A LIGHTWEIGHT MODEL FOR REPRESENTING AND REASONING WITH TEMPORAL INFORMATION IN BIOMEDICAL ONTOLOGIES

Martin J. O'Connor and Amar K. Das
Stanford Center for Biomedical Informatics Research, Stanford University, Stanford, California 94305, U.S.A.

Keywords: Temporal models, Temporal reasoning, Temporal queries, Semantic Web, SWRL, SQWRL, OWL.

Abstract: Over the past decade, the number, size, and complexity of databases for health-related research have grown dramatically. Ontologies are being developed and used by many scientific communities to support sharing, integration, and management of the diverse information in these databases. As critical as ontologies have become, ontology language such as OWL typically provide minimal support for modeling the complex temporal relationships that are common in biomedical research data. As a result, ontologies often cannot fully express the temporal knowledge needed by many biomedical applications and thus users and developers must pursue *ad hoc* solutions to these challenges. In this paper, we present a methodology and set of tools for representing temporal information in biomedical ontologies. This approach uses a lightweight temporal model to encode the temporal dimension of biomedical 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 this model.

1 INTRODUCTION

The past decade of research has seen a growing consensus regarding the critical nature of controlled terminologies and ontologies in the construction of biomedical systems. Ontologies are used to convey the biomedical meaning of experiments in a computer-accessible format, and they permit integration of data and knowledge from diverse sources by standardized labelling of concepts. The use of ontologies to represent biomedical knowledge will be particularly essential to the next generation of Web-enabled applications in healthcare and biomedical research. The Semantic effort (Berners-Lee et al., 2001) aims to provide languages and tools to provide explicit semantic meaning for data and knowledge shared among these types of applications. In particular, the Ontology Web Language (OWL; McGuinness and Harmelen, 2004) and its associated Semantic Web Rule Language (SWRL; Horrocks et al., 2004) provide a powerful standardized approach for representing and reasoning with information.

Despite the power of these technologies, they have very limited support for the modeling of temporal information. OWL, for example, provides no temporal support beyond allowing data values to be typed as basic XML Schema dates, times or durations (XML Schema, 2009). SWRL includes operators for manipulating these temporal values (SWRL, 2004) but, again, these operators work at a very low-level. There are no standard high-level mechanisms to consistently represent and reason with temporal information.

Since the temporal dimension is central in practically all biomedical data, these representation shortcomings have significant system implementation consequences. Primarily, they restrict the complexity of the temporal information that can be represented in biomedical ontologies. In addition, they reduce the possibilities for automated validation of this information. Crucially, these restrictions also limit the temporal expressivity of the deductive rules and queries that can be formulated over ontology-encoded biomedical data. Formulating these rules and queries thus requires custom solutions using technologies that may not leverage the formal knowledge representation techniques provided by ontologies. There is a pressing need for solutions that provide robust knowledge-level mechanisms for representing and reasoning with temporal information in ontologies.

O'Connor M. and Das A. (2010).
A LIGHTWEIGHT MODEL FOR REPRESENTING AND REASONING WITH TEMPORAL INFORMATION IN BIOMEDICAL ONTOLOGIES.
In *Proceedings of the Third International Conference on Health Informatics*, pages 90-97
DOI: 10.5220/0002747300900097
Copyright © SciTePress

2 BACKGROUND

Historically, the centrality of time in biomedical applications has driven the development of custom temporal management solutions. One of the first medical systems to address the temporal representation problem was the Time Oriented Database (TOD; Wiederhold, 1981). TOD had a three-dimensional view of clinical data, with time represented explicitly as one of the dimensions. Data relating to a particular patient visit, for example, were indexed by patient identifier, clinical parameter type, and visit time. TOD supported a set of basic temporal queries that allowed data values following certain temporal patterns to be extracted. The later Arden Syntax (Hripsack et al., 1994) 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 about associated data, 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 interval-based representation of temporal data. 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 it 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 point at which the information was recorded, or the point at which 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 (Snodgrass et al., 1998).

