A Reasoner for the Web

Tim Berners-Lee

Vladimir Kolovski

Dan Connolly

James Hendler

Vladimir Kolovski

Yoseph Scharf

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

Cwm (pronounced coom) is a general-purpose data processor for the Semantic Web, for RDF somewhat what sed, awk, etc. are for text files or XSLT is for XML. It is a forward chaining reasoner that can be used for querying, checking, transforming and filtering information. Its core language is RDF, extended to include rules, and it uses RDF/XML or RDF/N3 serializations as required. Cwm's logic and functions are both specially adapeted to be used effectively in the environment of an unbounded webb of data and rules of diverse provenance. We discuss those features and give the experience gained in the design and use of this reasoner for the Semantic Web.

1 Introduction: Motivation and Goals

The popularity of the Semantic Web is ever-increasing, one of the main reasons being the improving tool support for its foundational languages (The Resource Description Framework (RDF)(Klyne and Carroll 2004) and Web Ontology Language (OWL)(Dean and Schreiber 2004)). This tool support usually comes in the form of ontology editors and reasoners for OWL[1]. However, for the general purpose manipulation of data, a rule-based engine makes intuitive things which are unintuitive, difficult or sometimes impossible using Description Logics. The requirements for such a rule engine were twofold: to be a general tool to allow Semantic Web based systems to be built, so as to provide feedback on the overall architecture, and provide an on-ramp for newcomers; and to provide a platform on which to build more sophisticated applications to research future directions for Semantic Web design.

In the context of the Semantic Web, RDF is the lingua franca for representing machine-processable documents. RDF uses URIs as symbols. The *Web* in Semantic *Web* comes from the fact that from your document you can link to just about

[1] Because of its clear and concise semantics, OWL-DL can boast a large number of reasoners.

any web resource you like. However, the Web contains many sources of information, with different characteristics and relationships to any given reader. Whereas a closed system may be built based on a single knowledge base of believed facts, an open web-based system exists in an unbounded sea of interconnected information resources. This requires that an agent be aware of the provenance of information, and responsible for its disposition. A tool in this type of environment should have the ability to determine what document or message said what and match graphs against variable graphs.

The goal of the Semantic Web is the beneficial reuse of data in unplanned ways. (@@this is discussed in the N3 section)

We introduce a tool called Cwm[2] motivated by the above requirements. Cwm is a general-purpose data processor for the Semantic Web, analogous like sed, awk, etc. for text files or XSLT for XML. It is a forward chaining reasoner which can be used for querying, checking, transforming and filtering information. Since RDF itself is not sufficient to express rules, Cwm is based on an extension of RDF called Notation3 (N3) which has support for quoting (making statements about statements), variables and rules.

Being based on a more expressive logic language added a host of features to Cwm not available to other RDF processing tools: possibility of filtering RDF graphs after merging them, for example. Since N3 is expressive enough so that positive datalog-like rules can be expressed in it, cwm is able to reason using a first order logic but without classical negation. Combining this reasoning functionality with its ability to retrieve documents form the web as needed, the system can be considered a reasoner for the web. It has grown from a proof of concept application to a popular rule engine, used in major research projects. The goal of this paper is to explain the functionality and distinguishing features of this tool in more detail.

The second goal cwm, as a platform for research, was in a way to serve as a experimental ground for more expressive Semantic Web applications (which cover some of the upper layers of the Semantic Web layer cake). To this end, we have added *proof* generation/checking support to the system, and also a crypto module which empowers Cwm users with the tools to build *trust* systems on the Semantic Web.

In the next section there will be a brief introduction to N3, mainly describing how it extends RDF. Then, we will briefly explain the system architecture of cwm. From that point on, we will talk about the distinguishing features of the tool, such as its builtin functions, proof generation and proof checking. Cwm has been used as the rule engine in research projects (Policy Aware Web) and for automating tasks at the W3C - we will cover that in Applications, and wrap up with Future Work and Conclusions.

[2] originally from Closed World Machine, now a misnomer, as cwm explicitly deals with open world.

2 Preliminaries: N3

Notation3 (N3) is a language which is a compact and readable alternative to RDF's XML syntax, but also is extended to allow greater expressiveness. It has two important subsets, one of which is RDF 1.0 equivalent, and one of which is RDF extended with rules. In this section, we will briefly introduce N3 show how it is different from RDF.

