Representing Contextualized Data using Semantic Web Tools

Robert MacGregor
(Information Sciences Institute, University of Southern California, U.S.A.
macgregor@isi.edu)

In-Young Ko
(Information Sciences Institute, University of Southern California, U.S.A.
iko@isi.edu)

Abstract: RDF-based tools promise to provide a base for reasoning about metadata and about situated data—data describing entities situated in time and space—that is superior to alternatives such as relational databases or object-oriented databases. However, essential representational machinery is missing from the current generation of Semantic Web tools and languages. When that machinery is added, the resulting capabilities offer a combination of novelty and flexibility that may usher in a wave of commercial Semantic Web tool-based applications that precedes the true arrival of the Semantic Web. We have constructed a system, the Semantic Engineering Workbench (SEW), that is proficient at managing situated data. Achieving a practical implementation necessitated extending the basic RDF tools (Hewlett-Packard's Jena and Stanford's Protégé) to support *contexts*. In the SEW, a context references a set of statements having common spatial, temporal (and other metadata) attributes. We investigated multiple possible implementations of contexts, and found significant drawbacks in the most common approaches. The clear winners are quads (adding a fourth field of type 'context' to each triple results in a quadruple, or *quad*), and object-oriented contexts (a context mechanism that references individuals instead of statements). Most existing Semantic Web tools (e.g., Jena and Protégé) do not understand contextualized data. For these tools, object-oriented contexts provide an elegant solution. We invented a new semantic primitive, called 'theRealThing', that is a generalization of the 'owl:sameIndividualAs' property. If <e1, theRealThing, r> and <e2, theRealThing, r> hold, then e1 and e2 are distinct resources (having different sets of attributes) that denote the same real-world entity. The SEW uses the 'theRealThing' property to automatically generate abstractions of related sets of resources. Our CHIME visualization tool utilizes the SEW to generate a continuous stream of abstracted entities representing summarizations of spatio-temporally situated entities. CHIME offers a preview of novel capabilities enabled by Semantic Web technology.

Categories: H.4.0 – General Information Systems Applications, H.3 – Information Storage and Retrieval

1 Introduction

The n-Dimensional Information Management project at the University of Southern California's Information Sciences Institute is using Semantic Web tools as the base representation technology for a data visualization project called CHIME.[1] CHIME imports spatio-temporally situated data from multiple data sources, normalizes it, and

[1] www.isi.edu/chime/

displays it in multiple ways—as map overlays, in event maps, and in a contextualized tabular display.

CHIME datasets naturally separate into two classes of data, which intelligence analysts often call "internal" and "external." Internal data is the "ordinary" kind—entities, relationships between entities, and attribute values; external data is metadata such as author, source, observation date, entry date, location, etc. Most information representation systems are very clumsy at representing external data. We would like to claim that RDF [Brickley and Guha 03] is well-adapted for representing external data. Unfortunately, this is not the case—we found it necessary to build a fairly sophisticated representation layer above RDF to achieve a satisfactory match between language and requirements. Our extensions include contexts, and a generalization of the `owl:sameIndividualAs` property.

We use the term *contextualized* to refer to sets of data attributes that vary according to the context in which they are viewed (examples of contextualized data include data that changes across time, or data that changes according to a security setting). Ordinary provenance data (e.g., author, creation date) is not normally contextualized. One of our key findings was that contextualized data is strictly harder to represent than provenance data (this is a practical result, not a theoretical one). We will show how commonly used conventions for representing provenance data fail miserably when representing contextualized data.

In this paper, we will examine various aspects of the RDF language, to see what's useful and where RDF falls short. Contexts are a continuing issue for debate with the RDF community—they are generally regarded as important, if not essential for many applications, but there are many different ways to represent them. Here we will try to separate out some of the good ideas from the bad.

Section 2 provides an example of a query over temporally-situated (contextualized) data. Section 3 examines several different forms of contexts. Section 4 contains a brief advertisement for *quads*. Section 5 introduces the `theRealThing` predicate and the notion of a *snapshot*, and shows how they can be used to define an alternate form of context. Section 6 illustrates how snapshots are used to automatically compute abstractions of entities. Section 7 provides some background on the SEW architecture. Section 8 summarizes our conclusions.

2 An Example of Contextualized Data

One of the CHIME datasets consists of a large XML file containing data about ship sitings. Each top-level XML component describes the location of a ship at a particular time, along with attributes such as what kinds of cargo it contains. Using the map display and a time slider, CHIME makes it easy to see where many different ships are located at any given time. We are working to gradually increase the complexity of queries representable using CHIME, the difficulty being that we want ordinary users to be able to compose the queries. An example of a query that is still a bit beyond us today (but we know how to get there) is the following: "Retrieve freighters that visited Antwerp on April 2003 whose cargo included aluminum pipes." We have submitted a challenge problem to the Semantic Web community for examples of how to phrase this query in an RDF query language in a form that is cognitively palatable, and have not yet received a satisfactory answer. However, if we extend RDF to embrace "quads" and contexts, then the query can be expressed quite succinctly. We will use that as the starting point for our discussion, and then work backwards to RDF.

