is faster than Pellet (v1.3 beta) and Racer when it comes to reasoning over large ABoxes, as shown in (Motik & Sattler 2006). However, when comparing the rule language of Pellet (v1.5.0) and KAON2, (Kolovski, Parsia, & Sirin 2006) claim that Pellet is superior to KAON2. Our decision against KAON2 is due to the following issues. KAON2 does not support nominals (with which some MUSING ontologies come up); it has a limited RDF(S) SPARQL support when using properties in subject and object position; it "only" provides the DL-safe subset of SWRL (Motik, Sattler, & Studer 2004); it does not support existential variables in the consequent of rules which we need in order to introduce new individuals; there is no way to have predicate quantification in SWRL rules; contrary to Pellet and Racer, it performs ABox consistency checking at query time.

Software Modules

OWLIM and Jena, the rule components in Pellet and Racer, as well as Jess are essentially forward-chaining or data-driven inference engines, performing a fixpoint computation w.r.t. a set of rules and an initial set of facts. Rules in a forward chaining engine can be used to define most of the *implicit* underlying OWL semantics as well as to encode *explicit* domain-specific rules. Axiomatic facts and entailment rules of the first kind can be found in (Hayes 2004) or (ter Horst 2005), e.g.,

```
→ (rdf:type rdf:type rdf:Property)

(?p rdf:type owl:SymmetricProperty),
(?s ?p ?o)
→ (?o ?p ?s)
```

As we see from this fact/rule, it is important to have RDF/OWL properties even in subject and object position of an RDF triple, as well as to allow property quantification, which syntactically reminds us of second-order predicate logic.

Domain-specific rules of the second kind are rules that incorporate numerical comparison and arithmetics, e.g.,

$$\forall x, y \,.\, \textit{Customer}(x) \wedge \textit{hasPurchased}(x, y) \wedge y > 1000000 \Rightarrow \textit{VIP}(x)$$

which we would write in Jena as[6]

```
(?x rdf:type Customer),
(?x hasPurchased ?y),
greaterThan(?x, 1000000)
→ (?x rdf:type VIP)
```

We note here that disjunctive rules, i.e., rules with a disjunctive head, as well as negation are excluded from the reasoning engines which are employed in our hybrid architecture. Disjunctive rules would require a disjunctive reasoning space (very expensive!). General negation in a forward-chaining setting no longer guarantees that rule application is order-independent. However, certain forms of negations can clearly be rewritten through the introduction of instances from `owl:Nothing`, the bottom type. Here is again an example from OWL-Time:

$$\forall T_1, T_2 \,.\, \textit{before}(T_1, T_2) \Rightarrow \neg \textit{before}(T_2, T_1)$$

We might restate this implication as

$$\forall T_1, T_2 \,.\, \textit{before}(T1, T2) \wedge \textit{before}(T_2, T_1) \Rightarrow \bot$$

Such a rule can also be seen as an integrity constraint as known from data bases. Using Jena notation, we have

```
(?ts1 before ?ts2),
(?ts2 before ?ts1),
makeTemp(?bottom)
→ (?bottom rdf:type owl:Nothing)
```

The above rule makes use of the Jena built-in `makeTemp()` that generates a fresh new individual, actually an RDF blank node. Such a rule is no longer *DL-safe* (Motik, Sattler, & Studer 2004) nor is translatable to SWRL (Horrocks *et al.* 2004), since it proposes an existential variable in its consequent. An alternative would instead to directly assign `owl:Nothing` as `rdf:type` of both `?ts1` and `?ts2`.

We will now focus on the reasoning components that we employ in our hybrid architecture and in which situations we intend to use them. Given this information, it should then be clear (at least there are good reasons for) that a reasoning component consisting of only a single reasoner (e.g., Jena) does not seem feasible at the moment.

Pellet Pellet (Sirin *et al.* 2007) (http://pellet.owldl.com/) is a description logic reasoner for OWL DL. Contrary to OWLIM and Jena (see below), Pellet is based on tableaux algorithms developed for expressive description logics (Baader & Sattler 2001), including all the features defined in the forthcoming OWL 1.1 standard. We opted for Pellet to perform initial TBox and ABox consistency checking and against Racer, since tests have shown that, at least in our setting, Racer did not scale up (as well as Jena did not). In order to perform full consistency checking, we furthermore need an OWL DL reasoner, since most of MUSING's ontology is of species OWL DL. We note here that both OWLIM and Jena (see below) do not provide full OWL DL expressiveness. We have used Pellet v1.5.0.

