The Federated Ontology of the PAL Project
Interfacing Ontologies and Integrating Time-Dependent Data

Hans-Ulrich Krieger[1], Rifca Peters[2], Bernd Kiefer[1], Michael A. van Bekkum[3], Frank Kaptein[2], Mark A. Neerincx[3]

DFKI, the German Research Center for AI, Campus D3 2, Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany[1]

Interactive Intelligence Group, EEMCS, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands[2]

TNO, Kampweg 5, 3769 DE, Soesterberg, The Netherlands[3]

{*krieger, kiefer*}*@dfki.de*[1], {*r.m.peters, f.c.a.kaptein*}*@tudelft.nl*[2], {*michael.vanbekkum, mark.neerincx*}*@tno.nl*[3]

Keywords: knowledge representation & ontologies; Description Logics & OWL; representation of & reasoning with time-dependent data; transaction time; semantic interoperability; integration of upper & domain ontologies.

Abstract: This paper describes ongoing work carried out in the European project PAL which will support children in their diabetes self-management as well as assist health professionals and parents involved in the diabetes regimen of the child. Here, we will focus on the construction of the PAL ontology which has been assembled from several independently developed sub-ontologies and which are brought together by a set of hand-written interface axioms, expressed in OWL. We will describe in detail how the triple model of RDF has been extended towards transaction time in order to represent time-varying data. Examples of queries and rules involving temporal information will be presented as well. The approach is currently been in use in diabetes camps.

1 INTRODUCTION

In this paper, we describe ongoing work carried out in the European project PAL (Personal Assistant for a healthy Lifestyle) which will *improve child's diabetes regimen by assisting the child, health professional and parent. The* PAL *system will be composed of a social robot (NAO), its (mobile) avatar, and an extendable set of (mobile) health applications ... which all connect to a common knowledge-base and reasoning mechanism* (citation taken from the project's homepage; see `http://www.pal4u.eu`).

The **focus of this paper** lies on the construction of an integrated ontology, **PALO**, the PAL Ontology, that has been assembled from several independently-developed ontologies which are brought together by an interface specification, expressed in OWL (McGuinness and van Harmelen, 2004).[1] Within PAL, **PALO** serves as the common language which helps to interlink data, delivered from both symbolic and statistical components of the PAL system.

We will also detail how the triple data model of RDF is extended by two further arguments to incorporate temporal information in order to represent time-varying data (*transaction time*). In order to record the resulting quintuples, they can *either* be transformed into a set of semantic-preserving triples when stored in a triple repository, such as OWLIM (Kiryakov et al., 2005), by applying, e.g., W3C's N-ary relation encoding scheme (Hayes and Welty, 2006), *or* can be utilized immediately, when transferred to an *n*-tuple repository, such as *HFC* (Krieger, 2013). In PAL, we have opted for the latter case for various reasons. In this paper, we will also sneak a peek on the temporal entailment rules (Krieger, 2016) and queries that are built into the semantic repository hosting the data and which can be used to derive useful new information.

[1]The ontologies are publicly available for open research and to other institutions upon request; see `http://www.dfki.de/lt/onto/pal/`.

2 ONTOLOGIES

Overall, **PALO** consists of **eight** sub-ontologies, **seven** of which are truly independent and do not have knowledge of one another. **One** further ontology brings them together through the use of hand-written interface axioms, employing axiom constructors such as rdfs:subClassOf and owl:equivalentProperty, or by posing domain and range restrictions on certain underspecified properties. It is worth noting that across the ontologies, each property has been cross-classified as being either *synchronic*, i.e., property instances staying constant over time, or *diachronic*, i.e., changing over time (Krieger, 2010). This property characteristic can be used, amongst other things, to check the consistency of a temporal ABox or as a distinguishing mark in an entailment rule.