More than a dozen formal extensions to the relational data model were proposed. These multiple approaches ultimately led to efforts to develop a consensus language, called TSQL2 (Snodgrass, 1995). This language supports both temporal and non-temporal tables, and 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. No complete implementations of TSQL2 were produced, however, but several temporal query systems were written using its core features. The Chronus system (Das and Musen, 1994) and its later evolution, Chronus II (O'Connor et al., 2002) were both developed to perform temporal queries in clinical decision support systems. Like TSQL2, they used a SQL-based language for querying temporal data in relational databases. These systems were successfully used in several biomedical applications (Nguyen at al., 1999; Goldstein et al., 2004).

Experience with these systems illustrated that the entire TSQL2 specification is not necessary for querying biomedical data. A few simple language extensions can provide a large increase in expressivity. The most important lesson learned is that a principled temporal model is key to developing these 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 (Snodgrass, 1995). While numerous models were proposed to represent temporal information in both relational databases and 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 they are held to be true. Such facts have a value and one or more valid times. In other words, every temporal fact holds information denoting the facts's *valid-time*. Conceptually, this representation means that every temporal fact is held to be true or *valid* during the time or times associated with this fact. No conclusions can be made about the fact for time periods outside of its valid-time. The valid-time model effectively provides a mechanism to standardize 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 for use in relational systems, however, and can be used in any information system that requires a consistent representation of temporal information.

3 TEMPORAL MODEL

The valid-time model has been used in ontology-based systems. Shahar (1999), for example, has made extensive use of this model in clinical decision support systems using a Frame-based ontological representation. Adding a temporal dimension to the OWL ontology language is not straightforward, however. OWL does not provide any constructs for modeling time. As with the relational model, a simple instant timestamp representation is all that it supports. More importantly, OWL's logic-based formalism makes it difficult to model dynamically changing information. Some formal temporal extensions have been developed (e.g., Sim et al., 2008) but these proposals are fairly elaborate and none has resulted in practical and usable representations.

Rather than extending OWL's logical model, other researchers have attempted to support temporal representations on top of OWL. For example, OWL-Time (Hobbs and Pan, 2004) proposes an ontology that provides rich description of temporal instants, intervals, durations, and calendar terms. However, this representation is not all that 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. In recent work, researchers have described the development of a user-level valid-time model in OWL (O'Connor et al., 2009). This model was used to encode all clinical data collected during a clinical trial. A constraint language was developed using this model and was used to specify the temporal constraints contained in clinical trial documents.

We have adopted the temporal valid-time model used in this system and simplified it so that it can more easily be integrated with existing ontologies. This enhanced model was designed to be lightweight, thus allowing it to be layered on existing ontologies without requiring significant redesign of these ontologies. This model concerns itself with time only and provides a simple approach to adding a temporal dimension to existing entities in domain ontologies. Typically, these entities will be in the information model part of these ontologies, though the temporal model can also be used to add temporal information to other entities. The temporal model conforms closely to the valid-time temporal model used by Chronus II (O'Connor et al., 2002), which was based on an earlier model by Shahar (1999).

We developed an ontology in OWL to encode this valid-time temporal model. We henceforth use the prefix `temporal` for entities defined in this ontology. The core class modeling 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 or times during which the associated information is held to be true. Values of this property are modeled by a class called `temporal:ValidTime`, which has subclasses `temporal:ValidInstant` and `temporal:ValidInterval`, which represent instants and intervals, respectively. The class `temporal:ValidInstant` is associated with the property `temporal:hasTime`, and the `temporal:ValidInterval` class, 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 `temporal:hasGranularity` property with a range class called `temporal:Granularity`. Specific granularities, such as days and minutes, are represented 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 an investigator wishes to add a temporal dimension to a blood pressure measurement that is described using a class called `BloodPressureMeasurement`, 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, which will allow them to consistently record the temporal information associated with the measurement. Similarly, if the investigator wishes to work with prescriptions using an existing class called

`Prescriptions` they might chose to use 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 that we wish to add a temporal dimension to, effectively asserting that instances of that class have a temporal extent. For example, if an investigator wishes to take the earlier blood pressure measurements class and model it as a temporal fact they can simply take the class and make it a subclass of the `temporal:Fact` class. 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 the `temporal:ValidInterval` class 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 so does not necessitate modifications to 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 REASONING AND 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 (Horrocks et al., 2004; SWRL, 2004) provides a small set. However, the operators in this set are very basic, providing simple instant-based comparisons only.