A quad is a four-tuple of the form <C, S, P, O> where C is a context, and S, P, O, are the RDF subject, predicate, and object fields.[2] Section 3 discusses the semantics of our context 'C.' Here is the query in an RDQL variant that supports a quad syntax instead of a triple syntax:

```
SELECT ?f
WHERE  ((null ?f rdf:type ex:Freighter),
        (null ?c rdf:type ex:Context),
        (?c ?f ex:location ex:antwerp),
        (?c ?f ex:hasCargo ?cargo),
        (?c ?cargo ex:consistsOf ex:AluminumPipe),
        (null ?c ex:beginDate ?begin),
        (null ?c ex:endDate ?end),
        (null ?begin ex:before "May 1 2003"),
        (null ?end ex:after "March 31 2003"))
```

This is actually quite a reasonable query. Its fairly concise, and fairly readable. Unfortunately, there are few Semantic Web systems that implement quads—for many of us in the Semantic Web community, quads are still a wish that has not come true.[3]

The above RDQL query assumes that temporal data describing the time of a ship siting is attached to a context rather than to the ship itself. This is crucial for several reasons. There are many sitings of each ship, so we can't attach all of the temporal data for a ship to a single resource. In our application, we "condition" the source XML data being translated into RDF by creating a new context object for each ship siting, and attaching the temporal data (and other external data) to the context.[4] CHIME contains an n-dimensional filtering mechanism that requires a uniform approach to representing spatial, temporal, and other external data. Our use of contexts as the single point of attachment for external data provides that uniformity.

3 Contexts

The notion of contexts has been around a long time, and there is no consensus on the semantics of a context. Cognitively, a context consists of a set of facts (here, RDF statements) and a description of an environment within which those facts are believed to be true. A context implementation includes some kind of mapping from a context object to the statements in it. There are many ways to define such a mapping. Also, the context points to some form of definition of the "environment." Our system defines the environment by directly attaching assertions to the context.

Quads were invented to make it easy to map from a context to a statement. The meaning of a quad is that the triple represented by arguments two through four belongs to

[2] Some people prefer to write quads with the context field in fourth position.

[3] Intellidimension's RDF Gateway supports quads (www.intellidimension.com).

[4] Location data is also copied to the context object, but for simplicity we are only focusing on the temporal data.

the context referenced by the first argument. Some quads point to triples whose semantics are not context dependent; in that case, we put a null value in the context position. Here is an example of a set of quad statements:

```
[_:cxt1 _:f1 ex:location ex:antwerp]
[_:cxt1 _:f1 ex:hasCargo _:cargo1]
[null _:cxt1 ex:beginDate “April 3 2003”]
[null _:cxt1 ex:endDate “April 4 2003”]
```

These statements assert that, for the time period April 3 to April 4 2003, the facts “the location of f1 is Antwerp” and “f1 has cargo cargo1” are both true.

The sad fact is that quads are not supported by most Semantic Web tools, so we need some other way to map a context to a set of statements. RDF has provided an exceedingly clumsy way to do this, using reified statements. Here is an equivalent same set of statements, expressed as triples using reified statements.

```
[_:st1 rdf:subject _:f1]
[_:st1 rdf:predicate ex:location]
[_:st1 rdf:object ex:antwerp]
[_:st1 rdf:type rdf:Statement]
[_:st2 rdf:subject _:f1]
[_:st2 rdf:predicate ex:hasCargo]
[_:st2 rdf:object _:cargo1]
[_:st2 rdf:type rdf:Statement]
[_:st1 ex:inContext _:cxt1]
[_:st2 ex:inContext _:cxt1]
[_:cxt1 ex:beginDate “April 3 2003”]
[_:cxt1 ex:endDate “April 4 2003”]
```

Pretty hideous, isn’t it. Some RDF proponents argue that reified statements aren’t really so bad, because a system can be built that compresses the storage blow-up that you see here back down to something equivalent to our first set of statements. However, an informal poll has failed to discover a remedy for the significant cognitive overload engendered by the use of reified statements. Here is our original RDQL query, rewritten to execute against triples and reified statements:

```
SELECT ?f
WHERE  ((?f type Freighter),
        (?st1 type Statement),
        (?st1 subject ?f),
        (?st1 predicate location),
        (?st1 object antwerp),
        (?st2 type Statement),
        (?st2 subject ?f),
        (?st2 predicate hasCargo),
        (?st2 object ?cargo),
        (?st3 type Statement),
        (?st3 subject ?cargo),
        (?st3 predicate consistsOf),
        (?st3 object AluminumPipe),
        (?st1 inContext ?c),
        (?st2 inContext ?c),
        (?st3 inContext ?c),
        (?c beginDate ?begin),
        (?c endDate ?end),
        (?begin before "May 1 2003"),
        (?end after "March 31 2003"))
```

