A Method for Representing and Querying Temporal Information in OWL

Martin J. O'Connor and Amar K. Das

Stanford Center for Biomedical Informatics Research
Stanford, CA 94305, U.S.A.

Abstract. Ontologies are becoming a core technology for supporting the sharing, integration, and management of information sources in Semantic Web applications. As critical as ontologies have become, ontology languages such as OWL typically provide minimal support for modeling the complex temporal information often contained in these sources. As a result, ontologies often cannot fully express the temporal knowledge needed by many applications, forcing users and developers to develop *ad hoc* solutions. In this paper, we present a methodology and a set of tools for representing and querying temporal information in OWL ontologies. The approach uses a lightweight temporal model to encode the temporal dimension of data. It also uses the OWL-based Semantic Web Rule Language (SWRL) and the SWRL-based OWL query language SQWRL to reason with and query the temporal information represented using our model.

1 Introduction

The Semantic Web effort [5] aims to provide languages and tools that specify explicit semantic meaning for data and knowledge shared among knowledge-based applications. In particular, the Ontology Web Language (OWL; [23]) and its associated Semantic Web Rule Language (SWRL; [15]) provide a powerful standardized approach for representing information and reasoning with it. Despite the power of these technologies, they have very limited support for temporal information modeling. OWL, for example, provides no temporal support beyond allowing data values to be typed as basic XML Schema dates, times or durations [38]. SWRL includes operators for manipulating these values, but its operators work at a very low level. There are no standard high-level mechanisms to consistently represent and reason with temporal information in OWL.

Because the temporal dimension is central in many information sources, these shortcomings have significant consequences. Primarily, they restrict the complexity of temporal information that can be represented in application ontologies. In addition, they reduce the possibilities for automated temporal validation of the temporal information that they do have. Crucially, these restrictions also limit the temporal expressivity of deductive rules and queries that can be formulated over ontology-encoded data. Formulating these rules and queries thus requires custom solutions using technologies that may not leverage the formal knowledge representation techniques

A. Fred, J. Filipe, and H. Gamboa (Eds.): BIOSTEC 2010, CCIS 127, pp. 97–110, 2011.

provided by ontologies. There is a pressing need for solutions that provide robust knowledge-level mechanisms for representing and reasoning with temporal information in OWL ontologies.

2 Background

Historically, the importance of time in many applications has driven the development of custom temporal management solutions. In particular, the centrality of the temporal dimension in biomedical data drove early research in the area. One of the first biomedical systems to address the problem was the Time Oriented Database (TOD; [37]). TOD had a three-dimensional view of clinical data, with time represented explicitly as one of the dimensions. Data relating to individual patient visits, for example, were indexed by patient identifier, clinical parameter type, and visit time. TOD supported a set of basic temporal queries that allowed extraction of data values following certain temporal patterns. The Arden Syntax [16] also supports a basic instant-based temporal representation.

Both TOD and the Arden Syntax model time by associating an *instant timestamp* with particular records. An instant timestamp permits a range of simple temporal questions, such as "Did the patient suffer from shortness of breath before the visit?" or "Did the patient receive Ibuprofen last week?" However, associating an *interval timestamp* with data enables more complex queries. Interval timestamps are composed of a start timestamp and a stop timestamp. Later systems used this type of temporal representation. These systems were typically built to operate with relational database systems and exploited the considerable amount of research on temporal database systems in the 1990s.

This research aimed to address the shortcomings of relational databases for representing temporal information. While relational databases can readily store time values, the relational model provides poor support for storing complex temporal information. A simple instant timestamp is all that the relational model provides, and there is no consistent mechanism for associating the timestamp with non-temporal data. For example, if a database row contains some temporal information, there is no indication as to the relationship between it and the non-temporal data in the row. Does the timestamp refer to the time when the information was recorded, or to when it was known? Other shortcomings include no standard way to indicate a timestamp's granularity. As a result, the relational model provides very limited capabilities for temporal querying.

More than a dozen formal extensions to the relational data model were proposed to address the problem. These approaches ultimately led to the development of a consensus query language, called TSQL2 [30]. TSQL2 supports temporal and non-temporal tables. It also provides a temporal relational algebra that can undertake temporal selection of data, temporal joins based on temporal intersection, and temporal catenation of interval time stamps. The TSQL2 query language is compatible with standard SQL, since it is a strict super set of it. Efforts were made to introduce some TSQL2 temporal features into the SQL3 standard [31], but these efforts did not succeed. No complete implementations of TSQL2 were produced, but several temporal query systems were written using its core features. TSQL2-influenced systems include Chronus [1994] and

its later evolution, Chronus II [24]. Like the earlier TOD system [37], both were developed to meet the temporal query requirements of biomedical applications.