When we talk about an ontology here, we have to make a distinction between information from the TBox (terminological knowledge), RBox (general information about properties), and ABox (assertional

knowledge). The TBox and RBox of the PAL domain stays constant, i.e., will *not* change over time. Only relation instances from the ABox might undergo a temporal change, e.g., the weight of a child at certain times, but *not* the birthdate.

2.1 *HFC*

HFC is a bottom-up forward chainer and semantic repository implemented in Java, comparable to popular systems such as Jena and OWLIM. HFC supports RDFS and OWL reasoning à la (Hayes, 2004) and (ter Horst, 2005), but at the same time provides an expressive language for defining custom rules, involving functional and relational variables, complex tests and actions, and the replacement of triples in favour of *tuples of arbitrary length*. The query language of *HFC* implements a subset of SPARQL, but at the same time provides powerful custom M:N aggregates, not available elsewhere. In PAL, we are using *HFC* to store universal knowledge (TBox, RBox), to query time-varying data (ABox), and to reason about temporal change. This is explicated in detail in Section 3.

2.2 Upper

PAL makes use of a minimal and stripped-down upper ontology that we have originally developed for the EU projects MUSING, MONNET, and TREND-MINER (Krieger and Declerck, 2014), showing a tri-partite division of the most general class Entity, viz., upp:Abstract, upp:Happening, and upp:Physical. Most notable for PAL is the upp:Happening representation which distinguishes between atomic upp:Situations and decomposable upp:Events, using properties such as upp:startsWith, upp:continuesWith, and upp:endsWith. This allows us to encode PDL-like processes and makes it also possible to define pre- and post-conditions. upp:Happenings are upp:basedOn upp:Entitys, upp:leadsTo other upp:Entitys, and upp:involves other upp:Agents.

2.3 DIT++

The DIT++ ontology is based on the taxonomy of dialogue acts, defined by Harry Bunt and colleagues (Bunt et al., 2012). The DIT++ taxonomy is translated into a subclass hierarchy, led by the most general class dial:DialogueAct. We have taken over the *general-purpose communicative functions* and parts of the *dimension-specific communicative functions*. The former dimension involves dialogue acts, such as dial:Request, dial:Instruct, or dial:AcceptSuggestion. The latter contains communicative acts which help to maintain a dialogue, by indicating, e.g., dial:AlloFeedback or dial:Pausing. dial:DialogueActs are equipped with several important properties, such as dial:sender and dial:addressee. A dialogue act furthermore incorporates the (shallow) semantics of a natural language utterance through property dial:frame. Property dial:follows records the temporal succession of dialogue acts, whereas dial:refersTo allows to refer back to previously introduced dialogue acts (e.g., as used in indirect speech).

2.4 Time

The time ontology basically defines the classes time:DiachronicProperty and time:SynchronicProperty, making it possible to characterize OWL properties (via rdf:type) as being able to undergo a temporal change or not (see Section 2), for instance

dom:birthdate rdf:type time:SynchronicProperty
dom:weight rdf:type time:DiachronicProperty

We have furthermore defined the property time:assign to implement the concept of an imperative, programming language variable that can change over time and whose time series needs to be recorded. Such functionality is used in PAL in the dialogue processing module (see Section 4.1).

2.5 Logic

The representation of transaction time in Section 3 needs to talk about the *truth* ($= \top$) and *falsity* ($\bot$) of statements. For this, we make use of a logic ontology which includes even more general *polarity* values, such as *don't know* (?) and *error* (!), arranged in a class subsumption hierarchy: $! \sqsubseteq \{\top, \bot\} \sqsubseteq ?$.

