A Proof Markup Language for Semantic Web Services

Paulo Pinheiro da Silva   Deborah L. McGuinness
Richard Fikes

Knowledge Systems Laboratory, Stanford University
Stanford, CA 94305, USA.
e-mail: {pp,dlm,fikes}@ksl.stanford.edu

Abstract

The Semantic Web is being designed to enable automated reasoners to be used as core components in a wide variety of Web applications and services. In order for a client to accept and trust a result produced by perhaps an unfamiliar Web service, the result needs to be accompanied by a justification that is understandable and usable by the client. in this paper, we describe the Proof Markup Language (PML), an interlingua representation for justifications of results produced by Semantic Web services. We also introduce our Inference Web infrastructure that uses PML as the foundation for providing explanations of Web services to end users. We additionally show how PML is critical for and provides the foundation for hybrid reasoning where results are produced cooperatively by multiple reasoners. Our contributions in this paper focus on technological foundations for capturing formal representations of term meaning and justification descriptions thereby facilitating trust and reuse of answers from web agents.

1 Introduction

The Semantic Web is being designed to enable automated reasoners to be used as core components in a wide variety of Web applications and services. In order for a client to accept and trust a result produced by an unfamiliar Web service, the result needs to be accompanied by a justification that is understandable and usable by the client. For such justifications to be viable for Semantic Web applications and services, they must be expressed using a standard ontology and a standard Semantic Web representation language so that any client can interpret a justification produced by any service, be portable across the Web in that all references in a justification are URIs, and be combinable so that justifications can be formed for results produced by multiple reasoners.

In this article, we describe the Proof Markup Language (PML) as an interlingua representation for justifications of reasoning results produced by Semantic Web services, and we describe example uses of PML in hybrid reasoning and explanation generation. PML is an ontology added to W3C's OWL Semantic Web representation language [13] so that a PML justification is expressed in OWL and is therefore exchangeable among Semantic Web services and clients using the RDF/XML syntax. PML is a component of the Inference Web (IW) infrastructure for Web-based explanations [11, 9], and makes use of the Inference Web IWBase distributed repository of meta-data including information sources, reasoning systems, and inference rules [10].

PML provides a means of describing a justification as a sequence of information manipulations used to generate an answer. Such a sequence is referred to as a *Proof.* A PML proof can represent many kinds of information manipulations ranging from formal logic derivations to natural deduction derivations to data base and information retrieval operations to the natural language processing performed during information extraction.

The rest of the article is organized as follows. Section 2 describes the Inference Web Semantic Web infrastructure for explanations. Section 3 introduces the PML specification. Section 4 demonstrates practical uses of the PML format for explanations and hybrid reasoning. Section 5 presents related work. Section 6 concludes the article and describes future work.

2 Inference Web

Inference Web (IW) is a framework for explaining answers produced from Semantic Web services and applications. IW provides tools and infrastructure for building, maintaining, presenting, exchanging, combining, annotating, filtering, comparing, and rendering proofs and proof fragments. Inference Web is composed of the following portions:

- *Specifications*: Question answering components may generate answers and justifications for their answers using the PML format described in this article. PML provides a proof interlingua representation. IW includes a specification of PML in OWL[1]. IW also utilizes the Proof Protocol for Deductive Reasoning (PPDR) [15], which is an abstract, uniform way of specifying inference rules.

- *Data*: PML documents published on the Web become a portion of the *Inference Web data* used that is referenced when browsing and summarization tools are presenting explanations, abstractions, and other viewing options to users. Examples of PML documents produced by several question answering systems can be browsed on the Web by following the links in http://iw.stanford.edu/proofs.html. Inference Web also processes the PML documents to identify when portions of them may be combined to

[1]http://iw.stanford.edu/2004/03/iw.owl

form more complex conclusions. IW also uses rewrite rules to transform the proofs into more understandable explanations. The IW data includes the PML proofs and explanations along with a registry of information used for proof presentation. The registry, called IWBase, is a distributed repository of meta-data including sources, inference engines and inference rules. This information is used to support follow-up questions concerning explanations, proofs, and their individual components.