Experience with these systems illustrated that the entire TSQL2 specification is not necessary for developing a powerful temporal query language. A few simple language extensions can provide a large increase in temporal expressivity. The most important lesson learned is that a principled temporal model is key to developing the extensions. This model must enforce a consistent representation of all temporal information in a system. One of the important results of the TSQL2 standardization efforts was the convergence on the *valid-time temporal model* [30]. While numerous models were proposed to represent temporal information in relational databases and in other types of information systems, this model was selected because it coupled simplicity with considerable expressivity.

In the valid-time model, a piece of information—which is often referred to as a *fact*-can be associated with instants or intervals denoting the times that the fact is held to be true. Facts have a value and one or more valid-times. Conceptually, this representation means that every temporal fact is held to be true during the time(s) associated with it. No conclusions can be made about the fact for periods outside of its valid-time. The valid-time model provides a mechanism for standardizing the representation of time-stamped data. When this model is used in a relational system, temporal information is typically attached to all tuples in a temporal table. This approach effectively adds a third dimension to two-dimensional relational tables. The valid-time model is not restricted to use in relational systems, however, and can be used in any information system that requires a consistent representation of temporal information.

The valid-time temporal model has been used to model time in the XML domain, primarily by developing extensions to XPath [2, 39]. Extensions to the XQuery language have also been developed to facilitate temporal querying of XML data described using the valid-time model [11].

The importance of time was recognized early in the Semantic Web effort [7]. A variety of approaches have been proposed to represent temporal information in RDF [22] and OWL [23], the two primary Semantic Web languages. The valid-time model predominates in these approaches. One of the earliest RDF systems was BUSTER [35], an information retrieval system that used a temporal reasoning framework to help answer time-oriented questions. Several extensions to the RDF model have been proposed to facilitate representing temporal information in RDF ontologies [12, 28]. Recent systems have described RDF-based temporal query languages that use these extensions. These systems include T-SPARQL [34], which was developed as an extension to the SPARQL RDF query language, and TOQL [3], a SQL-like temporal query language.

A variety of OWL-based temporal systems have also been developed. Unfortunately, adding a temporal dimension to OWL is not straightforward. OWL's logic-based formalism makes it difficult to model dynamically changing information [18]. Also, OWL does not provide any temporal constructs. As with the relational model, a simple instant timestamp representation is all that it supports. Because OWL supports only binary predicates, relations cannot be directly equipped with a temporal argument. Rather than extending OWL's logical model, researchers have attempted to support temporal representations on top of OWL. For example, OWL-Time [14]

proposes an ontology that provides rich descriptions of temporal instants, intervals, durations, and calendar terms. However, this representation is not lightweight and concerns itself with descriptions of individual data elements, rather than building a temporal model to consistently describe all temporal information in a system.

When developing a temporal model on top of OWL, two approaches are common: *reification* and *fluents*. Reification involves introducing a new object plus a relation to associate an entity with its temporal extent or valid-time. A related approach, known as *fluents* [36], defines a time slice to encode the temporal dimension occupied by an entity. Entities are then connected through their time slices, rather than through source ontology relations. Both approaches can be modeled at the user level without modifications to OWL itself. A disadvantage is that the enforcement of temporal semantics is outside OWL, and only limited OWL reasoning can be used by the temporal reasoning processes. Both approaches also require some rewriting of a source ontology to model the temporal dimension, though this process can be relatively straightforward with reification. An additional disadvantage is that these approaches introduce new relations and objects to model the temporal information associated with an entity. As a result, querying temporal information using these approaches can be cumbersome.

Another approach that does not modify OWL is *versioning* [4]. Here, when an ontology is modified, a new version is created to represent the temporal evolution of the ontology. Most implementations adopt a variety of optimization strategies to ensure that entire copies of the ontology are not generated for each new version. However, irrespective of the optimizations adopted, versioning suffers from significant disadvantages. The primary one is that querying for events occurring during particular time intervals can be computationally expensive. Significant information redundancy is also typical of this approach.

A large amount of research has aimed at extending the description logic model underlying OWL to incorporate time [17, 18, 20]. These approaches involve defining new OWL operators and associated semantics. A variety of temporal description logics have been proposed, though there is still no agreement on a standard approach. There has been comparatively little work related to developing query languages for temporal description logics, with very few systems described in the literature [10].