N3 syntax is very readable, using the *subject predicate object* form. E.g.:

```
:John :parent :Bob.
```

The colon form of qname is similar to XML's qname form. In this paper, for the sake of space, we do not define the namespaces to which prefxes refer. Blank nodes (bnodes, RDF's form of existential variable) are expressed using '[]', for example: `[ :brother :John ]` specifies something that has `:brother :John`.

An N3 rule is written as a statement whose predicate is `log:implies`, for example:

```
{ ?x parent ?y. ?y sister ?z } log:implies { ?x aunt ?z }.
```

Set of statements surrounded by braces is called an N3 formula. Formula is a generalization of an RDF-graph, adding variables.

N3 rules are at the same time more and less expressive than Datalog. On one hand, N3 is restricted to binary predicates (triples), but on the other, existentials (bnodes) are allowed in the consequents of the rules. Another restriction worth mentioning is that, since monotonicity is one of the design goals for N3, there is no negation-as-failure. The only form of negation allowed in N3 is scoped negation as failure, which will be described in section 4.8.

One of the design goals of N3 is that information, such as but not limited to rules, which requires greater expresive power than RDF, should be sharable in the same way as RDF can be shared. This means that one person should be able to express knowledge in N3 for a certain purpose, and later independently someone else reuse that knowledge for a different unforeseen purpose. As the context of the later use is unknown, this prevents us from making implicit closed assumptions about the total set of knowledge in the system as a whole. As a result of this, N3 does not make the Closed World Assumption in general.

3 Architecture

Figure 1 represents a high-level architecture diagram of Cwm. In the following subsections, we will expand on the most important sections of the diagram.

3.1 Parsing and Serialization

The N3 and RDF languages are grammatically similar, to the extent that it is possible to translate one into the other in a streaming mode. The efficiencies evident from this mode of operation led to a design of an abstract syntax interface which

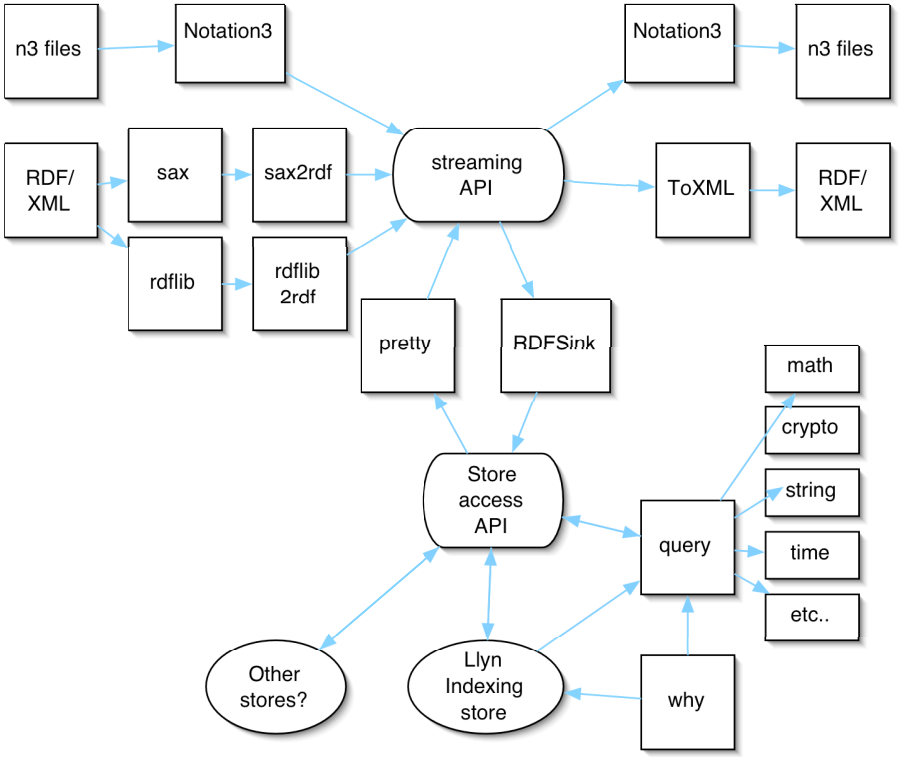

Fig. 1. cwm software architecture

parsers and serializers supported. This is still in use, but experience was that as the N3 language became more sophisticated, the streaming interface became more burdensome to support as an output format for the serialization module (`pretty.py`).