2.6 Domain

The domain ontology defines concepts and relation which are relevant to the PAL domain, e.g., dom:Activity (playing a game, cooking, making a diary entry), dom:Actor (child, family members, health professionals), emotional dom:Mood, or (learning) dom:Goals which progress over time (see Section 2.8). As the child (and its diabetes' history) is at the heart of the PAL project, dom:Child is consequently equipped with a large number of properties, dealing with family relationships, serious issues (hypoglycemia symptoms), hobbies, activities, or lab values. dom:LabValue bundles datatype properties relevant for the initial anamnesis and the diabetes use case, such as dom:bmi (body mass index), dom:height, or dom:bsl (blood sugar level). It is worth noting that such datatype properties usually map to custom XSD datatypes, designed for PAL (see Section 2.10).

2.7 Semantics

The shallow semantic representation in PAL is loosely build on *thematic* relations or roles (Fillmore,

1977), leading to general *verb* frames and including named arguments such as sem:agent, sem:patient, sem:theme, or sem:manner which can be found in frameworks, such as VerbNet, VerbOcean, or FrameNet (Ruppenhofer et al., 2006). These properties are defined on the very general class sem:Frame and are domain-restricted by very general classes; for instance, sem:agent and sem:patient map to the underspecified class sem:Actor. These general *docking* classes will later be interfaced with more specific classes from other sub-ontologies by means of interface axioms (Section 2.9). Even though the semantic representation is almost flat, additional roles such as sem:purpose (typed to sem:Frame) allow us to build up nested structures, say for a sentence like *OK, you will be asking* (frame: sem:AssigningRole) in a natural language quiz scenario between robot and child.

2.8 Goal

The goal ontology formalizes diabetes self-management progression and is based on the Dutch *Diabetes "weet & doe" doelen* (know & do goals) as formulated by the EADV (`http://www.eadv.nl/`). These recommendations structure knowledge and skills supposed to be obtained by the child from onset to adolescence in order to gradually increase autonomy. Thus, goals are attuned to age ranges and are divided into important topics, such as *nutrition* and *insulin*. These goals are translated into subclasses of goal:KnowledgeGoal and goal:SkillGoal, led by the superclass goal:T1DMGoal. One aim of the PAL system is to support self-management progression, by offering educational content and activities. The PAL system objectives that contribute to diabetes learning goals are defined as subclasses of goal:SupportingObjectives. Multilingual labels for Dutch, Italian, and English have been added to the goal classes as they were used in the dialogue. Properties, such as goal:hasLevel (the suggested age range) and goal:hasProgress (capturing percentage of completion) are defined on the general goal class goal:Goal. Dependencies between goals are captured via property goal:requiresAsClass which directly operates on class objects (see Section 4.2).

2.9 Pal

The PAL ontology first of all imports the previously introduced sub-ontologies, but also defines interface axioms in order to properly integrate the distributed information. This includes, e.g., restricting the domain and range of (possibly underspecified) properties or identifying (subsuming) classes and properties across ontologies. For example:

$$\text{dom:Actor} \equiv \text{upp:Agent} \equiv \text{dial:Agent} \equiv \text{sem:Actor}$$
$$\text{dom:Goal} \equiv \text{goal:Goal} \sqsubseteq \text{upp:Event}$$
$$\text{goal:contributesTo} \sqsubseteq \text{upp:leadsTo}$$
$$\forall\, \text{dial:frame} \,.\, \text{sem:Frame}$$

The *first* axiom identifies the important actor/agent classes that can be found in the various ontologies. The *second* statement makes goal:Goal (and dom:Goal) a subclass of the very general class upp:Event from the upper ontology (see Section 2.2). As a consequence, properties, such as upp:startsWith or upp:continuesWith, defined on upp:Event become available in instances of goal:Goal (*goals behave like events, occupying time*). The *third* declaration defines goal:contributesTo as a subproperty of the general property upp:leadsTo and constraints the relation signature from $\langle$upp:Happening, upp:Entity$\rangle$ to $\langle$goal:SupportingObjective, goal:T1DMLearningGoal$\rangle$. The *fourth* restriction links the underspecified dialogue act property dial:frame to shallow semantic frames (see Sections 2.7 and 4 for an example).

