Including Domain-Specific Reasoners with Reusable Ontologies

Richard Fikes Jessica Jenkins Qing Zhou
Knowledge Systems Laboratory
Computer Science Department
Stanford University, Stanford, CA 94305
{fikes, jessicaj, zhou}@ksl.stanford.edu

Key words: Reusable Knowledge, Ontologies, Temporal Reasoning, Intelligent Knowledge-Based Systems, Knowledge Engineering

Abstract

In this paper, we extend the methodology for developing intelligent systems that promotes building knowledge bases by assembling reusable ontologies and knowledge base modules from on-line libraries to include associating domain-specific reasoners with reusable ontologies and incorporating such reasoners into an intelligent system as part of the knowledge assembly process. The methodology is based on an existing hybrid reasoning architecture that provides an API and functional specification for special-purpose reasoners. We illustrate the methodology by describing a reusable time ontology and an associated set of domain-specific reasoners for the vocabulary of relations defined in that ontology. The ontology is designed to support large-scale knowledge system applications by providing an easily understandable, flexible, formally defined, and effective means of representing knowledge about time and the standard components and properties of calendars. The domain-specific reasoners support "telling and "asking" operations for temporal ordering relations on time points and the Allen relations on time intervals.

1. Introduction

In order for an agent to make statements and ask queries about a subject domain, it must use a conceptualization of that domain. A domain conceptualization names and describes the entities and the relationships among those entities that are to be considered in that domain. It therefore provides a vocabulary for representing and communicating knowledge about the domain.

Explicit specifications of domain conceptualizations, called *ontologies*, are essential for the development and use of intelligent systems as well as for the interoperation of heterogeneous systems. They provide a vocabulary for a domain and can be used as building block components of knowledge bases, object schema for object-oriented systems, conceptual schema for data bases, structured glossaries for human collaborations, vocabularies for communication between agents, class definitions for conventional software systems, etc.

Ontology construction is difficult and time consuming. This high development cost is a major barrier to the building of large scale intelligent systems and to widespread knowledge-level interactions of computer-based agents. Since many conceptualizations are intended to be useful for a wide variety of tasks, an important means of removing this barrier is to encode ontologies in a reusable form so that large portions of an ontology for a given application can be assembled from existing ontologies available from ontology libraries.

Intelligent systems are characterized by their ability to effectively reason with their knowledge. Experience with automated reasoners has made increasingly clear that in order to effectively deal with the demands of complex "real world" reasoning problems involving the use of heterogeneous knowledge, general-purpose reasoners need to be augmented with special-purpose reasoners that embody both domain-specific and task-specific expertise. That is, such problems require *hybrid reasoning* (as described, e.g., in [10] and [15]).

The system development methodology of assembling ontologies and knowledge bases from libraries has a significant deficiency in that it is currently missing the needed duel capability for assembling domain-specific reasoners from libraries that embody expertise for effectively reasoning with the assembled knowledge. In this paper we describe a means of achieving that capability and a case study that illustrates its use. In particular, we describe a means of including domain-specific reasoners with an ontology so that when the ontology is incorporated into a knowledge base, the accompanying reasoners can be incorporated into the system's automated reasoner facilities.

2. Adding Domain-Specific Reasoners to Ontologies

The vocabulary that an ontology specifies typically includes relation and function names. We characterize the general kinds of reasoning that are to be done with relations and functions as follows:

Evaluate Ground Atomic Expressions – Given a relational sentence S or function term F all of whose arguments are constants, determine the truth value of S or a constant that is equal to F. For example, determine that the sentence `(< 2 3)` is true or that the term `(+ 2 3)` is equal to 5.

Derive Instances of Non-Ground Relational Sentences – Given a relational sentence S having at least one argument that is not a constant, determine instances of S that are entailed by the knowledge base. For example, determine an instance of `(has-parent Joe ?p)` for each constant that the knowledge base entails is a parent of Joe.

Derive Sentences to be Cached – Given a sentence S that is being "told" (i.e., added) to a knowledge base K, determine additional sentences to be told to K that are entailed by K ∪ S.

An automated reasoner uses evaluators of ground atomic expressions to avoid having to store and retrieve (perhaps an infinite number of) the true instances in its knowledge base; uses derivers of instances of non-ground relational sentences while searching for answers to queries (i.e., during backward chaining); and uses derivers of sentences to be cached while storing a knowledge base (i.e., during forward chaining).