The systems described here demonstrate the variety of ways in which a temporal model can be represented in OWL. Each one involves a variety of tradeoffs. Description logic-based solutions are the most theoretically attractive, because they can fully exploit OWL reasoning functionality. However, they require modifications to OWL itself. User-level models do not involve modification to OWL, but the enforcement of temporal semantics can thus not be handled by OWL reasoners. However, user-level approaches are attractive because they can be readily implemented. User-level models also involve trade-offs. Many have verbose representations of temporal information, requiring the introduction of several new objects to represent each temporal entity. The models also vary in the amount of ontology rewriting that is necessary. Some require an entire ontology redesign, while others can easily be incorporated with existing ontologies.

Irrespective of the approach adopted, a common shortcoming of existing models is their poor support for temporal querying. Unlike efforts in the relational database field that focused on developing expressive temporal query languages, OWL-based

approaches have not emphasized ease of querying. With many user-level OWL models, for example, concise query expression can be difficult because of complex underlying model representations. The development of practical query languages for temporal description logics is an open research challenge. More advanced querying functionalities, such as grouping and aggregation, are also missing from these systems, further limiting expressivity.

3 Temporal Model

We have developed a valid-time temporal model in OWL. The model was encoded at the user-level and adopts a reification-based temporal representation mechanism. We chose this approach because it most easily meets the goal of compatibility with existing OWL-based tools. The resulting model can also be easily shared and used in third-party applications. As this paper shows, this model also provides a foundation for a simple, concise, yet expressive temporal query language. We built it by extending an earlier temporal valid-time model [25] and simplified it so that it could more easily be integrated with existing ontologies. This enhanced model was designed to be lightweight, allowing it to be layered on existing OWL ontologies without requiring that they be significantly rewritten. This model concerns itself with time only and provides a simple approach to adding a temporal dimension to existing entities in domain ontologies.

We developed an ontology in OWL to encode this valid-time temporal model [32, 33]. This paper henceforth uses the prefix `temporal` for entities defined in this ontology. The core class that models an entity that can extend over time is represented by an OWL class called `temporal:Fact`. This class is associated with a property called `temporal:hasValidTimes` that holds the time(s) during which the associated information is held to be true. Values of this property are modeled by a class called `temporal:ValidTime`, which has the subclasses `temporal:ValidInstant` and `temporal:ValidInterval`. These subclasses represent instants and intervals, respectively. `Temporal:ValidInstant` is associated with the property `temporal:hasTime`, and `temporal:ValidInterval` is associated with the properties `temporal:hasBeginning` and `temporal:hasFinish`. These three properties are of XML Schema type `xsd:DateTime`. Intervals and instances also have granularities associated with them. This association is modeled by the property `temporal:hasGranularity`, with a range class called `temporal:Granularity`. Specific granularities, such as days and minutes, are modeled as instances of this class.

One possible use of the valid instant and interval classes is to take an existing OWL class and add a user-defined property with a range of one of these two classes to it. The choice of class depends on whether one wishes to model an activity that occurs at a single instant in time or one that takes place over an interval of time. Also, if the activity occurs only once, the association will be represented as an OWL functional property, whereas an activity that may repeat can use a non-functional property. For example, consider the case where a user wishes to add a temporal dimension to a blood pressure measurement that is described using a class called `BloodPressure-Measurement`, which has properties for both the systolic and diastolic values. Blood pressures are typically recorded as instantaneous measurements, so the valid instant

class would be the appropriate property range choice here. By using the valid instant class as the range of a user-defined property associated with the measurement class, all instances of `BloodPressureMeasurement` can now use the `temporal:hasTime` and `temporal:hasGranularity` properties associated with the instant. This will allow them to consistently record temporal information associated with the measurement. Similarly, if a user wishes to work with prescriptions using an existing class called `Prescriptions`, they might choose the valid interval class as the range of a user-defined property associated with the class.

A more useful modeling approach is to directly use the `temporal:Fact` class to represent temporal entities. This class can be made the superclass of an existing OWL class in need of a temporal dimension, thus asserting that instances of that class have a temporal extent. For example, if an investigator wishes to take the blood pressure measurements class cited above and model it as a temporal fact, they can simply make the class a subclass of `temporal:Fact`. Instances of this class will now be able to use the `temporal:hasValidTimes` property to store their valid instants as instances of the `temporal:ValidInstant` class. Similarly, the earlier prescriptions class can be modeled as a temporal entity by making it a subclass of the temporal fact class and using `temporal:ValidInterval` to store the temporal intervals associated with it. The granularities of those instants or intervals can also be modeled with the `temporal:hasGranularity` property associated with the `temporal:ValidTime` superclass.

Representing temporal entities as subclasses of the `temporal:Fact` class can clarify the distinction between the temporal and non temporal entities in an ontology. This temporal representation can also coexist with any existing temporal representations in the ontology, and so does not necessitate modifying the temporal component of existing entities. In most cases, existing temporal information will need to be mapped from the source entities to conform to the format encoded by valid-time instants or intervals. This mapping may be non trivial in some cases, but will ensure a consistent representation of temporal information.