4.1 Basic Rules and Queries

Fortunately, 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 (1983) interval-based temporal operators. About two dozen built-ins implementing the entire set of the Allen operators are provided by this library. The library 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. This 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 this model. It can thus be used in rules to directly reason about valid time instants and intervals.

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. These valid times can any combination of instant or intervals. This rule classifies patients as trial-eligible if they have any completed 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, ?t) ^
hasDrug(?t, DDI) ^
temporal:hasValidTime(?t, ?tVT) ^
temporal:before(?tVT, "1999")
→ TrialEligible(?p)
```

The temporal built-ins can take any combination of valid-time instants, valid-time intervals, or XSD date or datetime literal values. In this case, the `temporal:before` built-in is supplied with a valid-time interval and a literal date value.

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 (SQWRL; O'Connor and Das, 2009) has been developed that 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:hasValidTime(?t, ?tVT) ^
temporal:before(?tVT, "1999")
→ sqwrl:select(?p)
```

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

4.2 More Advanced Queries

Operators to construct and manipulate sets are provided by SQWRL to provide more advanced querying functionality. 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 and the second specifies the element to be added to the set. This built-in will construct a single set for a particular query and will place all supplied elements into the set. Operators like `sqwrl:isEmpty`, `sqwrl:size`, `sqwrl:union`, and `sqwrl:difference` can then be applied to the resulting sets. SQWRL provides an additional clause to contain these set construction and manipulation operators. This clause comes at the end of the standard pattern specification and is separated from it using the ° character. For example, a query to list the number of patient in an ontology can be written:

```
Patient(?p) °
sqwrl:makeSet(?s, ?p) ^ sqwrl:size(?n, ?s)
→ sqwrl:select(?n)
```

Additional query features such as negation and disjunction can then be provided by set operators. For example, a query to list the number of *non* DDI drugs in an ontology can be written:

```
Drug(?d) °
sqwrl:makeSet(?sd, ?d) ^
sqwrl:makeSet(?sddi, DDI) ^
sqwrl:difference(?snonddi,  ?sd,  ?sddi)  ^
sqwrl:size(?n, ?snonddi) → sqwrl:select(?n)
```

These types of set operators support some fairly basic functionality. Additional set construction operators are required to allow more complex queries that support grouping of related sets of entities. This additional expressivity is supplied in SQWRL by grouped sets. These sets are partitioned by a group of arguments. This group is specified in a set grouping operator. The form of this grouping is:

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

This group can contain one or more entities. This grouping mechanism is analogous to GROUP BY clause in SQL.

For example, the construction of a set of treatments for each patient can be written:

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

Here, sets will be constructed for each patient and all treatments for a patient will be added to that patient's set.

More complex groupings will 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 will be constructed for each patient and treatment combination and all the start times for that combination will be added to the set.

Ordinal selection or aggregation operators can then be applied to a set if its elements are numeric or 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 will be a list of patients together with the time of the first treatment for each patient.

These set operations can be used with the temporal valid-time model to construct complex temporal queries. Consider, for example, the following query from the HIV domain:

List the average viral loads of all patients over two 4-day windows starting 12 and 24 days after initiation of a new treatment. Order those results by the maximum average viral load of the first window.

Assume each patient has treatment and laboratory properties modeled as facts using the valid-time model, with laboratory value timestamps stored as valid instants and treatment intervals stores as valid intervals. The query can then be expressed as:

```
Patient(?p) ^ hasTreatment(?p, ?t) ^
hasValidTime(?t,?tvt) ^
temporal:start(?tvt, ?b) ^
temporal:add(?w1Start, ?b, 12, days) ^
temporal:add(?w1End, ?b, 14, days) ^
temporal:add(?w2Start, ?b, 24, days) ^
temporal:add(?w2End, ?b, 28, days) ^
hasLab(?p, ?l1) ^ hasViralLoad(?l1, ?vl1) ^
temporal:hasValidTime (?l1, ?lvt1) ^
temporal:hasTime(?lvt1, ?l1t) ^
hasLab(?p, ?l2) ^ hasViralLoad(?l2, ?vl2) ^
temporal:hasValidTime (?l2, ?lvt2) ^
temporal:hasTime(?lvt2, ?l2t) ^
temporal:contains(?w1Start ,?w1End, ?l1t) ^
temporal:contains(?w2Start, ?w2End, ?l2t) °
sqwrl:makeSet(?sw1, ?vl1) ^
sqwrl:groupBy(?sw1, ?p, ?t) ^
sqwrl:makeSet(?sw2,?vl2) ^
sqwrl:groupBy(?sw2, ?p, ?t) ^
sqwrl:avg(?avl1, ?sw1) ^
sqwrl:avg(?avl2, ?sw2) →
sqwrl:select(?p, ?t, ?avl1, ?avl2) ^
sqwrl:orderBy(?avl1)
```