2.10 XSD Datatypes

Some of the datatype properties from the domain ontology utilize custom XSD types. For instance:

- *body mass index* dom:bmi, measured in xsd:kg_m2
- *blood sugar level* dom:bsl, *either* measured in xsd:mmol_L *or* xsd:mg_dL
- *diastolic blood pressure* dom:dbp, measured in xsd:mmHg

3 HANDLING TIME

This section shed some light on the representation of time-varying data in PAL and the underlying model, viz., *transaction time*. We will also look into how temporal information is utilized in queries and rules.

3.1 Metric Linear Time

In the following, we assume that the temporal measuring system is based on a *metric linear time*, so that we can compare starting/ending points, using operators, such as $<$ or $\leq$, or pick out input arguments in aggregates, using *min* or *max*. We furthermore require that time is *discrete* and represented by natural numbers. The implementation of *HFC* employs 8-byte long integers (XSD datatype `long`) to encode *milli* or even *nano* seconds w.r.t. a fixed starting point (Unix Epoch time, starting from 1 January 1970, 00:00:00). As a consequence, given a time point t, the next smallest or successor time point would then be $t+1$.

3.2 Transaction Time

Transaction time (Snodgrass, 2000) makes use of temporal intervals in order to represent the time during which a fact is stored in the database, even though

the ending time must not be known in advance. This is indicated by the wildcard **?** in the database table below which will later be *overwritten* by the concrete ending time.

We *deviate* here from the interval view by specifying both the starting time when an ABox statement is entered to the ontology, and, via a *separate* statement, the ending time when the statement is *invalidated*. For this, we exploit the polarity values $\top$ and $\bot$ from the logic ontology that we have already introduced in Section 2.5. This idea is shown below for a binary relation P. We write `P(c,d,b,e)` to denote row `<c,d,b,e>` in the database table `P` for relation P.

TIME	DATABASE VIEW	ONTOLOGY VIEW
⋮	⋮	⋮
t_1	add: `P(c,d,`t_1`,?)`	add: $\top$P(c,d)@t_1
⋮	⋮	⋮
t_2	**overwrite**: `P(c,d,`t_1`,`t_2`)`	——
t_2+1	——	**add**: $\bot$P(c,d)@t_2+1
⋮	⋮	⋮

As we see from this picture, the invalidation in the ontology happens at t_2+1, whereas $[t_1,t_2]$ specifies the transaction time in the database. Clearly, the same transaction time interval for P(c,d) in the ontology can be derived from the two statements $\top$P(c,d)@t_1 and $\bot$P(c,d)@t_2+1, assuming that there does *not* exist a $\bot$P(c,d)@t, such that $t_1 < t \leq t_2$ (we can effectively query for this by employing the ValidInBetween test; see Section 3.4 for its use in a rule).

Extending ontologies by transaction time the way we proceed here gives us a means to easily encode *time series data*, i.e., allows us to record the *history* of data that changes over time, e.g., the blood sugar level of a child (see Section 2.4). The formal foundations for extending the triple model with transaction time can be found in (Krieger, 2016).

Given polarity value $\pi = \{\top, \bot\}$, the above statements

πP(c, d)@t

are written in *HFC* as *quintuples*, i.e.,

π c P d t

As we opt for a *uniform* representation, *axiomatic triples* from the TBox and RBox of an ontology need to be extended by two further arguments; for instance,

owl:sameAs rdf:type owl:TransitiveProperty

becomes quintuple[2]

true sameAs type TransitiveProperty ”0”^^long

We read the above statement as *being true* ($\top$ = logic:true) *from the beginning of time* (long int 0 = ”0”^^xsd:long).

[2]We sometimes omit namespaces here in order to make sure that a quintuple fits into a single paper line.