This query captures the intended meaning accurately, but it is really quite awful. Not only is it harder to write, and much less readable, but it is likely to be *much* less efficient than the quad representation. Why is that? First of all, the number of "joins" is much larger. Second, and possibly more damaging, the optimizer now has to optimize over predicates like "subject," "predicate" and "object" that mix together extensions of many different predicates.

Before we finish our initial discussion of contexts, we discuss several variations on representing contexts. Each of them has significant drawbacks:

Some folks have advocated using resources of type `rdf:Bag` to point to statements in a context. In place of our "`inContext`" triples, one writes (reversing the arguments):

```
[_:cxt1 rdf:_1 _:st1]
[_:cxt1 rdf:_2 _:st2]
```

A query that interprets bags (instead of using a property such as `ex:Context`) becomes even less readable, because its necessary to substitute a null predicate instead of "`ex:inContext`" to match a bag to its members. It is hard to imagine why anyone would want to use bags to solve this problem.

Recently, lists were added to RDF. If we use lists instead of bags, we can't write a query anymore in RDQL, because RDF does not provide a list membership predicate. Also, the list implementation uses double the storage of bags (in terms of number of edges) and can no longer provide a constant time membership test.

Another possibility is to eliminate contexts altogether, and attach the metadata (the context definitions) directly to the reified statement resources. If the average context contains fewer than two statements (probably not the norm), this saves space. However, with this scheme you have lost a capability for "context switching"—metadata is not grouped into convenient subgraphs. This scheme also misses out on the opportunity for a uniform treatment of contextualized data. When a context class is defined, one is advised to identify a set of predicates that provide a standard means for representing the most commonly-occurring types of metadata. For example, our SEW (see Section 6) adopts a representation for "`beginDate`" and "`endDate`" that is independent of any representational scheme adopted by statements within a context. It does the same for latitude and longitude—it copies positional information from internal data to the context (making it part of the external data). This uniformity makes it easy to optimize spatio-temporal filtering on contexts. The same goes for security information, source information, etc.

Finally, some folks advocate using models as if they were contexts. Whether this is viable or not depends on several factors. Most RDF engines are not equipped to handle large numbers of models per query. If contexts are very coarse grained (relatively few contexts and many statements per context), then this might work out. We envision that applications that make serious use of contextualized data will want their attachments to be more fine-grained than that. Our CHIME application has hundreds of contexts per model, and will later on support thousands (or more) of contexts per model. Translating that into RDQL yields FROM statements that contain hundreds or thousands of URIs. We doubt if many RDF query systems are tuned to handle those kinds of numbers. Also, we are still waiting for someone to tell me what kind of query syntax that RDF systems use with this kind of context mechanism.

Several different knowledge representation systems (e.g., Loom [Loom 03], Epikit [Genesereth 92], CycL [Lenat and Guha 90], PowerLoom [Loom 03]) implement contexts (CycL calls them "microtheories"), and manage them using a model-as-context kind of semantics. All of these systems support quantification over contexts/models, and they assume that contexts can be arranged in a hierarchy, so that the truth of statements belonging to a context inherit to its child contexts. The contexts in these systems tend to be relatively coarse-grained, and are not particularly well-suited to representing provenance and temporal data. Each of these languages adopts an "`isT`" (is true in context) predicate to relate a context to a statement. When applied to a binary relation, the "`isT`" syntax is some variation of

```
    "(isT <context> <predicate> <subject> <object>),"
```

which is isomorphic to a quad representation. Primary uses of this kind of context are (i) building up hierarchies of models/contexts, where lower level ones inherit statements in higher ones, and (ii) representing and reasoning about hypothetical worlds. From a usage standpoint, these contexts are somewhat orthogonal to the kind of contexts that we are discussing for RDF, and the fact that the term "context" is used for both is unfortunate. They are not quite apples and oranges, but they might as well be.

Summarizing, we began with an example of a contextualized query that used quads as the means for relating contexts to statements, and the result was relatively concise and straightforward. We showed what an equivalent query looked like using one strategy for mapping contexts to reified statements, and the result was pretty horrendous. Then we explored a few alternative strategies also based on reified statements, and they all had additional drawbacks. We conclude that the representational strategies based on reified statements are demonstrably inferior to a quad strategy, and more simply, they are just plain inferior. We are not aware (before RDF) of any significant KR technology making significant use of reified statements (we added them to Loom, and the first author wrote a monograph on reified statements [MacGregor 93], but the ultimate conclusion was that they added significant complexity and awkwardness, while yielding relatively little value). They are really just an unfortunate mistake by the original RDF designers that is hindering the development of Semantic Web technology.