3. The JTP Hybrid Reasoning Architecture

The primary enabler for effectively incorporating domain-specific reasoners into a system is a hybrid reasoning architecture for the system's automated reasoning facilities. We have developed such an architecture for hybrid reasoning (called the JTP architecture), a library of general-purpose reasoning system components (called the JTP library) that supports rapid development of reasoners and reasoning systems using the JTP architecture, and a multi-use reasoning system (called the JTP system) employing the JTP architecture and library.

The JTP hybrid reasoning system architecture and reasoning system component library is intended to enable the rapid building, specializing, and extending of hybrid reasoning systems. Each reasoner in a JTP hybrid reasoning system can embody special-purpose algorithms that reason more efficiently about particular commonly-occurring kinds of information. In addition, each reasoner can store and maintain some of the system's knowledge, using its own specialized representations that support faster inference about the particular kinds of information for which it is specialized.

The JTP architecture is representation language independent. The implemented JTP reasoning system uses a first-order logic (FOL) representation language (i.e., KIF 3.0 [13]).

The JTP architecture assumes that there is a single initially empty knowledge base (KB) with respect to which all processing is done. A KB is considered to be a representation of a logical theory and to contain a set S of symbolic logic sentences and a set of justifications for each sentence in S. The architecture supports commands for loading a KB, adding an axiom to a loaded KB, removing an axiom from a loaded KB, and asking what (partial or full) instances of a sentence schema[1] are entailed by a loaded KB.

The primary work of the system is assumed to be performed by modules called reasoners. There are "telling" reasoners that are invoked when a sentence is being added to the KB and "asking" reasoners that are invoked when the KB is being queried. Reasoners produce "reasoning steps", each of which is a partial or completed proof. The reasoning steps produced by telling reasoners are completed proofs of additional sentences that are inferred from the reasoner's input. The reasoning steps produced by asking reasoners are partial or completed proofs of candidate answers to a query.

The output of a reasoner is an "enumerator" that can be "pulsed" to obtain the next reasoning step produced by the reasoner. Enumerators enable a reasoner to produce its output reasoning steps one or more at a time. Thus, the pulsing of an enumerator can cause the enumerator to output the next reasoning step already produced by a reasoner and stored in the enumerator or to invoke the reasoner so that it can produce the next one or more reasoning steps.

A reasoning system using the JTP architecture needs some means of determining to which of its arbitrary number of reasoners to route its inputs. That capability is provided by reasoners in the system that act as "dispatchers" of an input to other reasoners that the dispatcher determines may be able to process the input. Each dispatcher has a set of child reasoners associated with it and serves as a transparent proxy for those child reasoners.

4. A Case Study Using the JTP Architecture

We recently expanded our JTP reasoning system by developing a capability for reasoning with time-dependent knowledge. We accomplished this by first developing a "Reusable Time" ontology [11] that provided definitions for time points, time intervals, temporal relations on time points such as `before` and `after`, the Allen relations on time intervals [1], calendar objects such as the months of the year, etc. We then developed domain-specific telling and asking reasoners for the temporal relations on time points and the Allen relations on time intervals that made use of a

[1] A sentence schema is a sentence optionally containing free variables. An instance of a sentence schema S is a sentence that is S with each free variable replaced by a constant.

special-purpose knowledge store called a Temporally Labeled Graph (TLG) as described in [8].

We first describe the Reusable Time ontology with an emphasis on its novel features, and then we describe the reasoners developed for the ontology.

4.1. The Reusable Time Ontology

The representation of time is fundamental to any knowledge base that includes representations of change and action. Ontologies providing representations for time are therefore particularly important building blocks for a broad range of knowledge system applications. Although such time ontologies are available from both academic (e.g., Ontolingua [18]) and commercial sources (e.g., Cycorp [21]), they all have significant deficiencies in expressive power, formality, and/or coherence. Our objective here is to provide a time ontology that overcomes those deficiencies and provides knowledge engineers with an effective and easily understandable means of representing knowledge about time. The advantages of a standard time ontology include saving repetitious knowledge building efforts and facilitating potential merging of multiple ontologies built by different authors but using the same time ontology. Our time ontology is targeted toward practical use, and we made many design decisions during its development to facilitate knowledge engineering that we will describe later in this paper.