This query first extracts each treatment's baseline time and then calculates the two windows after that baseline, finds the viral loads that occur during those two windows, inserts each of them into distinct sets, and then calculates the average viral load values for each of those sets. It then orders the results by the average viral load of the first temporal window. The result will be a list of patients and their treatments together with the average viral loads during the two windows after the start of each treatment.

As can be seen, these queries can quickly become complex. In most cases, however, a combination of rules and queries can be combined to incrementally generate intermediate results as successively higher levels of abstraction so that the final query can be considerably shorter. These intermediate results can also be reused by other rules and queries.

4.3 Set-based Temporal Queries

Even more advanced temporal querying capabilities are typically required by many systems. In addition to support for basic interval manipulations, many queries will 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. An additional approach to simplifying these types of temporal queries is to directly support the manipulation of temporal facts in set operations. Instead of just supporting standard OWL entities such as classes, properties, individuals, and data values, these sets can natively understand the interval-based valid-time model underlying the facts placed in the set. They can thus support more powerful selection operators on temporal results, providing operations such as earliest, latest and so on.

We have extended the temporal built-in library to support these types of sets. The temporal library now supports the same set of construction and manipulation operators as the SQWRL library but allows only temporal facts to be placed in these sets. Additional temporal set operators, such as `temporal:first`, `temporal:firstN`, `temporal:last`, `temporal:lastN`, `temporal:nth`, and so on, are also provided. Operators applied to these temporal sets consider the interval-based semantics of the entities contained in the sets. So, if two sets are merged, for example, intervals belonging to value-equivalent entities are merged, a process known as coalescing (Bohlen et al., 1996). The standard Allen temporal operators can also be applied to sets, thus facilitating queries such as "Were all DDI prescriptions before all AZT prescriptions?"