Sesame Sesame (Broekstra, Kampman, & van Harmelen 2002) (http://www.openrdf.org/) implements an open source Java ontology middleware framework for storing and querying RDF(S) data. Sesame comes up with a flexible storage and inference layer (SAIL) that can be parameterized, so that, for instance different reasoners can be plugged in (e.g., OWLIM, see below) or different storage models can be chosen, varying from in-memory models, XML-DBs, or RDMSs. We decided to use Sesame, since it provides extremely fast access to RDF data and blends perfectly with the OLWIM rule engine (see next subsection). An in-memory variant of Sesame serves as the storage layer for both the OWLIM and the Jena reasoner in our hybrid architecture. We have used Sesame v1.2.7.

OWLIM OWLIM (http://www.ontotext.com/) is the current default SAIL for Sesame. The in-memory variant SwiftOWLIM (Ontotext 2007) is claimed to be the fastest RDF(S) and OWL engine (Kiryakov 2006). The rule language is mostly a variant of Datalog that supports the semantics of RDFS and OWL via axiomatic facts and entailment rules (Hayes 2004; ter Horst 2005). Entailment rules consist of an antecedant and a consequent, which are sequences of triples that are implicitly conjoined. Elements in subject, predicate, or object position are either XSD literals, URIs, or universally quantified variables. An unbound variable in the head of a rule is always regarded as an *existentially quantified* variable, leading to the introduction of an RDF blank node. We note here that this behavior is vital for domain-specific inference rules in MUSING, since it gives us the chance to introduce new individuals (and not merely only new triples, composed from already existing individuals). OWLIM essentially capture the functionality of OWL

[6] Side note: OWLIM has no means to do numerical comparison on the rule level, but only on the query level.

Lite, but has, at the same time, more expressive constructs, such as `owl:unionOf`. Domain-specific rules can be simply added to a rule file that defines an OWL semantics for OWLIM. We have used OWLIM v2.9.0.

Jena Jena (http://jena.sourceforge.net/) is a Java framework for building Semantic Web applications, originally developed by HP in their Semantic Web research program (http://www.hpl.hp.com/semweb/). Our main interest in Jena comes from the fact that it provides RDF and RDFS support, and implements an OWL semantics comparable in expressivity to OWLIM, again via the use of entailment rules (Reynolds 2006). The general rule reasoner runs in both forward and backward mode and can be parameterized through different rule files of increasing OWL complexity (name of files: micro, mini, full). It is worth noting that OWLIM proceeds in the same direction (rule sets: rdfs, owl-horst, owl-max). Jena supports SPARQL to query RDF data and provides an SWRL mode, where SWRL rules are mapped onto the more general rule format of Jena. We have used Jena v2.5.2.

OWLIM vs. Jena Both OWLIM and Jena are comparable in expressiveness if we focus ourself to OWL (rule sets full and owl-max, resp.). The application of domain-specific rules in Jena and OWLIM is achieved by simply adding those rules to the OWL rule file. Contrary to OWLIM, Jena comes up with a library of built-in primitives to do numerical comparison, arithmetics, regular expression matching, etc. (Reynolds 2007).

Both OWLIM and Jena support inequality constraints. They furthermore support existential quantification over rule variables, a very important feature that is used to generate new individuals (URIs). This is realized in both OWLIM and Jena by introducing RDF blank nodes. OWLIM achieves this via the use of unbound variables in the head of a rule, whereas Jena employs the build-in primitive `makeTemp()` in the body of a rule, whose argument (a variable) is bound to a new blank node that can be employed in the rule head later.

Whether the resulting deductive closure (the generated model) is consistent or not is done in both frameworks by querying for instances of type `owl:Nothing`, the bottom type in OWL. Instances of `owl:Nothing` are introduced by inconsistency rules, similar to the integrity constraint shown above. Bottom instances are derivable from both TBox and ABox inconsistencies through a combination of `owl:disjointWith` and either `rdfs:subClassOf` or `owl:equivalentClass`, or `owl:sameAs` and `owl:differentFrom`, resp.

Concerning runtime speed and space requirements, we must say that OWLIM outperforms Jena by a large margin.[7] Given all this information, we are now ready to present the overall hybrid reasoning architecture.

Why This Architecture?