A temporal ontology is based on a temporal logic. There are many different time theories in the literature [14], and there are difficulties associated with each of them. For example, instant-based time theories ([17], [2]) are not natural for representing events that have a duration, and Allen's interval-based theory has trouble with the Dividing Instant Problem [12]. In [4] and [22], the approach of treating both instants and intervals as independent primitives is presented, which is the approach we adopted. It is of interest to study, as we have here, what kinds of instants (instants of zero duration or instants of unit duration and which unit) and what kinds of intervals (convex or non-convex, open or closed) are appropriate to include in a library ontology.

Since our time ontology must allow knowledge engineers to specify precisely and easily wide-ranging granular systems, different time granularities must be supported in our ontology. Varying granularity does not merely mean that one can use different time units; it involves semantic issues of a layered temporal model and switching from one representation to a coarser or finer one. (See, for example, several formalisms for quantitative and qualitative time granularity in [3] and [9].)

Periodic intervals and calendar dates are also representation problems dealt with in our ontology. In [20] and [7], temporal formalisms provide frameworks for specification of periodic events and operations on them. Calendar dates have been studied more in the temporal database research area. (For example, [19] discusses calendar support for database systems using user-defined calendars.)

4.1.1. Ontology Overview

Our time ontology is implemented in KIF augmented with the frame language specified in the knowledge model of the OKBC (Open Knowledge Base Connectivity) knowledge server API [5]. The ontology is available from the Ontolingua server (http://ontolingua.stanford.edu) in Stanford's Knowledge Systems Laboratory (KSL), and a source file is available at http://ksl.stanford.edu/ontologies/time. The ontology is compatible with the HPKB-Upper-Level-Kernel ontology in the Ontolingua library, which was developed initially in DARPA's High Performance Knowledge Base (HPKB) program [6].

4.1.1.1. Choice of Time Theory

Our ontology is based on the notion of a time line analogous to a continuous number line. Time points and intervals are the temporal entities about which assertions are made. Each time point on the time line is analogous to a real number, and each convex time interval is analogous to an interval on the number line. We also define intervals that are not connected. Taking this approach implies that time is continuous and linear in our ontology. This is an intuitive assumption that is also useful in practice.

4.1.2. Ontology Class Hierarchy Structure

Time-Point and Time-Interval are the two fundamental classes in our ontology. Time-Point is the class of time points on the time line corresponding to real numbers on the number line. "13:00:00 exactly on Jan 3, 1970" is an instance of this class, whose corresponding real number is 219600, assuming "00:00:00 exactly on Jan 1, 1970" is the zero point and second is the unit of measurement.

Class Time-Interval is the class of non-empty sets of time points. Note that instances of Time-Interval do not directly correspond to connected intervals on the number line. "The time when I went jogging this week" is an instance of Time-Interval, but it is not connected. This generalized class makes it much easier to represent events such as "I play tennis every day from 6 to 7 p.m." or "Let's meet every Wednesday ", etc. With only connected time-intervals, we can still find ways to express the above scenarios, but only in awkward ways.

We have made Time-Point and Time-Interval disjoint classes. It would be theoretically sound to let Time-Point be a subclass of Time-Interval where there is only a single time point in the set. That choice, however, leads to a more cumbersome recursive formalization in which the start and end points of an interval are themselves intervals each of which has its own start and end points, etc.

Useful subclasses of Convex-Time-Interval are Calendar-Month and Calendar-Day. (Other subclasses such as Calendar-Year or Calendar-Week can be easily added.) Calendar-Month has 12 subclasses, Calendar-Month-January through Calendar-Month-December, Calendar-Day has subclasses, Calendar-Day-1 through Calendar-Day-31, and Calendar-Sunday through Calendar-Saturday. "January, 1999" is an instance of Calendar-Month-January, etc.

Non-Convex-Time-Interval and Convex-Time-Interval are a disjoint and complete decomposition of Time-Interval. Non-Convex-Time-Interval is the class of time intervals that are not connected, i.e., with "holes" in them.

Class Time-Quantity is a subclass of Physical-Quantity from the library ontology "Physical-Quantities" [16]. A time quantity is an "amount" of time that is represented by a real number and a time unit. Relation `magnitude` from the Physical-Quantities ontology has three arguments: a physical quantity, a unit, and a real number. If Q is a time quantity of 90 minutes, then `(magnitude Q Minute 90)` and `(magnitude Q Hour 1.5)` would both be true. Conversion between different time units can be done using `magnitude`. Adding two time quantities is analogous to adding two real numbers.