3.2 Store

The store (`llyn.py`) is a triple store, extended to cover the full N3 language, and also to record the provenance of each statement. A statement therefore is stored as the subject, predicate and object of RDF, plus the '*context*' the identity of the formula in which the statement is, and a record of the *reason* object which stores the reason for the statement having been added to the store. This is used in proof processing.

The store was originally designed and built with four indices, so that list was kept of each statement in which a given term occurred as, respectively, the subject, predicate, object or context. When there was a need to improve performance, this was changed so than now 7 indices are used, as almost every combination of subject , predicate, object pattern with wildcards are indexed. This is a questionable tradeoff, as many tasks involve the input and output of data with relatively little processing, so the indexing overhead may dominate.

3.3 Inference engine

The inference engine is essentially a simple forward chaining reasoner for N3 rules.

The matching engine which runs the rules operates simply by recursive search, with some optimizations. Firstly, the rule set is analyzed to determine which rules can possible affect the output of some other rules. A partial order is found, which for example in some cases will produce a pipeline. Otherwise, where rules can interact, they are tried repeatedly until no further rule firings occur. The second optimization is that, when matching a graph template, which is a series of template statements, the statements are ordered for processing as a function of the length of the index which would have to be searched, doing the smallest indexes first. This provides a significant improvement in many real-world examples where the data is very asymmetrical, in that some areas of the graph are dense and tabular in form, and others sparsely connected.

The inference engine also performs two extensions to its normal role, these being the execution of built-in functions, and the delegation of parts of the query to remote systems or remote documents.

4 Features

4.1 Filtering

Filtering in cwm (using –filter) is a convenient way to reduce the data being output. When a filter runs *only* the information gathered by the rules is preserved: everything else is discarded. We use a filter to select the logical relationships that we want from the mass of what is already known. For an example, let's say that for a particular genealogical knowledge base, we only want to know whether Joe and Bob are related. Thus, in addition to the data file (say, family.n3) that holds all of the family facts and the rules (uncle, cousin,sibling etc.) we wil add a filter file (uncleFilter.n3) that only looks for the facts we are interested in:

```
# What is the relationship between Joe and Bob
{ :Joe ?p :Bob } log:implies { ?p a :RelationshipBetweeJoeAndBob }.

# Is Bob an Uncle of Joe?
{ :Joe :uncle :Bob } log:implies { :Joe :uncle :Bob }.
```

When we ask cwm to consider the implication it concludes:

```
> python cwm.py family.n3 --think --filter=uncleFilter.n3
    :Joe     :uncle :Bob .
    :uncle     a :RelationshipBetweeJoeAndBob .
```

The command line can be read as: *read family.n3 and the deduce any new information you can given any rules you have. Now just tell me the information selected by the filter uncleFilter.n3.* Note that any data in the filter is not used - the filter file is only searched for rules.

4.2 Deltas

As the Semantic Web is built, using RDF graphs(Klyne and Carroll 2004) to represent data such as bibliographies(Beckett et al. 2002), syndication summaries(Beged-Dov et al. 2000) and medical terminology(Golbeck et al. 2003),a need arises for difference and sum functions for RDF graphs. The problem of updating and synchronizing data in the Semantic Web motivates an analog to text diffs for RDF graphs. An accompanying delta.py program will compare graphs and generate difference files (deltas). [REF rdf diff @@unpublished] addresses the problem of comparing two RDF graphs, generating a set of differences, and updating a graph from a set of differences. The paper discusses two forms of difference information, the context-sensitive "weak" patch, and the context-free "strong" patch. It also gives a proposed update ontology for patch files for RDF.

4.3 Built-in functions

Many reasoning engines have an ability to perform arithmetic, but arithmetic facts separated out from the facts of the knowledge base as being fundamentally different things. In cwm, this is not the case: arithmetic facts, as with all other facts where the validity can be checked or the result evaluated by machine, are represented as RDF properties. This is done for various reasons. Philosophically, the design was not prepared to commit to a partitioning of knowledge into two parts in this fashion. It was felt necessary to be able to reason about these functions as well as to evaluate them. We understand that this causes problems for a Derscription Logic based systems, but this is not a DL system. it was also done for architectural simplicity. Making this simple decision allows all the language support for RDF and N3 to be immediately adopted for the arithmetic expressions; the store can store them, the serializer can output them and so on. Contrast this with the situation in for example SPARQL(Seaborne and Prud'hommeaux 2006), in which a special place in th query expression is reserved for filter expressions, and whole separate syntax is supported for it.