The architecture depicted in Figure 1 is based on measurements (loading, consistency checking, querying) and on a solid inspection of the descriptive means of each formalism. To carry out realistic performance measurements, we have worked with an ontology that contains a fair amount of classes as well as individuals. At the time of writing, the MUSING ontology features a well-equipped TBox, consisting of the extended PROTON ontology, an XBRL ontology (http://www.xbrl.org) for the German accounting principles (Declerck & Krieger 2005), and the time ontology, while the ABox is for the time being only filled with a smaller number of test instances. We thus have chosen the ontology that backs up the LT-World information portal (http://www.lt-world.org/), consisting of approximately 1,000 classes and 20,000 instances. Due to space requirements, we will report on the actual numbers in the oral talk.

Pellet, OWLIM & Jena Since most of the MUSING ontology is of species OWL DL, we decided to have a pure description logic reasoner as the starting point in our architecture. This reasoner is feeded with the initial ontolgy that is going to be populated later. However, rule knowledge is not implemented in the preliminary rule language of Pellet. The only purpose of Pellet here is to make sure that the initial TBox and ABox are consistent (if this is desired).

As we said above, both OWLIM and Jena are of similar expressivity when dealing with RDF/OWL only. OWLIM, however, is so much faster on the LT-World ontology when it comes to TBox/ABox consistency checking and equational reasoning over ABox individuals. Unfortunately, OWLIM does not provide means to formulate numerical constraints and to perform simple arithmetics—this is where Jena shines.

Given this information, we decided to have both reasoners in sequence, working on the same storage model provided by Sesame. Thus OWL ABox reasoning and domain-specific rules not involving numerical constraints/arithmetic are handled solely in OWLIM, whereas rules dedicated to numbers are applied in Jena.

Since both reasoners might generate new individuals and triples not seen before by the other reasoner, the fixpoint computation stretched over OWLIM and Jena. Thus the resulting deductive closure is only reached if both reasoners do not see any new information from their counterpart. In order to let OWLIM and Jena work on the same Sesame repository, we have adapted Weijian Fang's Jena Sesame Bridge (http://sourceforge.net/projects/jenasesamemodel/) to work on the latest versions of Sesame, OWLIM, and Jena.

QDP—The Querying-Distilling-Populating Cycle It is worth noting that the result tables originating from external queries can fruitfully be employed to further populate the ontolgy with information that is not expressible via OWLIM or Jena rules. This is due to the fact that logical variables in both OWLIM and Jena rules only bind *one* individual at a time. Rule knowledge that is based on the availability of *all* individuals matching a logic variable in a specific clause can thus not directly be formulated within these formalisms.

Consider the following example. Assume, we are asking for the duration of Jürgen Schrempp's CEOship within Daimler Chrysler (DC). A query will return a table of time slices (see ontology section for an explanation), each having

[7] We note here that we have not tried Oracle 11g yet.

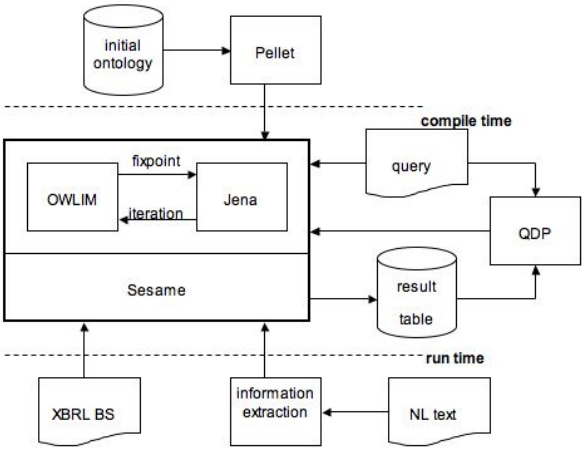


Figure 1: **The proposed system architecture.** Input to the system at the moment either comes from natural language text or from XBRL balance sheets. The initial ontology (essentially the TBox) is checked for consistency by a full description logic reasoner (Pellet: OWL DL). This ontology then is forwarded to the main resoning component (bold solid-line box) whose storage model is based on Sesame. ABox equational resoning is performed via Ontotext's OWLIM, whereas numerical constraints and arithmetic is handled solely by the Jena engine form HP. Addional information from result tables will also populate the ontology through QDP.

a different temporal extent. In order to compute the maximal interval in which Schrempp was the CEO of DC, we need to know the minimum starting time and the maximum ending time of all intervals.[8]

In order to emulate this behavior, we need to post predefined queries to the ontology (Q = querying), followed by the construction of individuals from the result tables (D = distilling), and finally by entering these new individuals to the ontology (P = populating). This again might trigger a new fixpoint iteration in OWLIM/Jena. This future work is assumed to be realized in Figure 1 by the QDP box.