- *Tools and Services*: Inference Web includes a tool suite including: a browser for displaying proofs and explanations; an abstractor for transforming potentially long and incomprehensible PML proofs into shorter and more understandable PML explanations; an explainer for users to ask for explanations and the tool to present explanations in multiple, alternative ways; a registrar for submitting and maintaining the evolving IWBase entries; a proof generation service for facilitating the creation of PML proofs by inference engines; and the IWBase registry for storing information used in proofs and explanations.

The IWBase (formerly known as the IW Registry) is a hyperweb of distributed repositories of meta-information relevant to proofs and explanations, including knowledge provenance information [16]. Every entry in these repositories is an instance of an IWBase concept as described in Section 3.3. For example, *Ontology* is an IWBase concept that is the superclass of entries representing: ontologies, knowledge bases, thesauri, etc. An ontology entry describes stores of assertions about the ontology such as its original creator(s), date of creation, data of last update, version, URL (for browsing), description in English, etc. IWBase's provenance information is expanding on an as-needed basis driven by application demands.

Every entry has a URI and is stored both as a file written in OWL and as a set of tuples in a database. IWBase files are mainly used by PML proofs to annotate their content as described throughout this article.

3 PML Specification

PML classes are OWL classes (thus they are subclasses of `owl:Class`). They are used to build OWL documents representing both proofs and proof provenance information. Thus, PML concepts can be considered to be either *proof level concepts* or *provenance level concepts*. Primitive types mentioned in this article are from the XML schema specification[2].

[2]http://www.w3.org/TR/xmlschema-2/

3.1 Proof Level Concepts

NodeSet[3], InferenceStep, and Expression are the main constructs of proofs and explanations.

A NodeSet represents a step in a proof whose conclusion is justified by any of a set of inference steps associated with the NodeSet. PML adopts the term "node set" since each instance of NodeSet can be viewed as a set of nodes gathered from one or more proof trees having the same conclusion.

- The `URI`[4] of a node set is the unique identifier of the node set. Every node set has one well-formed URI.
- The `Conclusion` of a node set represents the expression concluded by the proof step. Every node set has one conclusion, and a conclusion of a node set is of type Expression.
- The expression language of a node set is the value of the property `hasLanguage` of the node set in which the conclusion is represented. Every node set has one expression language, and that expression language is of type Language.
- Each inference step of a node set represents an application of an inference rule that justifies the node set's conclusion. A node set can have any number of inference steps, including none, and each inference step of a node set is of type InferenceStep. The inference steps are members of a collection that is the value of the property `isConsequentOf` of the node set. A node set without inference steps is of a special kind identifying an unproven goal in a reasoning process as described in Section 4.1.2 below.

An InferenceStep represents a justification for the conclusion of a node set. Inference steps are anonymous OWL classes defined within node sets. For this reason, it is assumed that applications handling PML proofs are able to identify the node set of a inference step. Also for this reason, inference steps have no URIs.

- The rule of an inference step, which is the value of the property `hasRule` of the inference step, is the rule that was applied to produce the conclusion. Every inference step has one rule, and that rule is of type InferenceRule (see Section 3.3.3). Rules are in general registered in the IWBase by engine developers. However, PML specifies three special instances of rules: *Assumption*, *DirectAssertion*, and *UnregisteredRule*. When specified in an inference step, the *Assumption* rule says that the conclusion in the node set is an explicit assumption. The *DirectAssertion* rule says that the conclusion on the node was told by the sources associated with the

[3]PML concept names are typed in sans serif style and PML attribute names are typed in `courier` style.

[4]http://www.ietf.org/rfc/rfc2396.txt

inference step (see the `hasSource` property of an inference step). The *UnregistredRule* says that the conclusion in the node set was derived by some unidentified, unregistered rule. *UnregisteredRule*s allow the generation of proofs-like structures applying undocumented, unnamed rules.