Within the reasoner, built-in functions are made part of the query. During query optimization, *light* builtins (such as negation of an integer) are assumed to be faster than searching the store, and are performed the moment they can be, while *heavy* builtins, such as accessing the Web, making a remote query, or recusively invoking the reasoner itself, are assumed to take a long time, and are postponed until anything faster, including searching the local store, as been done.

There is also support for N-ary bultins, implemented using lists

```
{   (?x.tempInF "32") math:difference ?a.
    (?a "0.5555") math:product ?c.
} => {
     ?x  tempInC ?c.
}
```

4.4 Logic Built-ins

As we mentioned before, our design goals were that the

a) tool has to be able to interact with the web, and

b) rather than retrieving all the data into one big dataset and believing it, rules often have to know which data came from what document. For this, the concept of a formula - a set of RDF statements - is introduced.

In order to interact with the web, cwm has a builtin called `log:semantics`. The `log:semantics` of a document is the formula which one gets by parsing a Semantic Web document. Cwm regards the web as being a mapping between symbols and formulas. As it is a built-in function, when cwm needs to evaluate it it will retrieve up the document (in N3 or RDF/XML format) and parse it, returning the formula. Example:

```
{ <foo.rdf> log:semantics ?f } => { ?f a :InterestingFormula}.
```

After a formula has been retrieved, it is possible to inquire into its contents by using log:includes. One formula `log:includes` a second formula if for each statement in the second, there is a corresponding one in the first.

So let's say we we have a concept of a Semantic Web home page for a person. We decide on the policy that if someone's home page says that they are a vegetarian, then we believe that they are a vegetarian.

```
{?x :homePage log:includes { ?x a :Vegetarian }}=> { ?x a :Vegetarian}.
```

4.5 Proof generation

The proof generation facility is an integral part of the CWM software. For a cwm command line which would have produced a given result, adding `--why` to the command line instead produces a proof of that result. The proof is a chain of inference steps (assertions and rules) with pointers to all the supporting material.

This supporting material is generated every time a fact is added to the KB. When this event occurs, the reason for modifying the KB is stored along with the modification. Possible reasons include, but are not limited to: a triple being inferred by a rule (with a record of the bindings substituted) , being introduced by a builtin or being retrieved from a web resource. This reasons are described in the proof ontology available at [ref].

An important point needs to be made here. An interesting aspect of proof checking on the Web is that the proofs presented may contain not just traditional logics, but also proof steps grounded in "nonlogical"; justifications. For example, suppose that two parties A and B agree on an oracle Z that is to be trusted on the matter of membership to some group. Then, if A wants to prove membership to that group it will generate a proof of membership and one of the inference steps in the proof will be "because Z says so". Thus, in many cases, there will be inference steps like the one described - justifications that must be shared between parties without the ability to appeal to a formal theory. Thus, in cwm, steps in a proof may be made

by reference to an agreed upon "oracle" rather than to a logical mechanism. The web bultin functions allow these trust-like processing to be reduced to operations in logic on the web.

A separate program, check.py, is used to validate a proof.

4.6 Trust/ Digital Signatures

The security, trust, information quality and privacy issues arising from the vision of the Semantic Web as a global information integration infrastructure are essential. Cwm provides the tools with which to write responsible systems. The software itself doesn't make it easier to decide on the trust model to use - but it does provide the capability to express it. Moreover, CWM includes a cryptography module which allows a user to generate public/private keys and sign and verify documents. Between them, the logical and cryptographic functionality, together with the capacity to look up the web, allow a wide variety of trust-aware systems to be built from scratch. One appeal of this technique is that because the language does not have any assumptions about the trust infrastructure built in, the user is not forced to trust based the topology of existing systems, such as a formalized hierarchical public key system. All kinds of different forms of trust can be used for different data. Another appeal is that in fact the trusted code base is fairly small, only the basic builtin functions (code shared by all users and will be hopefully well inspected), and declarative rules which may themselves by automatically analyzed.

4.7 Web-aware queries

The cwm system can operate in modes in when a symbol occurs during processing, it will be dereferenced on the web so that any published information about the symbol may be loaded. The loading can be triggred by a symbol appearing as subject, predicate, or object when a statement is loaded, or when a statement occurs with in a query term being matched. These modes can be controlled individually from the cwm command line.