4 Temporal Querying

Once all temporal information is represented consistently in an ontology, it can then be manipulated using reusable methods. While OWL itself has no temporal operators for manipulating time values, its associated rule language SWRL [15] provides a small set. However, the operators in this set are very basic, and provide simple instant-based comparisons only.

4.1 Basic Temporal Queries: Allen's Operators

SWRL provides a mechanism for creating user-defined libraries of custom methods—called *built-ins*—and using them in rules. We have used this mechanism to define a library of methods that implement Allen's [1] interval-based temporal operators. Our library provides built-ins implementing the entire set of the Allen operators. It also supports operations on basic XML Schema temporal types, such as `xsd:date`, `xsd:dateTime`, and `xsd:duration`. Operators to perform granularity conversion and duration calculations at varying granularities are also provided.

The following rule illustrates the use of a built-in defined by this library called `temporal:before`, which can be used to see if one valid time is before another. This rule classifies patients as trial-eligible if they completed any DDI drug therapy before 1999. In this rule, a patient has a property called `hasTreatment` which has a range class that is a subclass of the `temporal:Fact` class and holds a list of valid-time intervals for each treatment.

```
Patient(?p) ^ hasTreatment(?p, ?tr) ^ hasDrug(?tr, DDI) ^
temporal:hasValidTime(?tr, ?trVT) ^ temporal:hasTime(?trVT, ?t)
        temporal:before(?t, "1999") → TrialEligible(?p)
```

The library also has a native understanding of the valid-time temporal model and supports an array of temporal operations on intervals defined using the classes in our model. It can thus be used to directly reason about valid time instants and intervals. As a result, basic XSD temporal data values do not have to be extracted from valid time objects. The previous rule can thus be shortened to:

```
Patient(?p) ^ hasTreatment(?p, ?t) ^ hasDrug(?t, DDI) ^
temporal:hasValidTime(?t, ?tVT) ^ temporal:before(?tVT, "1999")
                        → TrialEligible(?p)
```

If temporal facts are supported by built-ins, even more concise rules can be written. For example, with this support, a more complex rule indicating that patients are trial-eligible if they had drugs prescribed during the first three months of 2008 that were taken for longer than one week, can be written:

```
Patient(?p) ^ hasTreatment(?p, ?t) ^ hasDrug(?t, ?drug) ^
        temporal:overlaps(?t, "2008-1", "2008-3") ^
temporal:duration(?d, ?t, temporal:weeks) ^ swrlb:greaterThan(?d, 1)
                        → TrialEligible(?p)
```

Here, the `temporal:overlaps` built-in is supplied with a temporal fact and two literal date values. Internally, it extracts the intervals associated with the temporal fact `t`, constructs an interval from the two supplied dates, and then performs the temporal comparison of the intervals.

The temporal built-ins in our library can take any combination of temporal facts, valid-time instants, valid-time intervals, or XSD date or datetime literal values. As this paper will show, the ability of built-ins to directly reason with objects defined by the temporal model can significantly reduce the number of clauses required to express temporal criteria, resulting in considerably more concise rules. It can also free users from concerns about the low-level representation details of the temporal model.

In addition to being able to write temporal rules, the ability to write temporal queries on an ontology is also desirable. A SWRL-based query language called the Semantic Query-Enhanced Web Rule Language [26] provides such support. Using built-ins, SQWRL defines a set of SQL-like query operators that that can be used to construct retrieval specifications for information stored in an OWL ontology. These operators are used in the consequent of a SWRL rule to format the information matched by a rule antecedent. This antecedent is effectively treated as a pattern specification for the query. The prefix `sqwrl` is conventionally used for SQWRL built-ins. The core built-in defined by SQWRL is `sqwrl:select`. This built-in takes one or more arguments, which are typically variables used in the antecedent of a rule, and builds an internal table using the arguments as the columns of the table.

For example, the earlier rule to determine trial-eligible patients can be rewritten as a query as follows:

```
Patient(?p) ^ hasTreatment(?p, ?t) ^ hasDrug(?t, DDI) ^
              temporal:before(?t, "1999")
                  → sqwrl:select(?p)
```

This query will return a table with one column listing all patients who have completed a DDI drug therapy before 1999.

SQWRL queries have access to all built-in libraries available to SWRL and can thus be used to perform temporal queries using the full range of built-ins in the temporal library.