4.1.3. Functions and Relations

We defined the following functions on domain Time-Point. Function `location-of` maps a time point to a time quantity (i.e., a duration) that is the amount of time from an arbitrarily selected "point zero" on the time line to the time point. That time quantity locates the time point on the time line and can be used to determine relations between two time points. Functions `year-of`, `month-of`, `day-of` `week-day-of`, `hour-of`, `minute-of`, and `second-of` are functions defined for convenience in terms of the value of `location-of`. In many domains, it is natural to specify time points by year, month, day, etc. In that case, the function value of `location-of` can be calculated from those specified function values. `year-of`, `hour-of`, `minute-of` and `second-of` have range Integer, and `month-of`, `day-of`, and `week-day-of` have range Calendar-Month-Type, Calendar-Day-Type and Calendar-Week-Day-Type, respectively. Calendar-Month-Type is a class of classes whose instances are the 12 subclasses of Calendar-Month. Calendar-Day-Type is also a class of classes whose 31 instances are classes Calendar-Day-1 through Calendar-Day-31. Similarly for Calendar-Week-Day-Type, its instances are classes Calendar-Sunday through Calendar-Saturday.

Three binary relations are defined on Time-Points: `before`, `after`, and `equal-point`, which correspond to "<", ">", and "=" on the number line, respectively.

Three basic functions on domain Time-Interval are `starting-point`, `ending-point`, and `duration` (range Time-Quantity). The starting point of a time interval is the greatest lower bound of the points in the interval, and the ending point is the least upper bound of the points in the interval.

For Convex-Time-Interval, the function value for `duration` is the difference between the `location-of` function values of the interval's starting point and ending point. The value of `duration` of a non-convex time interval is the sum of durations of all convex time intervals contained in the non-convex time interval that have no points in common.

We define the entire set of 13 Allen relations on Time-Interval as well as the relations on Time-Interval defined in the HPKB-Upper-Level-Kernel ontology for compatibility. These definitions are based on comparisons of their starting points and ending points. For example, `Precedes` is defined by:

```
(<=> (precedes ?i ?j)
     (and (Time-Interval ?i)
          (Time-Interval ?j)
          (before (ending-point ?i)
                  (starting-point ?j))))
```

Consider an example of defining "a week in January" using some of the classes and relations. This is a class each of whose instances is a particular week in January of a particular year. The definition would be as follows:

```
(and (subclass-of WiJ Convex-Time-Interval)
     (=> (WiJ ?w)
         (and (duration ?w (the-quantity Day
7))
              (exists (?j)
                (and (Calendar-January ?j)
                     (or (during ?w ?j)
                         (costarts ?w ?j)
                         (cofinishes ?w
?j)))))))
```

Relations `during`, `costarts`, and `cofinishes` are three Allen relations. Function `the-quantity` is the constructor for time quantities. The definition says that each week in January W is a convex time interval whose duration is 7 days and for which there exists a "calendar January" J such that W is during J or W costarts J or W cofinishes J.

4.1.4. Some Ontology Design Issues

4.1.4.1. Time Granularity

In section "Functions and Relations" above, we said that the `location-of` function value of a time point locates this time point on the time line as a real number locates a point on a number line. However, we cannot always identify this real number, i.e., we cannot measure time with definite accuracy. Thus, we address the issue of time granularity in our ontology. When we are describing an ideal physics experiment, time can be specified with perfect accuracy. At other times, "day" would be an appropriate primitive time unit for an American history book, "minute" for a daily class schedule, "microsecond" for measuring CPU time, etc. Since our ontology is intended to be multi-use, it must be able to handle different levels of granularity.

In our ontology, granularity is specified for time points, not for time intervals. A time point with a certain level of granularity is a single time point with the uncertainty that it may be anywhere in a particular time interval. For example, time point "Jan 1st, 2002" with day granularity is a single time point that can be any point within the convex time interval starting midnight of Dec 31st, 2001, and ending midnight of Jan 1st, 2002. For time points with day granularity, the time line, in both directions and starting from "point zero", is evenly divided into time slots of duration one day. The `location-of` function value for a time point only determines the slot in which the time point locates, in much the same way as truncating real numbers to integers, 1.56 to 1, -5.99 to –5, etc.