- The antecedents of an inference step is a sequence of node sets each of whose conclusions is a *premise* of the application of the inference step's rule. The sequence can contain any number of node sets including none. The sequence is the value of the property `hasAntecedent` of the inference step. The fact that the premises are ordered may be relevant for some rules such as *ordered resolution* [17] that uses the order to match premises with the schemas of the associated rule. For other rules such as modus ponens, the order of the premises is irrelevant. In this case, antecedents can be viewed as a set of premises.

- Each binding of an inference step is a mapping from a variable to a term specifying the substitutions performed on the premises before the application of the step's rule. For instance, substitutions may be required to unify terms in premises in order to perform resolution. An inference step can have any number of bindings including none, and each binding is of type VariableBinding. The bindings are members of a collection that is the value of the property `hasVariableMapping` of the inference step.

- Each discharged assumption of an inference step is an expression that is discharged as an assumption by application of the step's rule. An inference step can have any number of discharged assumptions including none, and each discharged assumption is of type Expression. The discharged assumptions are members of a collection that is the value of the property `hasDischargeAssumption` of the inference step. This property supports the application of rules requiring the discharging of assumptions such as natural deduction's *implication introduction*. An assumption that is discharged at an inference step can be used as an assumption in the proof of an antecedent of the inference step without making the proof be conditional on that assumption.

- Each source of an inference step refers to an entity representing original statements from which the conclusion was obtained. An inference step can have any number of sources including none, and each source is of type Source as described in Section 3.3.1. The sources are members of a collection that is the value of the property `hasSource` of the inference step. An inference step's source supports the justification of the node set conclusion when the step's rule is a *DirectAssertion*.

- The engine of an inference step, which is the value of the property `hasInferenceEngine` of the inference step, represents the inference engine that produced the inference step. Each inference step has one engine, which is of type InferenceEngine.

- The timestamp of an inference step, which is the value of property `hasTimeStamp` of the inference step, is the date when the inference step was produced. Time stamp is of the primitive type dateTime. Every inference step has one time stamp.

An inference step is said to be well-formed if:

1. Its node set conclusion is an instance of the conclusion schema specified by its rule;

2. The expressions resulting from applying its bindings to its premise schemas are instances of its rule's premise schemas;

3. It has the same number of premises as its rule's premise schemas; and

4. If it is an application of the *DirectAssertion* rule, than it has at least one source, else it has no sources.

Further proof verification may be performed by checking side conditions on declarative rules [15] and running verification methods on method rules. However, a discussion of this level of verification is beyond the scope of this article.

PML node set schemas and PML inference step schemas used later in the article are defined as follows. A PML **node set schema** is a PML node set which has a conclusion that is either a sentence schema[5] or a sentence; which has a set of variable bindings that map free variables in the conclusion to constants; which has zero of more inference steps; and whose inference steps are either inference steps or **inference step schemas**. An inference step schema is an inference set of a node set schema whose antecedents are node set schemas.

An Expression is a PML representation of well-formed logical expressions written in accordance with a given Language.

A proof generated by the Wine Agent[6] is used in Section 3.2 below to describe how answer justifications are represented in a set of PML documents. Figure 1 presents the last node set of a set of PML documents. There, the node set conclusion is a triple written in KIF [5] and based on the RDF predicate `type` saying that `TonysSpecialty` is of type `seafood`. In fact, the `?x` in the node set conclusion is a variable since the conclusion is written in KIF (according to the value of the `hasLanguage` property of node set) and KIF variables are prefixed with a question mark. Moreover, according to the value of the `hasVariableMapping` property of the inference step justifying the node set conclusion, `?x` is bound to the term `SEAFOOD`. The conclusion of the node set in Figure 1 has one justification since the node set has a single inference step.

[5]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.