4.2 More Advanced Temporal Queries: Grouping and Aggregation

Temporal querying capabilities more advanced than the ones noted above are typically required by many applications. In addition to support for basic interval manipulations, many queries need more complex selection of results. For example, queries such as *List the first three doses of the drug DDI* or *Return the most recent dose of the drug DDI* are common. Because of OWL and SWRL's open world assumption, expressing these types of temporal queries on OWL ontologies can be difficult.

These queries require closure operations for correct formulation. The core SQWRL operators support some degree of closure when querying, and do so without violating OWL's open world assumption. As shown, queries like *List all patients in an ontology who had drug treatments for longer than three months* can be expressed fairly directly. However, queries with more complex closure requirements cannot be expressed using the core operators. For example, the query *List the first three DDI treatments for each patient* is not expressible. Basically, while the core operators support basic closure operations, no further operations can be performed on the results of these operators. Queries with negation or complex aggregation functionality are similarly not expressible using the core operators.

We have extended SQWRL to support the closure operations necessary for these types of temporal queries. Two types of collections are supported: *sets* and *bags*. As might be inferred, sets do not allow duplicate elements, whereas bags do. A built-in called `sqwrl:makeSet` is provided to construct a set. Its basic form is:

```
swqrl:makeSet(<set>, <element>)
```

The first argument of this set construction operator specifies the set to be constructed. The second specifies the element to be added to the set. This built-in constructs a single set for a particular query and will place the supplied element into the set. If a variable is specified in the element position, then all bindings for that variable in a query will be inserted into the set. A built-in called `sqwrl:makeBag` is provided to construct a bag. Its basic form is:

```
swqrl:makeBag(<set>, <element>)
```

The scope of each collection is limited to the query that contains it. Collection operators, such as, for example, `sqwrl:size`, can be applied to these collections. The results of these operators can be used by built-ins in the query, thus allowing access to the results of the closure operation. Two new SQWRL clauses are provided to contain

these collection construction and manipulation operators. The *collections construction* clause comes after a standard SWRL pattern specification and is separated from it using the ° character. It is followed by a *collections operation* clause that contains built-ins that operate on the collections, and is again separated from it by the ° character. In outline, a SQWRL query with collections operators looks as follows:

```
<SWRL Pattern Specification> °
<Collections Construction Clause> °
<Collections Operation Clause>
→ <Select Clause>
```

The construction clause can only contain SQWRL collection construction operators, such as `sqwrl:makeSet` and `sqwrl:makeBag`. The collection operators clause can contain only collection operators, such as `sqwrl:size`, in addition to built-ins that operate on the results of these operations. It may not contain other SWRL atom types.

Collections support basic closure operations over the entire ontology. However, the earlier query to list the first three DDI treatments for each patient is still not expressible using this approach. Additional collection operators to allow more complex queries that group related entities are also required. This additional expressivity is supplied by collections that are partitioned by a group of arguments. A built-in called `sqwrl:groupBy` provides this functionality. The general form of this grouping is:

```
sqwrl:makeSet(<set>, <element>) ^ sqwrl:groupBy(<set>, <group>)
```

or

```
sqwrl:makeBag(<bag>, <element>) ^ sqwrl:groupBy(<bag>, <group>)
```

This group can contain one or more entities. The first argument to the `sqwrl:groupBy` built-in is the collection, and the second and (optional) subsequent arguments are the entities to group by. Only one grouping can be applied to each collection.

Using this mechanism, the construction of a set of treatments for each patient can be written:

```
Patient(?p) ^ hasTreatments(?p, ?t) °
sqwrl:makeSet(?s, ?t) ^ sqwrl:groupBy(?s, ?p)
```

Here, a new set is built for each patient matched in the query and all treatments for each one are added to that patient's set.

SQWRL also provides standard operators such as `sqwrl:union`, `sqwrl:difference`, `sqwrl:intersection`, and so on. These operators employ standard set semantics and generate sets. Queries can use them to examine the results of two or more closure operations, permitting the creation of queries that are far more complex. Putting elements into collections effectively provides a closure mechanism. Clearly, two-phase processing is required for these queries—a query cannot, say, determine how many elements are in a collection until it has been constructed. As mentioned, the language restricts the atoms that are processed in the second phase to collection built-ins and other built-ins that operate on the results of these collection built-ins. That is, the first phase of query execution is analogous to standard rule execution. The second phase consists of operations on the collections constructed as a result of that execution.

More complex groupings have multiple grouping entities. For example, to make a set of the start times of each patient's treatment, the set construction operator must be supplied with both patient and treatment grouping arguments:

```
Patient(?p) ^ hasTreatment(?p, ?t) ^
temporal:hasValidTime(?t, ?vt) ^ temporal:hasStartTime(?bt, ?start) °
    sqwrl:makeSet(?s, ?start) ^ sqwrl:groupBy(?s, ?p, ?t)
```

Here, a set is constructed for each patient and treatment combination, and all the start times for the combination are added to the set.