4 Quads

A common argument against quads goes "We have triples, you want quads, where is it going to stop? Quintuples? Sextuples?" The answer is, quadruples is all you need (this is a well-educated guess). Some RDF systems (Jena's RDB models is an example [Jena 03]) internally implement a quad structure that adds a model column to the subject/predicate/object columns. This allows them to map a model to a set of statements. It might seem that adding contexts to their quads would turn their quads into quintuples. Although one *could* add a fifth context column, a better solution is to convert the models column to a context column and adopt the convention that each context belongs to exactly one model. That way, we have quads, and we can also directly map each statement to a model through its associated context. In other words, you can convert a triples-plus-model architecture to a quad architecture with no significant increase in storage requirements.

5 Snapshots

The original query that we posed, "retrieve freighters that ..." is problematic because it returns a set of resources of type `ex:Freighter`, but omits any connection between those resources and the statements that support the conclusion (that each freighter visited Antwerp sometime in April, etc.). There is a simple solution—just add a context variable to the SELECT clause, yielding the query:

```
SELECT ?f, ?c
WHERE  ((null ?f rdf:type ex:Freighter),
        (null ?c rdf:type ex:Context), ...
```

This query returns Freighter resources paired with contexts that can be used to index to the statements that hold at the time specified in the query. Note, however, that your application must be able to correctly interpret freighter/context pairs.

While it may become commonplace in the future for tools and applications to fluently manipulate contextualized data, this is not the case today. Many (or most) RDF tools are not equipped to reason with contexts. For example, the Protégé system [Protégé 03] used within the SEW is clueless in this regard. If Protégé were used in conjunction with any of the reified-statement based schemes we've outlined thus far, it would show all of the statements for a contextualized resource jumbled together. Furthermore a scheme based on reified statements breaks down completely if an RDF tool enforces single-valuedness constraints on temporally-sensitive edges like latitude and longitude (each freighter will have many latitude attributes and many longitude attributes). Our project encountered this problem when we first considered using reified statements—this was a major stimulus for searching out alternate solutions.

At this point in our discussion, we have uncovered a practical drawback to the quad approach, and our reified statement schemes have become untenable, so we need to look for something else. The solution we came up with involves creating a new predicate we call "`theRealThing`". If five different temporal sitings are recorded for a freighter `ex:f1`, we create six resources of type `ex:Freighter`, one denoting the actual freighter, and five that represent "views" of ex:f1 at each of the five different temporal contexts. We call each of these five views "snapshots"—conceptually, each represents a snapshot of the state of `ex:f1` at a different point in time. Each snapshot is a resource that is related to `ex:f1` by the `ex:theRealThing` property. The meaning of a statement

```
[_:ss1 ex:theRealThing ex:thing1]
```

is that the real world entity that fills the "role" denoted by `_:ss1` is the same entity as that denoted by `ex:thing1`. As another example of the semantics of the `ex:theRealThing` property, consider a person Sue who works as a CFO on weekdays, and as a camp counselor on weekends. Sue is filling two different employee roles, and these employees are distinct entities. If we were to define the relationship between the person Sue and the two employees Sue-as-CFO and Sue-as-camp-counselor using `owl:sameIndividualAs`, the statements that applied to each of the three Sues would get all jumbled together. Relating them using `ex:theRealThing` enables us to infer that the same individual fills all three roles in real life, but the roles remain distinct within our RDF database. The `ex:theRealThing` property generalizes (is a super property of) `owl:sameIndividualAs`. `ex:theRealThing` is transitive and non-symmetric. Statements inherit backwards across a `ex:theRealThing` relationship. For example, if the statements

```
[_:sscfo ex:theRealThing ex:Sue]
[ex:Sue rdf:Type ex:Person]
[_:sscfo rdf:type ex:CFO]
```

are true, then

```
[_:ss1 rdf:Type ex:Person]
```

is also true.