[6]http://www.ksl.stanford.edu/people/dlm/webont/wineAgent/

```
<rdf:RDF>
 <iw:NodeSet rdf:about="http://.../tonysns4_0.owl#tonysns4_0">
  <iw:conclusion>
      (|http://www.w3.org/1999/02/22-rdf-syntax-ns#|::type
       |http://.../tonys.daml#|::|TonysSpecialty| ?x)
  </iw:conclusion>
  <iw:hasLanguage rdf:resource="http://.../registry/LG/KIF.owl#KIF"/>
  <iw:isConsequentOf rdf:parseType="Collection">
   <iw:InferenceStep>
    <iw:hasRule rdf:resource="http://.../registry/DPR/GMP.owl#GMP"/>
    <iw:hasInferenceEngine
       rdf:resource="http://.../registry/IE/JTP.owl#JTP"
       rdf:type="http://.../iw.owl#InferenceEngine"/>
    <iw:hasAntecedent rdf:parseType="Collection">
     <iw:NodeSet rdf:about="http://.../tonysns4_1.owl#tonysns4_1"/>
     <iw:NodeSet rdf:about="http://.../tonysns4_5.owl#tonysns4_5"/>
    </iw:hasAntecedent>
    <iw:hasVariableMapping rdf:parseType="Collection">
     <iw:VariableMapping iw:Variable="?x">
      <iw:Term>
           |http://.../tonys.daml#|::|SEAFOOD|
      </iw:Term>
     </iw:VariableMapping>
    </iw:hasVariableMapping>
   </iw:InferenceStep>
  </iw:isConsequentOf>
 </iw:NodeSet>
</rdf:RDF>
```

Figure 1: A PML node set.

3.2 Proofs

Since a PML node set can have multiple inference steps and each antecedent of each of those inference steps can have multiple inference steps, a PML node set N and the node sets recursively linked to N as antecedents of inference steps represent a graph of alternative proofs of N's conclusion. In this section, we describe how to extract individual proofs of N's conclusion from that graph of alternative proofs. We shall call each such extracted proof a "proof from N".

We begin by defining a *proof* as a sequence of "proof steps", where each proof step consists of a conclusion, a justification for that conclusion, and a set of assumptions discharged by the step. "A proof of C" is defined to be a proof whose last step has conclusion C. A proof of C is conditional on an assumption A if and only if there is a step in the proof that has A as its conclusion and "assumption" as its justification, and A is not discharged by a later step in the proof. An unconditional proof of C is a proof of C that is not conditional on any assumptions. (Note that assumptions can be made in an unconditional proof, but each such assumption must be discharged by a later step in the proof.) Finally, proof $P1$ is said to be *subproof* of $P2$ if and only if the sequence of proof steps that is $P1$ is a subsequence of the proof steps that is $P2$.

Given these definitions, we can now define the proofs that are extractable from a PML node set as follows: for any PML node set N, P is a "proof from N" if and only if:

1. The conclusion of the last step of P is the conclusion of N;
2. The justification of the last step of P is one of N's inference steps S; and
3. For each antecedent A_i of S, exactly one proof from A_i is a subproof of P.

If N is a node set having conclusion C, then a proof from N is a proof of C.

Figure 2 presents a fragment of the proof represented within the set of PML documents produced by the Wine Agent. The actual rendering of the proof from the set of PML documents as presented in Figure 2 was performed by the IWBrowser discussed in Section 4.3.1 below. The basic idea for getting a proof from the set of PML documents is that tools can traverse the set of node sets, which may be inter-connected by inference steps within the node sets. For example, the node set in Figure 1 has the conclusion of the last step of the proof fragment in Figure 2 (that TonysSpecialty has type Seafood). The node set in Figure 1 has two antecedents (that crab is a subclass of seafood and that since crab is a subclass of ?x then tonysSpecialty has type ?x) that are new proofs and are subproofs of the proof fragment in Figure 2.

A description of the entire proof for the answer that TonysSpecialty is a kind of seafood is beyond what is necessary for pedagogical purposes in this article. However, the set of PML documents representing the proof for this answer is available on the Web[7], which can be browsed[8] using the IW Browser.

3.3 Provenance Level Concepts

Inference Web stores provenance information about proofs and explanations in the IWBase. This section describes the concepts supported by IWBase that are part of the PML specification.

3.3.1 Provenance Element

ProvenanceElement represents a information unit describing the origin of some PML proof level concept introduced in Section 3.1. ProvenanceElement is a superclass of the PML concepts at the provenance level and the ProvenanceElement attributes are described as follows:

- The `URI` of a provenance element is the unique identifier of the provenance element. Every provenance element has one well-formed `URI`, which is an instance of the primitive type anyURI.