Our experience is that it is wise to follow predicates when statements are loaded, foring what we term the *ontological closure* of the loaded data. The ontologies loaded (and their ontologies) are finite and small in number, and provide useful metadata to control processing as described below.

However, it is generally unwise to follow subject or object links when loading statements, as in a well-connected web the entire Semantic Web could be eventually loaded.

By contrast, when a query is being matched, it is very reasonable to load any URIs occuring in subject or object positions the query, as this typically brings in new data but not an infinte amount. A query is evaluated by matching a graph template to the graph of data which can be found on the web. If each node in the graph, has useful and relevant information associated with it, and that information is loaded by the query engine when the node is first mentioned, then it is possible

for the query to successively bring in documents which together form the parts of a large graph as it is needed.

These modes of operation, and ones like them, are, we belive, important to the development of the Semantic Web. Future research should be directed toward protocols which involve conventions for the sort of information which is published against the URIs used as symbols for arbitrary things in the Semantic Web.

4.7.1 Definitive documents

There are times when a particular document on the web contains all cases of a particular relation. For example, the relationship between a US state and its two letter code exists in 50 cases. Another document might for example store a definitive list of the MIT course numbers.

In this case, queries involving these properties become self-answering according to the following protocol. The query processor looks up the ontology for the property when it find it in a query. The ontology file mentions that there is a definitive document for the property, with a statement like for example

```
state:code log:definitiveDocument <stateCodes.rdf>.
```

The client then converts any query or query part of the form `?s state:code ?y` into a query on that document.

4.8 Scoped Negation as Failure

While some sources of data are definitive, others (such as a set of temperature measurements) are not: one never knows when evidence may come to light of another. This aspect of the Semantic Web makes negation as failure (NAF) meaningless unless it is associated to a specific dataset. The effect of a default with an explicit domain is achieved with `log:notIncludes`, the negation of `log:includes`. In the example below, if an order has an item which is car, and the order doesn't say that the car has some color, then the car is black.

```
{    <thisOrder.rdf>  log:semantics ?ORDER.
     ?ORDER  log:includes    { ?x  biz:item ?y. ?y a ex:Car };
     ?ORDER  log:notIncludes { ?y  ex:color [] }
} => {
     ?y ex:Color "black"
}.
```

4.8.1 Remote query processing

The modes mentioned above allow cwm to pick up data while it is processing a query, by loading RDF or N3 documents. We were also interested in applications in which large quantities of existing data were in live SQL databases. The cwm

query engine has the ability to pick up metadata from the schemas (the ontological closure above) which directs it, for certain specific Properties, to convert that part of the query into an SQL query. This is done by making the assertion that the property has a `log:definitiveService` whose URI is a made up form of mySQL URI which carries the information on how to access the data.

The implementation is a proof of concept only: it operates only with mySQL databases. We would recommend that, now (2006) that SPARQL will soon be available as a standard, that future designs implement this functionality by converting the query into SPARQL. In this way, systems of federated SPARQL servers should be set up. This is a very interesting direction for future research, specifically the protocols for defining the conditions under which a given server should be contacted for a given form of query.

5 Applications

We now discuss some applications for which we have used cwm. It has also been used as a general teaching and introductory tool for new develepers(Berners-Lee et al. 2003). To validate the usefulness of the technology, we used cwm for a variety of tasks, including tax code calculations, creation and filtering of calendar information, project management tasks such as dependency analysis and bug tracking, and all kinds of quick manipulations of data occuring in everyday personal or enterprise use of data. Here we mention two specific applications as examples.

5.1 Policy Aware Web

The Policy-Aware Web (PAW) Project is a collaboration between Mindswap and DIG aimed at developing a framework for decentralized, rule-based access control on the Web using Semantic Web technologies. The main design goals of the project are to keep access control policies as transparent as possible, and offload the work of proving whether access should be granted or not to the client, instead of the server. The latter is done by requiring the clients to generate proofs (using CWM) of their rights to access a particular resource protected by N3 policies. Thus, cwm's proof generation/checking features are extensively used in PAW.

5.2 Automating Technical Reports

CWM has also been used in the "Technical Report Automation" project. This W3C project, based on the use of Semantic Web tools and technologies, has allowed the streamlining the publication paper trail of W3C Technical Reports, to maintain an RDF-formalized index of these publications and to completely automate the maintenance of the lists of Technical Reports.