Returning to our ships example, a set of statements for a snapshot `_:ssf1` of the freighter `ex:f1` might look like:

```
[_:ssf1 ex:theRealThing ex:f1]
[_:ssf1 ex:latitude 222]
[_:ssf1 ex:longitude 333]
[_:ssf1 ex:hasCargo _:sscargo1]
[_:sscargo1 ex:hasCargo _:cargo1]
[_:sscargo1 ex:consistsOf ex:FrozenBagels]
[_:ss1 ex:inContext _:cxt2]
[_:cargo1 ex:inContext _:cxt2]
[_:cxt2 ex:beginDate “March 12 2003”]
[_cxt2 ex:endDate “March 12 2003”]
```

This example is doing something we haven’t seen before—it maps the context `_:cxt2` to two snapshots, `_:ssf1` and `_:sscargo1`, that are not statements, i.e., we are now defining a context by referring to a set of snapshots instead of a set of statements. This notion of context, which we call an “object-centered context,” is somewhat radical (we have not encountered it in the literature, although it might be there), convenient, and as we shall see later, enables us to do some clever things with abstraction. The immediately obvious advantages are (i) everything can be represented using triples, with no use of reified statements, and (ii) tools like Protégé that don’t understand contexts have no trouble viewing and editing snapshots.

Our object-centered contexts are not as radical as they might seem, because they are formally defined as sets of statements—an object-centered context `_:cxt` contains all statements of the form `[_:ss _:pp _:oo]` where `[_:ss ex:inContext _:cxt]` is true. In other words, if SS is the set of snapshots related to `_:cxt` by the ex:inContext property, then `_:cxt` consists of all statements having subjects that belong to SS.

Here is our original query, rephrased to use object-centered contexts:

```
SELECT ?f
WHERE  ((?ssf rdf:type ex:Freighter),
        (?ssf ex:location ex:antwerp),
        (?ssf ex:hasCargo ?sscargo),
        (?sscargo ex:consistsOf ex:AluminumPipe),
        (?ssf ex:inContext ?c),
        (?sscargo ex:inContext ?c),
        (?c ex:beginDate ?begin),
        (?c ex:endDate ?end),
        (?begin ex:before "May 1 2003"),
        (?end ex:after "March 31 2003"))
```

In this query all quads have become triples, and two additional `ex:inContext` clauses map the snapshot variables to the context variable.

The alert reader will have noticed that all references to quads have been eliminated. A good question is, "If we elect to use snapshots, does that mean that quads are unnecessary?" At the moment, we don't have a definitive answer. Snapshots are very handy for modeling contextualized objects, e.g., temporally situated objects like our freighters and cargo, or role-playing objects like our hardworking Sue. Such things as security tags would appear to also fall under the class of contextualized data (more privileged applications will see more statements attached to a given resource than less privileged applications, and may offer higher precision data values). However, snapshots are unnecessary for modeling ordinary provenance information (author, creator, etc.). The question of whether or not to use both quads and snapshots might depend on the ratio of contextualized data to provenance data. We would need a lot more experience to make this call.

A second reason to have both quads and snapshots is that because quads directly specify the mapping from context to statement, while object-centered contexts map to statements indirectly (through snapshots), there is a possibility that quads provide a more fine-grained control of one's model. We haven't run across an example of this, but we haven't yet tried to prove that object-centered contexts can represent anything that ordinary contexts can.

At present, we are not using quads within the SEW, because we don't have access to a quad store. However, we hope to do such a conversion some time in the future, while retaining our snapshots capability.

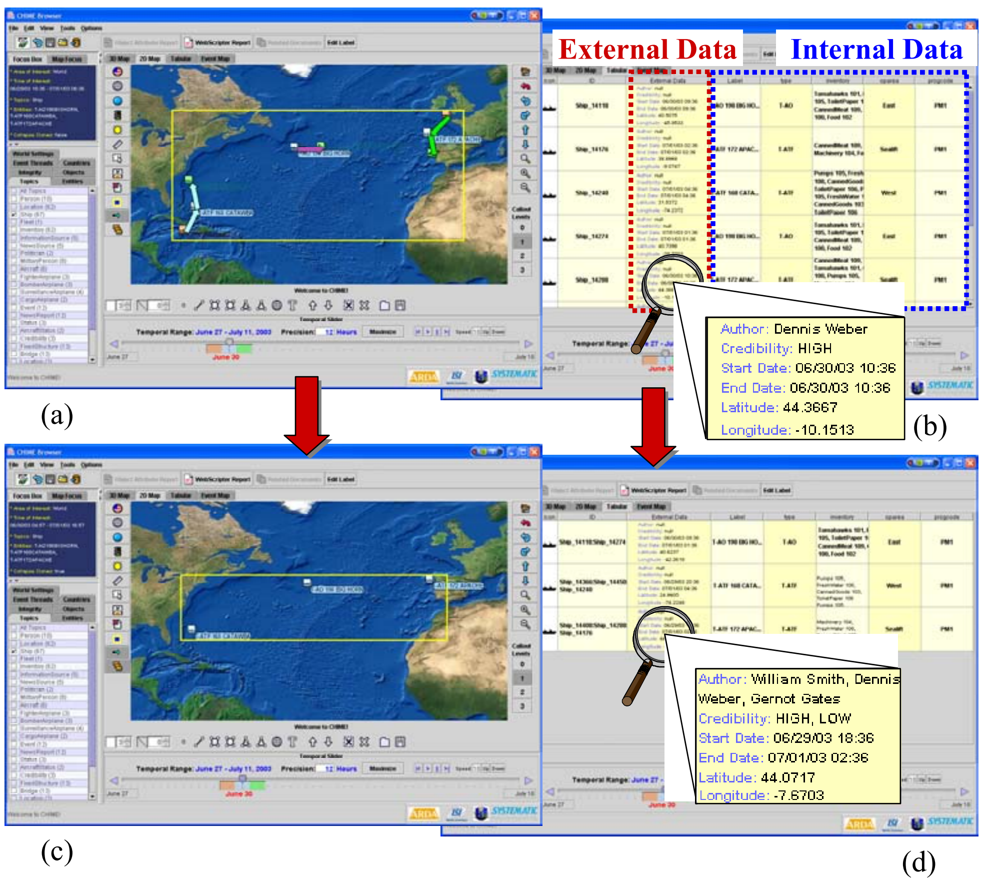
Semantic note: [Hirschman et al., 98] discuss the problem of co-reference semantics in the presence of change over time. To properly handle temporal change, they revised the semantics of the MUC-7 Co-reference Task definition from using the identity relation for co-reference to using something more general. They formally defined the difference by distinguishing between extensional and intensional mentions of entities.[5] The example in their paper mentions a single intensional entity (the CEO of a company) being filled by two individuals. That is different from our individual Sue filling two distinct intensional roles, CEO and camp counselor. It's not clear if their new co-reference relation (which didn't have a name in that paper) can properly treat the kinds of co-reference we are dealing with. A virtue of our `theRealThing` property is that it is semantically weaker, and therefore more broadly applicable, than other identity relations that we have come across.