[7]http://iw.stanford.edu/proofs/jtp/tonys/tonys/
[8]http://iw.stanford.edu/documents/isexample.html

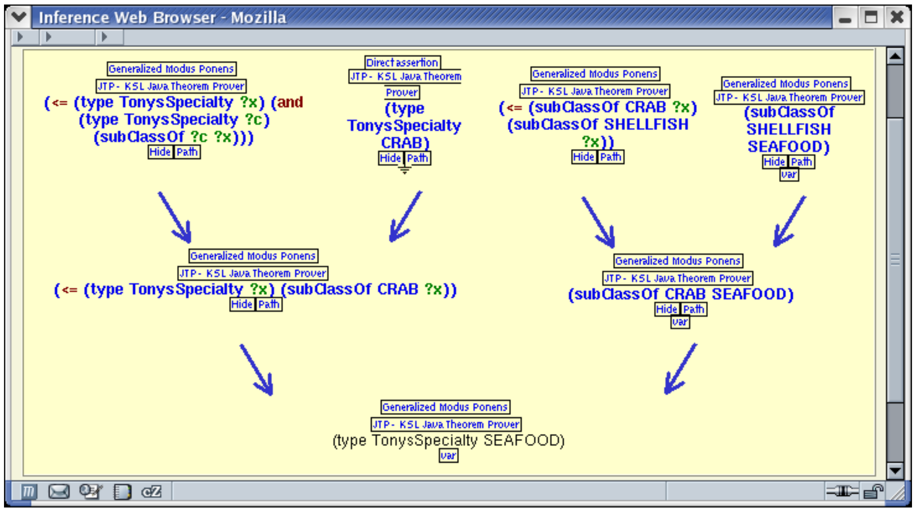
Inference Web Browser - Mozilla
Generalized Modus Ponens
JTP - KSL Java Theorem Prover
(<= (type TonysSpecialty ?x) (and (type TonysSpecialty ?c) (subClassOf ?c ?x)))
Hide Path
Direct assertion
JTP - KSL Java Theorem Prover
(type TonysSpecialty CRAB)
Hide Path
Generalized Modus Ponens
JTP - KSL Java Theorem Prover
(<= (subClassOf CRAB ?x) (subClassOf SHELLFISH ?x))
Hide Path
Generalized Modus Ponens
JTP - KSL Java Theorem Prover
(subClassOf SHELLFISH SEAFOOD)
Hide Path
Var
Generalized Modus Ponens
JTP - KSL Java Theorem Prover
(<= (type TonysSpecialty ?x) (subClassOf CRAB ?x))
Hide Path
Generalized Modus Ponens
JTP - KSL Java Theorem Prover
(subClassOf CRAB SEAFOOD)
Hide Path
Var
Generalized Modus Ponens
JTP - KSL Java Theorem Prover
(type TonysSpecialty SEAFOOD)
Var

Figure 2: A partial view of a proof extracted from PML documents.

- The URL of a provenance element describes a URL used to browse an element's web document. For instance, if the provenance element is an organization named the New York Times, then the http://www. nytimes.com URL can be used to access a web document about the organization. In this case the URL points to the organization's web site. A provenance element can have zero or one URLs.

- The Name of a provenance element describes a short name (or "nickname") for the element within the IWBase. Every provenance element has one name, and the name is an instance of the primitive type string.

- The Submitter of a provenance element represents the team of people responsible for the registration of the provenance element in IWBase. Every provenance element has one submitter, and the submitter is an instance of the type Team, which is a subclass of Source as described in Section 3.3.2.

- The DateTimeInitialSubmission of a provenance element is the date when the provenance element was first registered in IWBase. Every provenance element has one DateTimeInitialSubmission, and that is an instance of the primitive type dateTime.

- The DateTimeLastSubmission of a provenance element is the last date when the provenance element was registered in IWBase. Every provenance element has one DateTimeLastSubmission, and that is an instance of the primitive type dateTime.