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

```
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 for those treatments 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 each 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 sets can permit expressive yet relatively concise temporal queries.

5 CONCLUSIONS

We described a lightweight yet expressive temporal model that can be used to encode the temporal dimension of biomedical data in OWL ontologies. This model is designed to be integrated with existing ontologies without requiring redesign of those ontologies. It facilitates the consistent representation of temporal information in those ontologies, thus allowing standardized approaches to performing temporal reasoning and temporal queries on these ontologies. Using the rule language SWRL and the SWRL-based OWL query language SQWRL we show how knowledge-level temporal rules and queries can be constructed on the information contained in these ontologies. In particular, we show that extending SQWRL with set 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 the temporal information collected during a national clinical trials project (O'Connor et al., 2009). As mentioned, we developed a temporal constraint language on top of this model. 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 (Mabotuwana et al., 2009). We are currently using the updated model with the recent set-based SQWRL extensions to reason with breast cancer image annotation for tumor assessment (Levy et al., 2009).

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 mapping process can be time consuming and typically requires considerable domain expertise. However, 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 the number of them increases. We are developing rule management tools to tackle this problem (Hassanpour at al., 2009).

The methodologies and tools described in this paper aim to enhance the ability of software developers and investigators to encode critical forms of deductive biomedical knowledge in their applications. This knowledge can be represented directly in domain ontologies thus 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 large numbers of complex temporal patterns that characterize dynamic and causal phenomena in medicine and biology.

The ontologies and tools mentioned in this paper are freely available as an open-source plug-ins to the Protégé-OWL ontology development environment (Knublauch et al., 2004).

ACKNOWLEDGEMENTS

This research was supported in part by grant 1R01LM009607 from the National Library of Medicine.

REFERENCES

Allen, J.F., 1983. Maintaining knowledge about temporal intervals, *Communications of the ACM*, 26(11).

Berners-Lee, T., Hendler, J. and Lassila, O., 2001.The Semantic Web, *Scientific American*, pp. 35-43.

Bohlen, M.H., Snodgrass, R.T., Soo, M.D., 1996. Coalescing in temporal databases. *Proceedings of the International Conference on Very Large Databases*, Mumbai, India, pp. 180-191.

Das, A.K. and Musen, M.A., 1994. A temporal query system for protocol-directed decision support, *Methods of Information in Medicine*, 33: 358-370.

Goldstein, M.K., Coleman, R.W., Tu, S.W., O'Connor, M.J., Martins, S.B., Lavori, P.W., Shilpak, M.G., Oddone, E.Z., Advani, A., Gholami, P., Hoffman, B.B., Shankar, R.D., Musen, M.A., 2004. Translating research into practice: organizational issues in implementing automated decision support for hypertension in three medical centres. *Journal of the American Medical Informatics Association*, 11(5): 368-376.

Hassanpour, S., O'Connor, M.J., Das, A.K., 2009. Exploration of SWRL rule bases through visualization, paraphrasing, and categorization of rules. *International RuleML Symposium on Rule Interchange and Applications*, Las Vegas, NV, Springer-Verlag, LNCS 5858, pp. 246–261.

Hobbs, J. R. and Pan, F., 2004. 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.

Horrocks I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M., 2004. SWRL: a Semantic Web rule language combining OWL and RuleML. *W3C*.

Hripcsak, G., Ludemann, P., Allan Pryor, T., Wigertz, O.B., Clayton, P., 1994. Rationale for the Arden Syntax. *Computers and Biomedical Research*, 27: 291-324.

Knublauch, H., Fergerson, R.W., Noy, N.F., Musen, M.A., 2004. The Protégé OWL Plugin: An open development environment for semantic web applications, *Third International Semantic Web Conference*, Hiroshima, Japan, pp. 229-243.

Levy, M., O'Connor, M.J., Rubin, D.L., 2009. Semantic reasoning with image annotations for tumor assessment. *AMIA Annual Symposium*, San Francisco, CA.

Mabotuwana, T., Warren, J., 2009. 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.

Nguyen, J.H., Shahar, Y., Tu, S.W., Das, A.K., and Musen, M.A., 1999. Integration of temporal reasoning and temporal data maintenance into a reusable database mediator to answer abstract, time-Oriented queries: the Tzolkin System, *Journal of Intelligent Information Systems,* 13(1/2): 121-145.

O'Connor, M.J., Tu, S.W., and Musen, M.A., 2002. The Chronus II temporal database mediator, AMIA *Annual Symposium*, San Antonio, TX, pp. 567-571.

O'Connor, M.J., Shankar, R.D., Parrish, D.B., Das, A.K., 2009. Knowledge-data integration for temporal reasoning in a clinical trial system, *International Journal of Medical Informatics,* 78(1): S77-S85.

O'Connor, M.J. and Das, A.K., 2009. SQWRL: a query language for OWL. *OWL: Experiences and Directions (OWLED), Fifth International Workshop*, Chantilly, VA.

Shahar, Y. and Musen, M.A., 1993. RÉSUMÉ: a temporal-abstraction system for patient monitoring. *Computers and Biomedical Research*, 26: 255-273.

Shahar, Y., 1999. Timing is everything: temporal reasoning and temporal data maintenance in medicine, *Seventh Joint European Conference on Artificial Itelligence in Medicine and Medical Decision Making)*, Aalborg, Denmark.

Sim, S.K., Song, M.Y., Kim, C., Yea, S.J., Jang, H.C., and Lee, K.C., 2008.Temporal ontology language for representing and reasoning interval-based temporal knowledge, *LNCS 5367*, pp. 31–45.

Snodgrass, R.T., 1995. *The TSQL2 Temporal Query Language*, Boston, MA: Kluwer.

Snodgrass, R.T., Jensen C.S., Steiner, A., 1998. *Transitioning temporal support in TSQL2 to SQL3*. *Temporal Databases: Research and Practice*, pp. 150-194.

SWRL Temporal Built-ins, 2009. Available at: http:// protege.cim3.net/cgi-bin/wiki.pl?SWRLTemporalBuiltIns

Wiederhold, G., 1981 Databases for healthcare. *Lecture Notes in Medical Informatics*, Heidelberg, Germany, Springer-Verlag.

XML Schema, 2009. http:// www.w3.org/TR/xmlschema11-1/