Rule Examples

As we already said, rules in OWLIM and Jena will encode rule knowledge that is outside the expressive inventory of OWL. These rules basically capture knowledge that involves more than two distinct variables, numerical constraints, arithmetic, and the intrduction of new individuals (= existential quantification in the consequent of a rule). Here is an OWLIM examples that "closes" a temporal interval w.r.t. `ceoOf` relation:

```
p <rdf:type> <4d:Perdurant>
p <4d:hasTimeSlice> ts1
p <4d:hasTimeSlice> ts2    [ts1 != ts2]
ts1 <mus:ceoOf> c
ts2 <mus:ceoOf> c
ts1 <4d:hasTemporalEntity> i1
i1 <rdf:type> <time:OpenRightInterval>
ts2 <4d:hasTemporalEntity> i2
i2 <rdf:type> <time:OpenLeftInterval>
i1 <time:begins> s
i2 <time:ends> e
-------------------------------------
p <4d:hasTimeSlice> ts
ts <mus:ceoOf> c
ts <4d:hasTemporalEntity> i
i <rdf:type> <time:ClosedInterval>
i <time:begins> s
i <time:ends> e
```

Note that `ts` and `i` are brand new individuals (RDF blank nodes) that have been introduced to combine the information from both open intervals. Note further that such a rule is a heuristic rule, since there might be subintervals between `s` and `e` in which `p` is not the CEO of `c`. This rule can furthermore be generalized in that we can even quantify over the `ceoOf` relation using a predicate variable `rel`, leading to a rule that subsumes several ordinary "first-order" rules:

```
p <rdf:type> <4d:Perdurant>
...
ts1 rel c
ts2 rel c
...
-------------------------------------
p <4d:hasTimeSlice> ts
ts rel c
...
```

Both rules, however, might introduce unnecessary overhead in that they would produce not only maximal intervals, but all valid combinations. As noted before, the QDP component is a better choice to capture the desired behavior.

Let us focus on another rule that "merges" information from two perdurants that have operated under different names within different time periods (example: *Daimler Benz* and *DaimlerChrysler*). Here, we use the fact that an XBRL balance sheet of a company contains information about the former name of the company under `geninfo.company.id.name.formername` (`formerName` for short):

```
ts1 <rdf:type> <mus:Company>
ts1 <mus:hasBalanceSheet> bs
bs <mus:formerName> fn
ts2 <mus:hasName> fn
ts1 <4d:timeSliceOf> p1
ts2 <4d:timeSliceOf> p2
---------------------------
p1 <owl:sameAs> p2
```

As seen from this rule, merging here is achieved by telling the system that the associated perdurants of `ts1` and `ts2` are the same. Similar rules can be formulated using further XBRL attributes, such as `geninfo.company.id.parent.name` (example: Peugeot and Citroen are subsidiaries of PSA in France).

We finally present a rule that can further be generalized by quantifying over the predicate position of an RDF triple, perhaps completed by a type restriction of this predicate, using

[8] What is actually needed here are *aggregate* functions and a GROUP BY construct as known from SQL. Unfortunately, SPARQL does not come up with these means, so that we probably need a programming workaround here.

a `rdfs:subPropertyOf` classification scheme of succession relations. The rule basically talks about successions of management positions (here: CEO) of persons within companies. If `ts2` follows `ts1` and `ts1`'s position in `c` ends at `e`, then `ts2`'s position in `c` starts at `e`. Note that the temporal information is encoded in a brand new interval `i2` which is open to the right:

```
ts1 <rdf:type> <mus:Person>
ts1 <mus:ceoOf> c
ts1 <4d:hasTemporalEntity> i1
i1 <rdf:type> <time:Interval>
i1 <time:ends> e
ts1 <mus:precedes> ts2
-------------------------------------
ts2 <mus:ceoOf> c
ts2 <4d:hasTemporalEntity> i2
i2 <rdf:type> <time:OpenRightInterval>
i2 <time:begins> e
```

Note that the antecedent of the last rule makes use of the fact that `Interval` is the superclass of both `OpenLeftInterval` and `ClosedInterval` which are guaranteed to have a proper `ends` value.

References

Allen, J. F. 1984. A general model of action and time. *Artificial Intelligence* 23(2):123–154.

Baader, F., and Hanschke, P. 1991. A scheme for integrating concrete domains into concept languages. In *Proceedings of 12th International Joint Conference on Artificial Intelligence*, 452–457.

Baader, F., and Sattler, U. 2001. An overview of tableau algorithms for description logics. *Studia Logica* 69:5–40.

Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; and Patel-Schneider, P. 2003. *The Description Logic Handbook*. Cambridge: Cambridge University Press.

Broekstra, J.; Kampman, A.; and van Harmelen, F. 2002. Sesame: A generic archistecture for storing and querying RDF and RDF schema. In *Proceedings of the 1st International Semantic Web Conference* , 54–68. Springer.

Declerck, T., and Krieger, H.-U. 2005. Translating XBRL into description logic. An approach using Protégé, Sesame & OWL. In *Proceedings of the 9th International Conference on Business Information Systems (BIS)*.

Genesereth, M. R. 1991. Knowledge interchange format. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, 238–249.

Haarslev, V., and Möller, R. 2001. Description of the RACER system and its applications. In *Proceedings of the International Workshop in Description Logics, DL-2001*, 131–141.

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

Hobbs, J. 2004. An OWL ontology of time. http://www.isi.edu/ pan/time/owl-time-july04.txt.

Horrocks, I.; Patel-Schneider, P. F.; Boley, H.; Tabet, S.; Grosof, B.; and Dean, M. 2004. SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Submission.

Kiryakov, A. 2006. OWLIM: balancing between scalable repository and light-weight reasoner. Developer's Track, WWW2006.

Kolovski, V.; Parsia, B.; and Sirin, E. 2006. Extending $\mathcal{SHOIQ}(D)$ tableaux with DL-safe rules: First results. In *Proceedings International Workshop on Description Logic, DL-2006*.

Lutz, C. 2004. Combining interval-based temporal reasoning with general TBoxes. *Artificial Intelligence* 152(2):235–274.

Manola, F., and Miller, E. 2004. RDF primer. Technical report, W3C. 10 February.

Masolo, C.; Borgo, S.; Gangemi, A.; Guarino, N.; Oltramari, A.; and Schneider, L. 2003. WonderWeb deliverable d17. the wonderweb library of foundational ontologies, preliminary report. Technical report, ISTC-CNR.

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

Motik, B., and Sattler, U. 2006. A comparison of reasoning techniques for querying large description logic ABoxes. In *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, 227–241.

Motik, B.; Sattler, U.; and Studer, R. 2004. Query answering for OWL-DL with rules. In *Proceedings of the 3rd International Semantic Web Conference (ISWC 2004)*, 549–563.

Nebel, B., and Smolka, G. 1989. Representation and reasoning with attributive descriptions. Technical Report, IBM Germany.

O'Connor, M.; Knublauch, H.; Tu, S.; Grosof, B.; Dean, M.; Grosso, W.; and Musen, M. 2005. Supporting rule system interoperability on the semantic web with SWRL. In *Proceedings of the International Semantic Web Conference (ISWC)*.

Ontotext. 2007. SwiftOWLIM semantic repository for RDF(S) and OWL, ver. 2.9.0.

Pratt-Hartmann, I. 2005. From TimeML to $\mathcal{TPL}^*$. In *Proceedings of the Dagstuhl Seminar* Annotating, Extracting and Reasoning about Time and Events.

Prud'hommeaux, E., and Seaborne, A. 2007. SPARQL query language for RDF. Technical report, W3C. W3C Candidate Recommendation.

Pustejovsky, J.; Castaño, J.; Ingria, R.; Saurí, R.; Gaizauskas, R.; Setzer, A.; and Katz, G. 2003. TimeML: Robust specification of event and temporal expressions in text. In *Fifth International Workshop on Computational Semantics, IWCS-5*.

Racer Systems. 2005. RacerPro user's guide, version 1.9.

Reynolds, D. 2006. Jena rules tutorial. *Jena User Conference*.

Reynolds, D. 2007. Jena 2 inference support. Version 1.35.

Schmidt-Schauß, M. 1988. Subsumption in KL-ONE is Undecidable. Technical Report, Universität Kaiserslautern.

Shieber, S. M. 1986. *An Introduction to Unification-Based Approaches to Grammar*. CSLI Lecture Notes, Number 4. Stanford: Center for the Study of Language and Information.

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

Sirin, E.; Parsia, B.; Cuenca-Grau, B.; Kalyanpur, A.; and Katz, Y. 2007. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics* 5(2).

Smith, M. K.; Welty, C.; and McGuinness, D. L. 2004. OWL Web Ontology Language Guide. Technical report, W3C. 10 February.

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

Welty, C.; Fikes, R.; and Makarios, S. 2005. A reusable ontology for fluents in OWL. Research Report RC23755, IBM.