Information uploaded into *HFC* is also *backed up* by an external file. However, entailed information, obtained through successive rule applications (see Section 3.4) is not stored at all, as it can be restored through the same rules again. As a consequence, *wrongly-entered* information at time t can either be deleted directly in case no rule application has taken place since, or is deleted together with derived information from a later time $t' > t$ (like a DB rollback), followed by an application of the rules.

3.3 Queries and a Use Case

The query language of *HFC* can be seen as an extension of a subset of SPARQL towards general *n*-tuples. Consider the following quintuple excerpt from the ABox for *Lisa* who has undergone anamnesis at time 5544 and further lab values taken at 5577:

```
logic:true lisa rdf:type dom:Child ”5544”^^xsd:long
true lisa dom:hasLabValue lv22 ”5544”^^xsd:long
true lv22 dom:height ”133”^^xsd:cm ”5544”^^xsd:long
true lv22 dom:weight ”28.2”^^xsd:kg ”5544”^^xsd:long
true lv22 dom:bsl ”9.0”^^xsd:mmol_L ”5544”^^xsd:long
.....
true lisa dom:hasLabValue lv33 ”5577”^^xsd:long
true lv33 dom:weight ”28.6”^^xsd:kg ”5577”^^xsd:long
true lv33 dom:bsl ”165.6”^^xsd:mg_dL ”5577”^^xsd:long
.....
```

What this example shows is that the blood sugar level dom:bsl for *Lisa* was measured using different *units* at different times (cf. Section 2.10). Given that all possible lab values will *not* be taken every time a medical examination takes place, we would nevertheless like to know the *latest* value for each individual property; for instance in our case, that *Lisa* is 133 cm tall (time: 5544), weights 28.6 kg (time: 5577), and has been measured with a blood sugar level of 165.6 mg/dL also at 5577. This information can be obtained through the following quintuple-based query which utilizes the complex aggregate GetLatestValues:

```
SELECT ?prop ?val ?t
WHERE logic:true lisa dom:hasLabValue ?labvalue ?t &
      logic:true ?labvalue ?prop ?val ?t
AGGREGATE ?measurement ?result ?time =
  GetLatestValues ?prop ?val ?t ?t
```

The meaning of SELECT and WHERE does not differ from SPARQL, except that quintuples are involved instead of triples. AGGREGATE specifies an aggregate with four input and three output arguments which sorts the result table obtained from SELECT-WHERE and headed by ⟨?prop,?val,?t⟩ according to the last fourth element ?t. It then takes the newest values ⟨?val,?t⟩ (argument 2 and 3) for each property ?prop (argument 1) and finally returns the following table:

?measurement	?result	?time
dom:height	”133”^^xsd:cm	”5544”^^xsd:long
⋮	⋮	⋮
dom:weight	”28.6”^^xsd:kg	”5577”^^xsd:long
dom:bsl	”165.6”^^xsd:mg_dL	”5577”^^xsd:long
⋮	⋮	⋮

3.4 Rules

As we have shown in (Krieger, 2016), the entailment rules for RDFS (Hayes, 2004) and OWL (ter Horst, 2005) can be extended naturally towards a treatment of time-varying data which mimics transaction time (Snodgrass, 2000). Here, we will present two such entailment rules which will derive new information for the PAL domain. The first one deal with properties and subproperties (see Section 2.9 for two such properties). The original rule rdfs7x from (ter Horst, 2005) is (we separate the *if-then* parts by writing ->):

```
?p rdfs:subPropertyOf ?q
?v ?p ?w
->
?w ?q ?w
```

This is exactly the syntax used in *HFC* for writing rules. The transaction time extension using quintuples is quite natural:

```
logic:true ?p rdfs:subPropertyOf ?q ”0”^^xsd:long
logic:true ?v ?p ?w ?t
->
logic:true ?w ?q ?w ?t
```