The W3C Technical Reports index lists the formal work product of each W3C Working Group categorized by one of six "maturity levels". The maturity levels indicate the state of the work, from (first) working draft to adopted standard (called

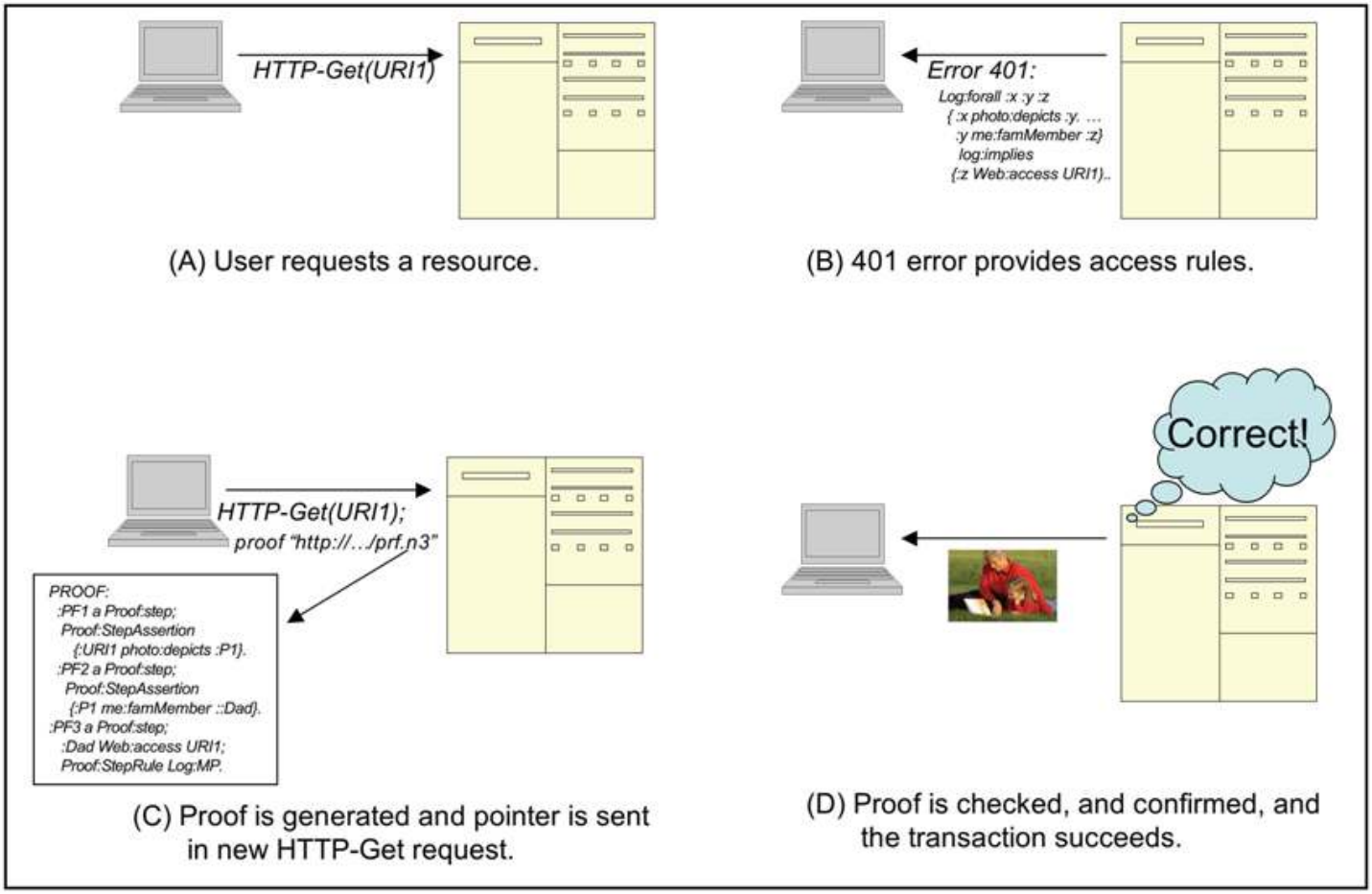

Fig. 2. policy aware web scenario

"W3C Recommendation"). The W3C Process defines the formal steps necessary for a document to advance in maturity level.