Function `granularity-of` is defined on domain Time-Point, with range Time-Granularity. Class Time-Granularity is defined in our ontology. There can be an arbitrary number of levels of granularity since every time quantity can correspond to a level of granularity. In practice, we only have granularities for commonly used time units. Year-Granularity, Month-Granularity, Day-Granularity, Hour-Granularity, Minute-Granularity, and Second-Granularity are defined as instances of Time-Granularity in our ontology. Other levels of granularity can be added when we need either a finer granularity than second or a granularity for Day-On-Planet-X of, say, 12.3456 hours. Function `time-unit-of` is defined on domain Time-Granularity with range Time-Unit, a subclass of Unit-Of-Measure in ontology "Physical-Quantities". The function `time-unit-of` maps Year-Granularity into Year, etc. There is a special instance of Granularity, Infinitely-Fine-Granularity, that does not have a `time-unit-of` function value. When users do not care to use granularity, as when describing an ideal physics experiment, all time points are assumed to have infinitely fine granularity.

The three basic binary relations on Time-Points (i.e., `before`, `after`, and `equal-point`) are fundamental to defining any relation on Time-Interval on different levels of granularity. For example, relation `equal-point` is defined as follows. Two time points are equal either when they both have infinitely fine granularity and exactly coincide with each other on the time line, or when they have the same granularity and fall in the same time slot. Two time points on two different levels of granularity cannot be said to be equal to each other. However, we define `before` and `after` to hold for two time points on different levels of granularity when the two time slots containing the time points don't overlap. For example, time point "Year 1970" is before time point "Jan 1st, 1999".

4.1.4.2. Class Calendar-Month

The need to associate attributes with each calendar month leads to using Calendar-January instead of 1 and Calendar-February instead of 2, etc. for the months of the year. These classes in the ontology store many calendar facts about each month, for example its number of days. We use a similar design for Calendar-Day and their subclasses.

As an example of the use of these classes, consider representing the holidays during the year. We could do that by making Holiday a subclass of Regular-Non-Convex-Time-Interval, and define functions `in-month` and `in-day` on domain Holiday with range Calendar-Month-Type and Calendar-Day-Type, respectively. "New Year's Day" would have `in-month` function value Calendar-January and `in-day` function value Calendar-Day-1. We could define a ternary relation `in-week-day` with its three arguments being an instance of Holiday, an instance of Calendar-Week-Day-Type, and an integer, respectively. "Father's Day", which is the second Sunday in June, could then be defined by:

```
(and (Holiday FsD)
     (in-month FsD Calendar-June)
     (in-week-day FsD Calendar-Sunday 2))
```

At the same time, we could define relation `has-holiday` with range Holiday on domain Calendar-Month-Type, Calendar-Day-Type, and Calendar-Week-Day-Type. For example, `has-holiday` would hold for Calendar-January and "New Year's Day". Holiday information can be well integrated into the calendar in this representation.

4.2. The Domain-specific Reasoners for the Reusable Time Ontology

We recently expanded our JTP reasoning system by incorporating the "Reusable Time" ontology and developing domain-specific "telling" and "asking" reasoners for the ontology's temporal relations on time points and time intervals.

We describe the development of those temporal reasoners in this section.

4.2.1. Temporally Labeled Graph

We began by implementing a special-purpose knowledge store called a Temporally Labeled Graph (TLG) as described in [8]. A TLG provides data structures for storing partial ordering relationships among time points and the following set of commands for querying, updating, and retrieving data from the data structures:

`query` – The `query` command takes as inputs two time points, A and B. The command returns the relation between A and B, which can be `before`, `after`, `equal`, `before-or-equal`, `after-or-equal`, `not-equal`, or `any-relation`.

`update` – The `update` command takes as inputs two time points and the relation between them. The TLG is updated appropriately. This command has no return value.

`getAllPointsAfter`, `getAllPointsBefore`, `getAllPointsEqual`, `getAllPointsAfterOrEqual`, `getAllPointsBeforeOrEqual` – These commands take as input a single time point A and return a collection of all known time points that occur in the respective relationship (i.e., after, before, same time as, after or at the same time as, or before or at the same time as) with A.

While this TLG and its commands allow the user to manipulate time points in many desirable ways, it does not allow for the assertion and querying of time intervals. It also is not integrated within a larger system that allows a user to have a KB of which time points are only one component. The JTP architecture allows us to integrate this TLG into a confederation of reasoners so that it can be more generally useful in reasoning tasks.