6 Aggregated Snapshots

CHIME is capable of dynamically synthesizing new contexts, and creating new snapshots that represent aggregations of other snapshots. We call a set of snapshots that point to the same real thing "siblings." For example, a set of resources/snapshots that track the

[5] Distinguishing between intension and extension was especially popular in the early days of description logics.

locations of a single moving ship are siblings (of each other). Suppose that each of the ships in a set of siblings `{ss1, ss2, ... }` is tied to a different temporal interval. We can define a context defined for a time interval that spans all of these individual temporal intervals, and then ask CHIME to compute a snapshot `ssA` representing the aggregation of `{ss1, ss2, ...}` with respect to that new context. Each attribute of this abstracted ship is computed by aggregating the values of the corresponding attribute for each of `ss1, ss2, ...`. For example, the latitude of `ssA` is the average of the latitudes of `ss1, ss2, ...`. If the ship siblings each have an attribute "`position`" whose value is a rectangular bounding box, ssA will have a position whose value is the smallest rectangle that contains all of the other rectangles. The value of the (multi-valued) "`hasCargo`" attribute for ss1 is union of "`hasCargo`" attributes of all of the siblings. The abstraction operation is recursive—each of the aggregated cargo snapshots may itself be an aggregated snapshot.



Author: Dennis Weber
Credibility: HIGH
Start Date: 06/30/03 10:36
End Date: 06/30/03 10:36
Latitude: 44.3667
Longitude: -10.1513

Author: William Smith, Dennis Weber, Gernot Gates
Credibility: HIGH, LOW
Start Date: 06/29/03 18:36
End Date: 07/01/03 02:36
Latitude: 44.0717
Longitude: -7.6703

Figure 1: Aggregation of snapshots presented in CHIME's map and tabular viewers— (a) 8 snapshots of 3 different ships are plotted on a map with their moving trails; (b) detail properties (dates, coordinates, inventories, etc.) of the 8 snapshots are listed in a tabular viewer; (c) 3 aggregated snapshots are plotted at abstracted locations (center positions in bounding boxes of siblings) on the map; (d) abstracted properties (merged temporal periods, unions of inventories, etc.) of the aggregated snapshots are listed at the tabular viewer

The computation that computes an aggregate attribute value is specific to each RDF property. Default computations have been built-in for many properties (for example, CHIME defines built-ins for many spatial and temporal properties. Aggregate computations that override the default can be defined for any property. In the absence of a more specific computation, the default computation for a multi-valued property returns the union of the attribute values attached to each of the siblings.

Aggregation is not limited to the temporal dimension. CHIME contexts are logically organized into a subsumption hierarchy computed along multiple dimensions, including time, space, topic, credibility, and source. The set of snapshots belonging to a context defined to subsume other contexts is computed by retrieving all sets of siblings belonging to the subcontexts, and computing an aggregated snapshot for each of these sets.

Figure 1 illustrates visually the effect of aggregating snapshots.