For many years the Technical Reports index had been maintained manually and the critical workflow data that it contained was not available in machine processable form. Checking the prerequisites for each maturity advancement was entirely manual. Eventually, the W3C pages listing the Working Groups, their chartered time periods, their deliverables, and the Technical Reports index itself were all turned into authoritative sources of Semantic Web data by adding RDF markup to the existing HTML. The next step was to have data extraction tools which in turn allowed cwm to check some of the dependencies and prerequisites at the time a new document is proposed for publication. The W3C Technical Reports index is now built automatically by cwm and is available to users in multiple views as well as in machine-processable RDF form, allowing a variety of analyses to be performed.

6 Related Work

Euler is an inference engine supporting logic based proofs. Unlike cwm, it is a backward-chaining reasoner enhanced with Euler path detection.

Pychinko is a Python implementation of the classic Rete pattern matching algorithm [REF]. Rete has shown to be, in many cases, the most efficient way to apply rules to a set of facts–the basic functionality of an expert system. Pychinko employs an optimized implementation of the algorithm to handle facts, expressed as triples, and process them using a set of N3 rules. Pychinko tries to closely mimic the features available in Cwm, as it is one of the most widely used rule engines in the RDF community. Pychinko has proven to be faster than Cwm, however it's limitation lies in its expressivity: Pychinko cannot handle most of the cwm builtins. It is worth mentioning here that the RETE engine used in Pychinko has been ported to Cwm - thus Cwm can now boast the same performance improvements.

7 Conclusion and Future Work

The N3 language enabled our use of the Semantic Web both by its simplicity and ease of use as a data language, and in the fact that its extra expressive power allowed rules and rules about what document said what to be written. The extra logical operations added in Cwm to seemed to be a fairly complete set, in that new applications were made by combining the old ones, and did not require the constant additions of new builtins. That said, many areas such as date and time processing, arithmetic for various forms of number, and functions on lists and sets are covered quite incompletely.

The cwm system is a useful, and much used, tool for many practical small applications. However, it was not optimised for speed, and as applications grow there are needs for cwm-compatible fast processors for specific forms of problem. In general our experience demonstrates both the need for and the adequacy of the N3 formula as a quoting system for procesing of data in real web-like environment, where the provenance of data is as important a factor as its content.

Particularly interesting were applications in which a small N3 file caused cwm to find and integrate in real time data from diverse sources on the web, and also small query files which cause cwm to follow links to the point where the query can be resolved. We hope in future to do more research into such systems which work intimately and scalably in the Web.

8 Acknowledgements

This work is supported in part by funding from US Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-00-2-0593, Semantic Web Development.

This work was also funded under NSF ITR 04-012.

Originally written by Berners-Lee and Connolly, CWM has been significantly

enhanced and extended by Scharf. Kolovski ported the *pychinko* rete engine, and authored this paper. Sean Palmer, Mark Nottingham, Jos de Roo, and other members of the W3C Semantic Web Interest Group provided code, tests, comments and encouragement.

References

Beckett, D., Miller, E., and Brickley, D. 2002. Expressing Simple Dublin Core in RDF/XML. Tech. Rep. http://dublincore.org/documents/2002/07/31/dcmes-xml/, Dublin Core Metadata Initiative. July.

Beged-Dov, G., Brickley, D., Dornfest, R., Davis, I., Dodds, L., Eisenzopf, J., Galbraith, D., Guha, R., MacLeod, K., Miller, E., Swartz, A., and van der Vlist, E. 2000. RDF Site Summary (RSS) 1.0. `http://web.resource.org/rss/1.0/`.

Berners-Lee, T., Connolly, D., and Hawke, S. 2003. Semantic Web Tutorial Using N3. `http://www.w3.org/2000/10/swap/doc/`.

Dean, M. and Schreiber, G. 2004. OWL Web Ontology Language Reference. Tech. Rep. http://www.w3.org/TR/2004/REC-owl-ref-20040210/, W3C. February.

Golbeck, J., Fragoso, G., Hartel, F., Hendler, J., Parsia, B., and Oberthaler, J. 2003. The national cancer institute's thesaurus and ontology. *Journal of Web Semantics 1,* 1 (Dec).

Klyne, G. and Carroll, J. J. 2004. Resource Description Framework (RDF): Concepts and Abstract Syntax. Tech. Rep. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/, W3C. February.

Seaborne, A. and Prud'hommeaux, E. 2006. SPARQL Query Language for RDF. Tech. Rep. http://www.w3.org/TR/2006/CR-rdf-sparql-query-20060406/, W3C. April.