Ordinal selection or aggregation operators can be applied to a collection if its elements have a natural ordering. These operators include `sqwrl:min`, `sqwrl:max`, `sqwrl:avg`, and so on. For example, a query to return the time of the first treatment for each patient can be written:

```
Patient(?p) ^ hasTreatment(?p, ?t) ^
temporal:hasValidTime(?d, ?vt) ^ temporal:hasStartTime(?vt, ?start) °
    sqwrl:makeSet(?s, ?start) ^ sqwrl:groupBy(?s, ?p, ?t) °
    sqwrl:min(?first, ?s) ^ temporal:equals(?first, ?start)
                → sqwrl:select(?p, ?start)
```

The result is a list of patients and the time of the first treatment for each one.

4.3 More Advanced Temporal Queries: Temporal Collections

Combining SQWRL's collection operators with the temporal valid-time model can provide support for very powerful selection operators on temporal results. Using SQWRL's grouping mechanism, aggregation operators such as earliest, latest, and so on can be supported. Supporting these operators is a key requirement for a temporal query language [8, 29].

We have further extended the temporal built-in library to support these types of collections. The library now supports the same set of construction and manipulation operators as SQWRL, but allows only temporal facts to be placed in the collections. The library natively understands the interval-based valid-time model underlying the facts placed in collections and provides collection operations on the collections. Additional temporal collection operators, such as `temporal:first`, `temporal:firstN`, `temporal:last`, `temporal:lastN`, `temporal:nth`, and so on, are also provided. Operators applied to these temporal collections consider the interval-based semantics of the entities in the collections. If two collections are merged, for example, intervals belonging to value-equivalent entities are merged. This process is known as *coalescing* [6]. The standard Allen temporal operators can also be applied to collections, facilitating queries such as *Were all DDI prescriptions before all AZT prescriptions?*

Consider, for example, a query to return the first treatment for each patient in an ontology, together with drug and dosage information. Again, assuming that each patient has a treatment property that holds a treatment class containing drug and dosage information which is modeled as a temporal fact using the temporal ontology, the query can be expressed as follows:

```
Patient(?p) ^ hasTreatment(?p, ?tr) ^
    hasDrug(?tr, ?drug) ^ hasDose(?tr, ?dose) °
temporal:makeSet(?trs, ?tr) ^temporal:groupBy(?trs, ?p) °
 temporal:first(?ftr, ?trs) ^ temporal:equals(?ftr, ?tr)
          → sqwrl:select(?p, ?tr, ?drug, ?dose)
```

A query to return the first three DDI treatments for each patient, together with dosage information, can be written:

```
Patient(?p) ^ hasTreatment(?p, ?tr) ^
hasDrug(?tr, DDI) ^ hasDose(?tr, ?dose) °
temporal:makeSet(?trs, ?tr) ^ temporal:groupBy(?trs, ?p) °
temporal:firstN(?f3tr, ?trs, 3) ^ temporal:equals(?f3tr, ?tr) →
sqwrl:select(?p, DDI, ?dose)
```

Here, the `temporal:firstN` built-in is used to select the first three treatments from a patient's treatment set.

A query to return the most recent DDI treatment for each patient, together with dosage information can be written:

```
Patient(?p) ^ hasTreatment(?p, ?tr) ^
hasDrug(?tr, DDI) ^ hasDose(?tr, ?dose) °
temporal:makeSet(?trs, ?tr) ^ temporal:groupBy(?trs, ?p) °
temporal:last(?ltr, ?trs) ^ temporal:equals(?ltr, ?tr) →
sqwrl:select(?p, DDI, ?dose)
```

Here, the `temporal:last` built-in is used to select the most recent treatment from each patient's treatment set.

As can be seen from these examples, natively supporting the valid time-model in collections allows expressive, yet relatively concise temporal queries. With SQWRL's grouping mechanism, queries with advanced aggregation requirements can be expressed directly in the languages. More advanced query functionality can also be supported by combining rules and queries to incrementally generate intermediate results at successively higher levels of abstraction.