7 Semantic Engineering Workbench and CHIME Architecture

The Semantic Engineering Workbench (SEW) provides an intelligent infrastructure for managing Semantic Web databases and developing Semantic Web applications. The SEW has been crafted by integrating key (open-source) software components into an integral whole. Retrieval capabilities and persistence is provided by combining Hewlett-Packard's Jena triple store with a relational database (we are currently using MySQL). Ontology editing is provided by Stanford's Protégé Knowledge Acquisition tool. The SEW implements several layers of API's. The highest levels provide object-oriented representations of data objects, while lower-levels enable access to triples. The SEW transparently converts triples retrieved from Jena into Protégé objects, using an on-demand strategy that imports data on an as-requested basis. The SEW is wholly implemented in Java, and currently runs on Windows PCs.

The CHIME's visualization system is tightly coupled to the SEW architecture. CHIME implements an "n-dimensional" filter that makes it easy for users to define (and bookmark) combinations of attributes that collectively define filters on underlying databases. CHIME is optimized for display of spatio-temporal data, but the filtering mechanisms work equally well for ordinary data, displayed in tabular form. The SEW provides specialized support representing contextualized or abstracted entities (e.g., snapshots of an entity moving through space-time). CHIME implements several perspectives for viewing these entities.

A pattern match or query submitted to the SEW is distributed to all currently open models, which may reside on different backend host machines. Large, frequently accessed models should be stored in a Jena-managed relational database. The SEW also knows how to import (and save) data from XML, RDF, and N3 files. When retrieving data from Jena into Protégé, the SEW converts all URI's into qualified names of the form `namespace:localName`, thereby making the data much more readable by humans. The Protégé editor can be used to assign meaningful labels to namespaces.

The CHIME system is loosely-coupled to the WebScripter report generator [Frank et al., 02] and to a collaborative semantic annotation tool that supports shared viewing of semantically-marked up Web pages. These tools are both research projects. Figure 2 illustrates the combined architecture.[6]

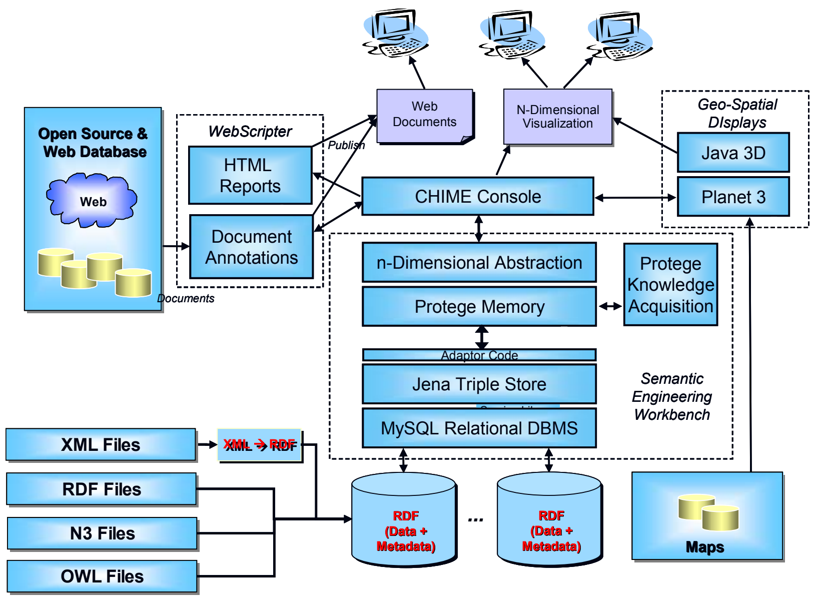


Figure 2: SEW's architecture and its connections with CHIME components and data sources

Summary of current SEW features:

- Persistent storage of large numbers of facts (RDF triples)
- Convenient and powerful ontology editing
- Multi-modal viewing of data
- N-dimensional data filtering
- Automated data conversion (e.g., XML-to-RDF)
- Intelligent management of models and namespaces
- Transparent access to distributed models/databases
- Specialized support for contexts and provenance data
- Specialized support for abstracted entities

[6] The Planet 3 Geo-Spatial display system is proprietary software, owned by Integrity Applications, Inc.

Upgrades projected for the upcoming year include converting the SEW from a triple server to a quad server, enhancing its performance, adding a user-friendly authoring capability, adding a lock mechanism to safeguard multi-user access to shared databases, and enhanced support for identity relationships.

8 Conclusion

Representing contextualized data poses a set of challenges that are not addressed by current RDF technology. To view and manipulate data from different viewpoints (different points in time, different security levels, etc.) requires that some kind of grouping mechanism be established that specifies (i) the definition of the viewpoint, and (ii) what statements should be visible from that viewpoint. Contexts provide such a grouping mechanism. Today, a number of Semantic Web researchers appear to be representing provenance information in RDF using ad hoc strategies that do not include a grouping mechanism. This is short-sighted, and will get them into trouble if they attempt to apply these same ad hoc techniques to contextualized data, rather than to simple kinds of provenance data.