As we see, the underlined parts of the three clauses correspond one-to-one to the original rule and all statements are valid (first argument: logic:true). Instantiations of the first clause will be RBox axioms which will not change over time, thus we assign time 0 here, whereas changing time in the other two clauses is addressed by a coinciding logic variable ?t. The next rule does *not* have a counterpart in neither (Hayes, 2004) nor (ter Horst, 2005). It addresses a *functional property* P defined on x whose value y at time t_1 is specified differently at a *later* time t_2 by z, *without* invalidating y before:

```
true ?p rdf:type owl:FunctionalProperty ”0”^^long
true ?x ?p ?y ?t1
true ?x ?p ?z ?t2
->
error ?x ?p ?y ?t2
error ?x ?p ?z ?t2
@test
?y != ?z
?t1 < ?t2
ValidInBetween ?x ?p ?y ?t1 ?t2
```

This rule derives that P(x,y)@t_2 as well as P(x,z)@t_2 is an *inconsistent* (but *not* a *false*) statement in case P(x,y) does not get invalidated at $t < t_2$: $\perp$P(x,y)@t. Whether this is the case is checked by ValidInBetween as explained before in Section 3.2. If the test succeeds, we mark the inconsistency through the use of the error modality ! (see Section 2.5) on the RHS.

4 ONTOLOGY IN USE

We have already presented an *use case* involving the ontology in Section 3.3, where a health professional is interested in obtaining the most recent lab values for a specific child. Here, we will look into two further examples.

4.1 Use Case 2: Dialogue Processing

The natural language dialogue engine in PAL utilizes sets of reactive *if-then*-like rules for the various health applications (e.g., diabetes diary, educational quizzes, sorting games). Simplified, the rules match against *general* as well as *specific dialogue situations* (= dialogue acts enriched by semantics and other information; see Sections 2.3 and 2.7) and generate continuations, describing how the dialogue proceeds. Both the matching information as well as the derived new information is grounded in time, represented by the transaction time model presented above, and stored in *HFC*. Even though the transaction time model and the ontology schema lead to a high abstraction level, *HFC* queries (Section 3.3) and rules (Section 3.4) would still be too *talkative* to be of easy use. Thus the reactive dialogue rules abstract away from things that need to be repeated over and over again (e.g., properties, such as dial:sender or dial:addressee; property chains; time). Here is an example of such a rule, a specialization of a general *answer*:

```
if (myLastDA <= @Request(Top)
    && lastDA < @Answer(Top)) {
  if (lastDA <= @Confirm(Top))
    lastDA.dialogueAct = AcceptRequest;
  else
    lastDA.dialogueAct = RejectRequest;
}
```

If the *sender*'s last *dialogue act* myLastDA is at least as specific as dial:Request (see Section 2.3) and we are given a confirmation by the addressee (stored in lastDA), the rule will assign a more specific dialogue act, viz., AcceptRequest to the field dialogueAct of variable lastDA; otherwise, RejectRequest is assigned. Even though lastDA and myLastDA look like imperative variables, they are implemented with the help of time:assign to record time series data (see Section 2.4). Furthermore, complex conditions, such as the subsumption tests above are compiled into complex SPARQL-like ASK queries.

4.2 Use Case 3: Goal Progression

The goal ontology is used to inform the child, its parents, and the healthcare professionals on the current status of self-management, but also to direct the PAL system to provide suitable content and activities. Imagine a child *Henk*, recently diagnosed with diabetes and started treatment, including self-management educational goals. *Henk* already learned that insulin intake is needed, thus goal:InsulinIntake is achieved and is given progress value 1.0. Note how the domain and goal sub-ontologies interacts (below, we omit the first argument logic:true and the transaction time argument of the quintuple in lack of space):

henk dom:hasTreatment henks_treatment
henks_treatment dom:hasGoal insulinIntake_henk
insulinIntake_henk goal:hasProgress "1.0"^^xsd:float

Henk's first selected objective is to learn to inject insulin. This requires knowledge on the location for injection and skills to prepare the insulin pen. Upon selection of goal:InsulinInjection, the progress value of this goal and its pre-conditions goal:PreparePen and goal:InsulinLocation is set to 0.0, as for related subclasses of goal:SupportingObjectives:

InsulinInjection goal:requiresAsClass PreparePen
InsulinInjection goal:requiresAsClass InsulinLocation
InsulinLocation goal:requiresAsClass AnswerI1
insulinLocation_henk goal:hasProgress "0.0"^^xsd:float
answerI1_henk goal:hasProgress "0.0"^^xsd:float
preparePen_henk goal:hasProgress "0.0"^^xsd:float

While playing a quiz, the PAL system keeps track of the scores and for each correct answer, the corresponding progress value is updated at a later time:

answerI1_henk goal:hasProgress "0.2"^^xsd:float

After correctly answering all related quiz question, the goal is achieved and all connected learning goals advance progression. Since goal:InsulinLocation has no other pre-condition, progress is updated to 1.0. As goal:InsulinInjection also specifies goal:PreparePen as a further pre-condition via property goal:requiresAsClass (see above), it is therefore progressing to 0.5 (both pre-conditions are equally important):

answerI1_henk goal:hasProgress "1.0"^^xsd:float
insulinLocation_henk hasProgress "1.0"^^xsd:float
preparePen_henk goal:hasProgress "0.0"^^xsd:float
insulinInjection_henk goal:hasProgress "0.5"^^xsd:float

ACKNOWLEDGEMENTS

The research described in this paper has been partially financed by the European project PAL (Personal Assistant for healthy Lifestyle) under Grant agreement no. 643783-RIA Horizon 2020.

Thanks to Antoine Cully and Maxime Petit (Imperial College London) for the preparation of the PAL *Data Definition*. We would like to thank the reviewers for their suggestions.

REFERENCES

Bunt, H., Alexandersson, J., Choe, J.-W., Fang, A. C., Hasida, K., Petukhova, V., Popescu-Belis, A., and Traum, D. (2012). Iso 24617-2: A semantically-based standard for dialogue annotation. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, pages 430–437.

Fillmore, C. J. (1977). The case for case reopened. In Cole, P. and Saddock, J. M., editors, *Grammatical Relations. Syntax & Semantics 8*, pages 59–81. Academic Press.

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

Hayes, P. and Welty, C. (2006). Defining N-ary relations on the Semantic Web. Technical report, W3C.

Kiryakov, A., Ognyanov, D., and Manov, D. (2005). OWLIM – a pragmatic semantic repository for OWL. In *Proceedings of the International Workshop on Scalable Semantic Web Knowledge Base Systems*, pages 182–192.

Krieger, H.-U. (2010). A general methodology for equipping ontologies with time. In *Proceedings LREC 2010*.

Krieger, H.-U. (2013). An efficient implementation of equivalence relations in OWL via rule and query rewriting. In *Proceedings of the 7th IEEE International Conference on Semantic Computing (ICSC)*, pages 260–263.

Krieger, H.-U. (2016). Integrating graded knowledge and temporal change in a modal fragment of OWL. In van den Herik, J. and Filipe, J., editors, *Agents and Artificial Intelligence*, Lecture Notes in Computer Science. Springer, Berlin. Revised selected papers from the 8th International Conference, ICAART 2016.

Krieger, H.-U. and Declerck, T. (2014). TMO—the federated ontology of the TrendMiner project. In *Proceedings of the 9th edition of the Language Resources and Evaluation Conference (LREC)*.

McGuinness, D. L. and van Harmelen, F. (2004). OWL Web Ontology Language Overview. Technical report, W3C.

Ruppenhofer, J., Ellsworth, M., Petruck, M. R., Johnson, C. R., and Scheffczyk, J. (2006). FrameNet II: Extended theory and practice. Technical report, International Computer Science Institute (ICSI), University of California, Berkley.

Snodgrass, R. T. (2000). *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann, San Francisco, CA.

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