5 Conclusions

We have created a lightweight, yet expressive temporal model that can be used to encode the temporal dimension of data in OWL ontologies. Our model has been designed to be integrated with existing ontologies with minimal redesign of them. It facilitates the consistent representation of temporal information in ontologies, thus allowing standardized approaches to performing temporal reasoning and temporal queries. We used SWRL and the SWRL-based OWL query language SQWRL to show how knowledge-level temporal rules and queries can be constructed on the information in the ontologies. The primary goal of our approach was to develop a concise, yet expressive query language. To date, OWL-based temporal research efforts tend to emphasize temporal representation, with comparatively little emphasis on methods to query the information in them. A particular failing of existing systems is poor support for queries with aggregates, a failing tends to limit the types of queries than be expressed. This paper shows how we have provided these functionalities by developing temporal extensions to SQWRL and demonstrates how the resulting language provides powerful mechanisms for expressing complex temporal queries. In particular, we show that extending SQWRL with collection operators that can be directly applied to data described using the temporal model provides a high degree of expressivity.

We used an initial version of the temporal valid-time model described here to encode temporal information collected during a national clinical trials project [25]. Other researchers have reported using our model in a hypertension management application to identify patients who satisfy a set of evidence-based criteria for quality improvement potential [21]. We are currently using the updated model with the recent set-based SQWRL extensions to reason with breast cancer image annotation for tumor assessment [19, 27].

A possible shortcoming of our approach is that all temporal information in a source ontology must be transformed to conform to the valid-time model. This process can be time-consuming, and typically requires considerable domain expertise. However, the mapping requirement is not unique to our method. If principled temporal reasoning mechanisms are to be applied to temporal information, some sort of mapping process to regularize the information is nearly always required, irrespective of the final reasoning processes.

An additional possible shortcoming is that complex temporal rules and queries can become difficult to maintain and extend as their numbers increase. We are developing management tools to tackle this problem [13].

The methodologies and tools described in this paper aim to enhance the ability of software developers and investigators to encode critical forms of temporal knowledge in their applications. This knowledge can be represented directly in domain ontologies, facilitating much higher-level analyses than would be possible with lower-level techniques. Ultimately, working at the knowledge level will enable investigators to make better sense of the complex temporal patterns typical in many domains.

Acknowledgements. This research was supported in part by grant R01LM009607 from the National Library of Medicine.

References

1. Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM 26(11) (1983)
2. Amagasa, T., Yoshikawa, M., Uemura, S.: A Data model for temporal XML documents. In: Ibrahim, M., Küng, J., Revell, N. (eds.) DEXA 2000. LNCS, vol. 1873, pp. 334–344. Springer, Heidelberg (2000)
3. Baratis, E., Petrakis, E.G.M., Batsakis, S., Maris, N., Papadakis, N.: TOQL: Temporal ontology querying language. In: Mamoulis, N., Seidl, T., Pedersen, T.B., Torp, K., Assent, I. (eds.) SSTD 2009. LNCS, vol. 5644, pp. 338–354. Springer, Heidelberg (2009)
4. Bedi, P., Marwah, S.: Versioning OWL ontologies using temporal tags. In: World Academy of Science, Engineering, and Technology (March 27, 2007)
5. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (2001)
6. Bohlen, M.H., Snodgrass, R.T., Soo, M.D.: Coalescing in temporal databases. In: Proceedings of the International Conference on Very Large Databases, Mumbai, India (1996)
7. Bry, F., Koch, C.: Querying the web reconsidered: design principles for versatile web query languages. Journal of Semantic Web and Information Systems (2005)
8. Chatziantoniou, D., Ross, K.A.: Querying multiple features of groups in relational databases. In: Proceedings of the International Conference on Very Large Databases (1996)