- The EnglishDescription of a provenance element is a description in English of the provenance element. A provenance element can have zero or one descriptions in English, and EnglishDescription is an instance of the primitive type string. The description in English is intended to be used by tools to present provenance elements to human agents. For example, the description in English of the *Modus Ponens* inference rule may be a better presentation and more informative for most users browsing a PML document than the presentation of the formal logical specification of the rule.

3.3.2 The Source Concept

A Source is a ProvenanceElement representing an entity which is the source of the original data. Current types of Inference Web sources include: InferenceEngine, Ontology, Organization, Person, Publication, Language, and Team. We provide a description of two source types here.

- An InferenceEngine represents an engine that is able to produce a justification for a given conclusion. Note that the use of the term "inference engines" in this article is not limited to engines with reasoning capabilities. For example, search engines retrieving information may serve as an inference engine and provide a justification of their answer by a direct assertion inference step. Similarly extraction modules may be viewed as inference engines and, in fact, we have registered a number of extraction modules from the UIMA extraction toolkit[2] so that extracted answers may be explained in Inference Web using PML.

- A Language represents a language used to write conclusions of node sets. Any language can be registered in IWBase including formal logical languages, such as KIF, and natural languages, such as English, and representation languages such as DAML+OIL and OWL.

3.3.3 The Rule Concept

An InferenceRule represents a ProvenanceElement specialization describing rules applied to premises deriving node set conclusions. An InferenceRule can be either a PrimitiveRule or a DerivedRule.

A PrimitiveRule is a type of an InferenceRule that is implemented by one or more inference engines. A given rule R_1 may not be called primitive until it becomes associated with one or more inference engines. Thus, assuming that R_1 is implemented by an inference engine E_1, the inference engine may declare R_1 to be a primitive rule. The notion that R_1 is a primitive rule for one specific engine is relevant since R_1 may also be derived from R_2, which is a primitive rule for another engine E_2. In this case, R_1 may be registered once as a primitive rule and zero or more times as a derived rule, depending on how many combinations of rules are used to derive R_1. For example a natural deduction reasoner E_1 may define *modus ponens* as a primitive rule and another reasoner E_2 may register Robinson's *resolution* rule as a primitive rule. The E_2 reasoner may be able to derive a *modus ponens* rule using its primitive resolution rule. Thus, any rule

R_1 can be registered multiple times, once as a primitive rule and multiple other times as derived rules depending on the number of different combinations of rules used to derive R_1.

- The specification of a primitive rule is a string describing in a declarative way the sequence of premise schemas, the conclusion schema, and the syntactical conditions for applying the rule.
- The language of a primitive rule is the language in which the primitive rule specification is written. A primitive rule may have one language, which is of type Language.

A DeclarativeRule is a PrimitiveRule and is well-formed if and only if all the syntactic conditions for the primitive rule hold.

A MethodRule is a PrimitiveRule whose conditions cannot be completely specified in terms of the attributes of the conclusion's node set, the conclusion's inference step applying the primitive rule, and the premises' node sets. Inference rules that are "procedural attachments" are examples of method rules. The current work on PML does not involve checking if method rules are well-formed although future plans include checking support.

A DerivedRule is an InferenceRule specified from a PML node set schema with the restriction that each node set schema must have one and only one inference step. The `Specification` of a derived rule represents a proof (as defined in Section 3.2) from a given PML node set schema since each PML node set must have one and only one inference step. Moreover, the derived rule's proof is a **proof schema** since the PML node sets of the proof are PML node set schemas.

4 Using PML Proofs

4.1 Support for Hybrid Reasoning

Experience with automated reasoners has made clear that in order to effectively determine answers to complex real-world questions, general-purpose reasoners need to be augmented with special-purpose reasoners that embody both domain-specific and task-specific expertise. That is, effective deductive answer determination requires *hybrid reasoning*.

We have developed an object-oriented modular 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 [3]. The JTP architecture and 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.