4.2.2. The TLG as a Knowledge Store

The first step in the integration was to provide wrappers for the TLG to conform to the JTP architecture's requirements for a "knowledge store". The TLG provides a knowledge store for literals whose relation is any of `before`, `after`, `equal`, `before-or-equal`, `after-or-equal`, or `not-equal`, and whose arguments are constants representing time points.

The TLG was augmented to support associating a set of justifications with each asserted relationship between time points, and new commands were implemented for adding and retrieving justifications. The TLG's `update` command was used to implement the command for adding a sentence, and the TLG's `query` and `getAllPoints–` commands were used to implement the command for retrieving sentences.

4.2.3. Reasoners for Time Points

We then implemented reasoners for relations between time points as follows.

4.2.3.1. Time Point Telling Reasoner

This reasoner processes literals (i.e., relational sentences or their negations) being told to the KB whose relation is any of `before`, `after`, `equal-point`, `before-or-equal`, `after-or-equal`, or `not-equal`, and whose arguments are constants representing time points. The reasoner simply stores the literal in the TLG knowledge store using its add sentence command.

4.2.3.2. Time Point Asking Reasoner

This reasoner processes literal schemas whose relation is `before`, `after`, `equal-point`, `before-or-equal`, or `after-or-equal`, or literal schemas whose relation is a variable and whose arguments are constants representing time points. The reasoner uses the retrieve sentences command of the TLG knowledge store to determine answers to the query. The reasoner enumerates a proof for each answer produced by the retrieve command containing appropriate bindings for the variables in the input literal schema.

4.2.4. Reasoners for Time Intervals

To provide reasoners for the Allen relations on time intervals, we used the functions `starting-point` and `ending-point` defined in the Reusable Time ontology.

4.2.4.1. Time Interval Telling

When a sentence using an interval relation is told, the interval relation can be translated into time point relations between the start and end points of the two intervals using axioms from the ontology such as the following:

```
(=> (precedes ?t1 ?t2)
     (before (ending-point ?t1) (starting-
point ?t2)))
```

The JTP library includes a "Horn clause telling reasoner generator" that takes as input a file of multi-literal Horn clauses and produces as output for each Horn clause HCi a Horn clause telling reasoner (HCTR) that uses HCi as a forward-chaining rule. The HCTR for a horn clause of the form "(=> (and LS1 …) LS0)", where each LSj is a literal schema, tells to the KB an instance of LS0 whenever corresponding instances of all of the conjuncts LSj are asserted. The HCTRs produced from axioms in the Reusable Time ontology derive literals relating the start and end points of intervals that can be stored in the TLG knowledge store by the Time Point Telling Reasoner.

For example, if the sentence `(precedes Joe-Birthday Amy-Birthday)` is told, then the sentence `(before (ending-point Joe-Birthday) (starting-point Amy-Birthday))` will be derived and added to the TLG knowledge store.

4.2.4.2. Time Interval Asking

While the translation of interval relations into point relations allows JTP to successfully process queries about time points, including the ones which have been implicitly defined via intervals, we would also like for the system to answer queries about intervals.

The JTP library includes a Generalized Modus Ponens asking reasoner, which is essentially a Model Elimination FOL theorem prover that invokes special-purpose reasoners to derive instances of subgoals, each of which is a literal schema. Adding to the KB sentences from the time ontology that define the point relations in terms of interval relations allows that reasoner to reformulate queries about interval relations in terms of point relations, and the Time Point Asking Reasoner is then able to complete the query using the TLG knowledge store.

As an example, this Horn clause is in the ontology:

```
(=> (before (ending-point ?t1) (starting-
point ?t2))
     (precedes ?t1 ?t2))
```

The point relation literals `(before (ending-point Bob-Vacation) (starting-point Jill-Vacation))` and `(after (starting-point Betty-Vacation) (starting-point Jill-Vacation))` are also in the KB. If the user asks the query `(precedes Bob-Vacation Betty-Vacation)`, the Generalized Modus Ponens reasoner will unify the sentence with the clause above, and JTP will attempt to solve the goal `(before (ending-point Bob-Vacation) (starting-point Betty-Vacation))`. The Time Point Telling Reasoner will then successfully be able to answer this query with the use of the TLG knowledge store.