This paper has surveyed a variety of context mechanisms, and found significant drawbacks to most of them. Two approaches, one using quads, and one using object-oriented contexts, demonstrate that cognitively attractive solutions do exist. All of the approaches based on the use of reified statements suffer by comparison—the primary effect of the RDF statement mechanism would seem to be to retard the growth of Semantic Web technology. Unfortunately, neither of our “winners” represents a small step beyond the current RDF language specification. Architecturally, extending a triple store to become a quad store isn’t that difficult. We have hinted at how such an extension can be made without incurring additional storage overhead. The sociological hurdle to switch from triples to quads is higher than the technological one.

We presented a new primitive, “`theRealThing`,” and snapshot mechanism that solve a number of intertwined problems relating to the representation of contextualized entities, role-playing individuals, and aggregate views of entities. By embedding these new capabilities into our server’s API, all of our applications benefit from these new capabilities. From a more academic perspective, these capabilities remind us that RDF is still a very coarse representation language, and that more subtle semantic structures (e.g., roles [Franconi and Rabito 94], qua-links [Freeman 81], aggregation [Patil et al., 81]) are still beyond reach of the W3C radar. For representing co-reference relations, our `theRealThing` property exhibits certain advantages over the `owl:sameIndividualAs` property that suggest that the latter property may not be the panacea that some users seem to think it is.

The current W3C RDF committee has treated the original RDF specification with an undue amount of reverence, failing to distinguish between its many good ideas and the few unfortunate ones. We are hoping that some of the RDF tool providers will take a bolder stance, by incorporating capabilities that enable us to reason more fluently with contextualized and aggregate data.

Acknowledgement

Effort sponsored by the Defense Advanced Research Projects Agency (DARPA) under agreement number F30602-00-2-0576, and the Advanced Research and Development Activity (ARDA) under contract number NMA401-02-1-2019. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, ARDA, the National Imagery and Mapping Agency or the U.S. Government.

References

[Brickley and Guha 03] Brickley, D., Guha, R.V. eds.: RDF Vocabulary Description Language 1.0: RDF Schema. W3C Working Draft, www.w3.org/TR/rdf-schema (January 2003).

[Franconi and Rabito 94] Franconi, E., Rabito, V.: A Relation-Based Description Logic. Proc. Int'l Workshop on Description Logics (DL'94), Bonn, Germany (May 1994).

[Frank et al., 02] Frank, M., Szekely, P., Neches, R., Yan, B., Lopez, J.: Webscripter: World-wide grass- roots ontology translation via implicit end-user alignment. Proc. WWW-2002 Semantic Web Workshop, Honolulu, Hawaii (May 2002).

[Freeman 81] Freeman, M.W.: The QUA Link. (J. G. Schmolze and R. J. Brachman, eds.) Proc. 1981 KL-ONE Workshop, Jackson, New Hampshire (1981) 54–64.

[Genesereth 92] Genesereth, M. R., eds.: The Epikit manual. Palo Alto, CA, Epistmemics, Inc. (1992).

[Hirschman et al., 98] Hirschman, L., Robinson, P., Burger, J., Vilain, M.: Automatic Co-reference: The Role of Annotated Training Data, Proc. AAAI 98 Spring Sym. on Applying Machine Learning to Discourse Processing, Stanford University, California (March 1998).

[Jena 03] The Jena toolkit. HP Labs Semantic Web activity, Hewlett-Packard Company, www.hpl.hp.com/semweb/ (May 2003).

[Lenat and Guha 90] Lenat, D.B., Guha, R.V.: *Building Large Knowledge-based systems: Representation and Inference in the Cyc Project*. Addison-Wesley, 1990.

[Loom 03] Loom and PowerLoom. Intelligent Systems Division, Information Sciences Institute, University of Southern California, www.isi.edu/isd/LOOM/LOOM-HOME.html (May 2003).

[MacGregor 93] MacGregor, R.: Representing Reified Relations in Loom, *Journal of Experimental and Theoretical Artificial Intelligence*, 5 (1993) 179-183.

[Patil et al., 81] Patil, R.S., Szolovits, P., Schwartz, W.B.: Causal Understanding of Patient Illness in Medical Diagnosis. Proc. 7th Int'l Joint Conf. on Artificial Intelligence (IJCAI), Vancouver, British Columbia (August 1981) 893–899.

[Protégé 03] The Protégé Project. Stanford Medical Informatics, The Stanford University School of Medicine, protege.stanford.edu (May 2003).