Proofs represented in PML play a central role in the JTP architecture in that they are used to represent both queries and reasoning results as they are sent to and received from reasoners during the hybrid reasoning process. In this section we describe JTP's use of PML proofs in hybrid reasoning.

4.1.1 JTP System Architecture

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 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 PML 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.

Since the set of answers to a query may be of unpredictable size and may require an unpredictable amount of time to derive, 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 provide output reasoning steps as they are derived and for additional derivations to be attempted on an as needed basis.

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.

4.1.2 Reasoning Steps

Reasoners produce enumerations of reasoning steps, and take reasoning steps as input. A reasoning step is a PML node set schema that represents a partial or completed proof of a symbolic logic sentence.

A reasoning step that is a node set schema N having conclusion C, variable bindings B, and no inference steps specifies a query to find proofs of instances of the sentence schema C/B (i.e., the sentence schema produced by applying the bindings B to C). A reasoner given such a reasoning step as input can produce either partial or complete proofs of instances C/B. Each partial or complete proof to be returned by the reasoner can be represented by adding inference steps and variable bindings to a copy of the input reasoning step. Each node set in a reasoning step produced by a reasoner that does not have an inference step is an unproven subgoal for which a proof is needed in order to complete the proof.

Thus, reasoning steps are used to represent both queries and reasoning results as they are sent to and received from reasoners during the hybrid reasoning process.

4.1.3 The Tell and Ask Commands

The *Tell Command* takes as input a sentence S and adds it to the KB. The command processor does that by forming a reasoning step R representing a proof of S justified as a direct assertion, and then calling a telling reasoner with P as input.

A telling reasoner takes as input a reasoning step that is a proof. The proof may represent either a sentence that is being told to the system (justified as a direct assertion), or a result of a previous inference, justified by an inference rule, that the reasoner is to build upon. The reasoner may assert the sentence to one or more knowledge stores, produce additional inferences in the form of new proofs, or signal that a contradiction has been found. The output of a telling reasoner is an enumerator whose output when pulsed is a proof representing the result of a new inference.

The *Ask Command* takes as input a sentence schema S, and produces as output an enumerator whose output when pulsed is an unconditional proof of an instance of S. The command processor produces its output by forming a reasoning step having conclusion S, and then calling an asking reasoner with that reasoning step as input.

An asking reasoner accepts as input a reasoning step R having conclusion C, and no inference step. R represents a query whose answers are instances of C. The reasoner attempts to produce reasoning steps having the same conclusion as R and a variable binding set that is a superset of R's variable binding set. The reasoner's output is an enumerator whose output when pulsed is such a reasoning step. If no reasoning steps can be produced, then the enumerator is empty.

For example, consider a query to find bindings for v_1 and v_2 such that "(P v_1 v_2)" is true. Assume the knowledge base contains the axioms "(=> (and (Q x y) (R y z)) (P x z))" and "(Q a b)". An asking reasoner that receives this query as a node set schema could return a node set schema with "(P v_1 v_2)" as the conclusion, $\{(v_1\ a)\}$ as the variable bindings, and an inference step schema whose rule is generalized modus ponens; whose bindings are (x a) and (y b); and whose antecedents are A_1, A_2, and A_3. The conclusion of A_1 would be "(=> (and (Q x y) (R y z)) (P x z))", and A_1 would have an inference step whose rule is "direct assertion". The conclusion of A_2 would be "(Q a b)", and A_2 would have an inference step whose rule is "direct assertion". The conclusion of A_3 would be "$(R\ b\ z)$", and A_3 would have no inference steps. Thus, "$(R\ b\ z)$" would be an unproven subgoal to be dispatched to an appropriate reasoner in an attempt to complete the proof.