5. Summary

In this paper, we extended the methodology for developing intelligent systems that promotes building knowledge bases by assembling reusable ontologies and knowledge base modules from on-line libraries to include associating domain-specific reasoners with reusable ontologies and incorporating such reasoners into an intelligent system as part of the knowledge assembly process. The methodology is based on the existing JTP hybrid reasoning architecture that provides an API and functional specification for special-purpose reasoners. We illustrated the methodology by describing a reusable time ontology and an associated set of domain-specific reasoners for the vocabulary of relations defined in that ontology. The time ontology is designed to support large-scale knowledge system applications by providing an easily understandable, flexible, formally defined, and effective means of representing knowledge about time and the standard components and properties of calendars. The domain-specific reasoners support "telling" and "asking" operations for temporal ordering relations on time points and the Allen relations on time intervals.

6. References

[1] James Allen; "Maintaining Knowledge about Temporal Intervals"; Communications of the ACM, 26(11), pp.832-843; November 1983

[2] F. Bacchus, J. Tenenberg, and J. Koomen; "A Non-reified Temporal Logic"; in Proc. KR'89, pages 2-10, 1989.

[3] C. Bettini, X. S. Wang, S. Jajodia; "A General Framework for Time Granularity and its Application to Temporal Reasoning"; Annals of Mathematics and Artificial Intelligence, 1(22):29-58, 1998.

[4] A. Bochman; "Concerted Instance-Interval Temporal Semantics: Temporal Ontologies"; in Notre Dame Journal of Formal Logic, 31(3):403-414, 1990.

[5] V. K. Chaudhri, A. Farquhar, R. Fikes, P. D. Karp, & J. P. Rice; "OKBC: A Programmatic Foundation for Knowledge Base Interoperability"; Proc. of the Fifteenth National Conference on Artificial Intelligence; Madison, Wisconsin; July 26-30, 1998.

[6] P. Cohen, R. Schrag, E. Jones, A. Pease, A. Lin, B. Starr, D. Gunning, and M. Burke; "The DARPA High-Performance Knowledge Bases Project"; AI Magazine, 19(4): Winter 1998, 25-50

[7] D. Cukierman, J. Delgrande; "A Language to Express Time Intervals and Repetition"; Second International Workshop on Temporal Representation and Reasoning; Melbourne Beach, Florida, USA; April 26, 1995.

[8] James Delgrande, Arvind Gupta, and Tim Van Allen; "A Comparison of Point-Based Approaches to Qualitative Temporal Reasoning"; Artificial Intelligence Journal, 131, 1-2, 2001; pp. 135-170.

[9] J. Euzenat; "Granularity in Relational Formalisms"; 1998.

[10] R. Fikes, J. Jenkins, & G. Frank; "JTP: A System Architecture and Component Library for Hybrid Reasoning"; Knowledge Systems Laboratory; KSL-03-01; 2003.

[11] Richard Fikes and Qing Zhou; "A Reusable Time Ontology"; AAAI-2002 Workshop on Ontologies and the Semantic Web; Edmonton, Canada; July 29, 2002.

[12] A. Galton; "A Critical Examination of Allen's Theory of Action and Time"; in Artificial Intelligence, 42:159-188, 1990.

[13] Michael Genesereth and Richard Fikes; "Knowledge Interchange Format, Version 3.0 Reference Manual"; KSL Technical Report 92-86; Knowledge Systems Laboratory, Computer Science Department, Stanford University; Stanford, CA; 1992.

[14] P. Hayes; "A Catalog of Temporal Theories"; Tech report UIUC-BI-AI-96-01, University of Illinois; 1995.

[15] Karen Myers; "Hybrid Reasoning Using Universal Attachment"; AI 67 (1994); pp 329-375.

[16] Ontology Physical-Quantities. Available through http://ontolingua.stanford.edu

[17] Y. Shoham; "Temporal Logics in AI: Semantical and Ontological Considerations"; in Artificial Intelligence, 33:89-104, 1987.

[18] Ontology Simple Time. Available through http://ontolingua.stanford.edu

[19] M. Soo; "Multiple Calendar Support for Conventional Database Management Systems"; Proc. Int. Workshop on an Infrastructure for Temporal Databases, 1993.

[20] P. Terenziani; Reasoning about Periodic Events"; Proc. TIME-95 International Workshop on Temporal Representation and Reasoning, 1995, pp. 137-144.

[21] Time ontology as part of HPKB-Upper-Level-Kernel. Available at http://ontolingua.stanford.edu and at http://www.cyc.com/products2.html#kb

[22] L. Vila, E. Schwalb; "A Theory of Time and Temporal Incidence Based on Instants and Periods"; 1996.