9. Das, A.K., Musen, M.A.: A temporal query system for protocol-directed decision support. Methods of Information in Medicine 33, 358–370 (1994)
10. Frasincar, F., Milea, V., Kaymak, U.: tOWL: integrating time into OWL. In: Semantic Web Information Management. Springer, Heidelberg (2010)
11. Gao, C., Snodgrass, R.: Temporal slicing in the evaluation of XML queries. In: 29th International Conference on Very Large Databases, Berlin, Germany (2003)
12. Gutierrez, C., Hurtado, C.A., Vaisman, A.: Introducing time into RDF. IEEE Transactions on Knowledge and Data Engineering 19(2), 207–218 (2007)
13. Hassanpour, S., O'Connor, M.J., Das, A.K.: Exploration of SWRL Rule Bases through Visualization, Paraphrasing, and Categorization of Rules. In: Governatori, G., Hall, J., Paschke, A. (eds.) RuleML 2009. LNCS, vol. 5858, pp. 246–261. Springer, Heidelberg (2009)
14. Hobbs, J.R., Pan, F.: An ontology of time for the Semantic Web. ACM Transactions on Asian Language Processing (TALIP): Special issue on Temporal Information Processing 3(1), 66–85 (2004)
15. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: a Semantic Web rule language combining OWL and RuleML. In: W3C (2004)
16. Hripcsak, G., Ludemann, P., Allan Pryor, T., Wigertz, O.B., Clayton, P.: Rationale for the Arden Syntax. Computers and Biomedical Research 27, 291–324 (1994)
17. Kim, S.K., Song, M.Y., Kim, C., Yea, S.J., Jang, H.C., Lee, K.C.: Temporal Ontology Language for Representing and Reasoning Interval-Based Temporal Knowledge. In: Domingue, J., Anutariya, C. (eds.) ASWC 2008. LNCS, vol. 5367, pp. 31–45. Springer, Heidelberg (2008)
18. Krieger, H.-U.: Where temporal description logics fail: Representing temporally-changing relationships. In: Dengel, A.R., Berns, K., Breuel, T.M., Bomarius, F., Roth-Berghofer, T.R. (eds.) KI 2008. LNCS (LNAI), vol. 5243, pp. 249–257. Springer, Heidelberg (2008)
19. Levy, M., O'Connor, M.J., Rubin, D.L.: Semantic reasoning with image annotations for tumor assessment. In: San Francisco, C.A. (ed.) AMIA Annual Symposium, San Francisco, CA (2009)
20. Lutz, C., Wolter, F., Zakharyashev, M.: Temporal description logics: a survey. In: 15th International Symposium on Temporal Representation and Reasoning (2008)
21. Mabotuwana, T., Warren, J.: An ontology-based approach to enhance querying capabilities of general practice medicine for better management of hypertension. Artificial Intelligence in Medicine 47(2), 87–103 (2009)
22. Manola, F., Miller, E.: RDF Primer. In: W3C Recommendation (2004)
23. McGuinness, D.L., van Harmelen, F.: OWL web ontology language overview. In: W3C (2004)
24. O'Connor, M.J., Tu, S.W., Musen, M.A.: The Chronus II temporal database mediator. In: AMIA Annual Symposium, San Antonio, TX, pp. 567–571 (2002)
25. O'Connor, M.J., Shankar, R.D., Parrish, D.B., Das, A.K.: Knowledge-data integration for temporal reasoning in a clinical trial system. International Journal of Medical Informatics 78(1), S77–S85 (2009)
26. O'Connor, M.J., Das, A.K.: SQWRL: a query language for OWL. In: OWL: Experiences and Directions (OWLED), Fifth International Workshop, Chantilly, VA (2009)
27. O'Connor, M.J., Das, A.K.: Semantic reasoning with XML-based biomedical information models. In: 13th World Congress on Medical Informatics (MedInfo 2010), Cape Town, South Africa (2010)
28. Pugliese, A., Udrea, O., Subrahmanian, V.S.: Scaling RDF with time. In: WWW Conference, pp. 605–614 (2008)

29. Rafiq, M.I., O'Connor, M.J., Das, A.K.: Computational method for temporal pattern discovery in biomedical genomic databases. In: IEEE Computational Systems Bioinformatics Conference (CSB 2005) (2005)
30. Snodgrass, R.T.: The TSQL2 Temporal Query Language. Kluwer, Boston (1995)
31. Snodgrass, R.T., Böhlen, M.H., Jensen, C.S., Steiner, A.: Transitioning temporal support in TSQL2 to SQL3. In: Etzion, O., Jajodia, S., Sripada, S. (eds.) Dagstuhl Seminar 1997. LNCS, vol. 1399, pp. 150–194. Springer, Heidelberg (1998)
32. SWRL Temporal Built-ins (2010), http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTemporalBuiltIns
33. SWRL Temporal Ontology (2010), http://swrl.stanford.edu/ontologies/built-ins/3.3/temporal.owl
34. Tappolet, J., Bernstein, A.: Applied Temporal RDF: Efficient Temporal Querying of RDF Data with SPARQL. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 308–322. Springer, Heidelberg (2009)
35. Visser, U.: Intelligent Information Integration for the Semantic Web. LNCS (LNAI), vol. 3159. Springer, Heidelberg (2004)
36. Welty, C., Fikes, R.: A reusable ontology for fluents in OWL. In: Formal Ontology in Information Systems: Proceedings of the Fourth International Conference (FOIS 2006), pp. 226–336. IOS Press, Amsterdam (2006)
37. Wiederhold, G.: Databases for healthcare. Lecture Notes in Medical Informatics. Springer, Heidelberg (1981)
38. XML Schema (2009), http://www.w3.org/TR/xmlschema11-1/
39. Zhang, S., Dyreson, C.E.: Adding Valid Time to XPath. In: Bhalla, S. (ed.) DNIS 2002. LNCS, vol. 2544, pp. 29–42. Springer, Heidelberg (2002)