4.2 Support for Knowledge Provenance

Users and reviewers of our work have consistently been interested in having simple ways to obtain the ground assertions that were used to find a particular conclusion C. A summary of the statements used in the proof provides one course level of abstraction of the proof. A tool using PML can take any particular answer represented by the conclusion C of a node set N and trace back through the inference steps used in a recursive way, looking at their antecedents and determining all of the sources used to arrive at C. Thus, the resulting collection has the sources of all possible proofs from N in the set of PML documents, which may be more interesting for some users than just the collection of sources of one proof from N. Beyond simple collections of the statements or sources used, PML documents also have links to the meta-information available for these sources. Thus a user may find that the particular conclusion C relied on exactly two knowledge bases used as information sources (and potentially a particular set of sentences in those two knowledge bases) and additionally may learn that those two knowledge bases were considered authoritative sources by a particular verification body and the two knowledge bases were updated within the last week. That may be as much information as some users would like to see about a conclusion at a particular time.

In practical scenarios such as those used on the ARDA AQUAINT[9] and NIMD[10] projects, presentation of knowledge provenance information can be large lists of assertions entailing the conclusion of a node set. In terms of PML proof concepts, the ground assertions are the conclusions of node sets justified by inference steps applying the direct assertion rule. The list of sources is a consolidation of the sources of the ground assertions' inference steps justifying the ground assertions. The meta-information of the sources are the entries of the sources in the IWBase. Thus, PML proofs are the artifacts relating knowledge provenance information in the IWBase to explanations of answers provided by Semantic Web applications and services.

4.3 Support for Explanations

Any transformation of the proof of a conclusion that is more understandable to the user than the proof is considered an explanation in this paper. The presentation of PML proofs as discussed in Section 4.3.1 provides a strategy to explain conclusions. Many users of the Semantic Web, may be unable to understand logical proofs or simply be unwilling to review them. For these users, the presentation of the ground assertions used to conclude C as already discussed in Section 4.2 may be a better type of explanation for C since it gives

[9]http://www.ic-arda.org/InfoExploit/aquaint/
[10]http://www.ic-arda.org/Novel_Intelligence/

them an understanding of what the conclusion depended upon without going into the details of the inference(s) used. In general, any abstraction of the proof of C may also be an explanation of C. The use of rewriting rules as discussed in Section 4.3.2 is an operational method for abstracting proofs and is one that we leverage to generate explanations that provide some notion of the inferences without providing all of the details.

4.3.1 Browsing PML Documents

The IW browser is a Web application used to render PML documents as human-friendly HTML documents. Given a node set's URI, the browser can render a proof by using the node set's inference steps to generate graph composed of node sets and their inference steps. The use of the PML format provides two immediate benefits for browsing proofs:

- *Direct Access to Proof Nodes*: Proofs can easily be composed of hundreds or thousands of node sets. Users, however, may not need to browse all nodes to understand a large proof. In fact, we often see that information from a few key node sets may be enough for most users to understand a proof. Further, not only may it be enough, it is preferable to only view a few critical components of the proof instead of viewing most or all of the proof. Thus, PML proofs allow users to refer to URIs of specific node sets of proofs rather than be forced to deal with a monolithic proof.

- *Lightweight Loading of Proofs*: The browser's **proof lens** is a metaphor of a magnifier visualizing parts of proofs. The lens **focus** is the conclusion of a given node set. The lens **magnitude** is the maximum number of inference steps traversed on any single path for presentation in the limited view. The user may "refocus" the lens by choosing something other than the current conclusion and then only view a portion of the proof supporting that statement. The lens notion supports lightweight proof loading since node sets are loaded only on demand, thus all the node sets not in the lens focus will not be loaded until a refocusing occurs that requires them for presentation.

When interacting with the browser, users may select from a number of *proof styles* and *sentence formats* for displaying PML documents. Proof style is the layout used during proof presentation. For example, a "textbook" logic layout style uses a bar to separate the conclusion of a node set from the premises of that conclusion.

The name of the inference step is placed on the right side of bar. Sentence format identifies the preferred way of formating the conclusions of node sets. The "raw" format means that conclusions are presented as they are represented in the PML document. Other sentence formats rely on the browser capability of translating conclusions from their original languages into the requested format. Since many users prefer to see a form of natural language output, we have provided a simple logic to English presentation module that will present nodes labeled in KIF in a limited English format.

4.3.2 PML Explanations

When Inference Web uses abstraction techniques for proofs, it will hide potentially a lot of information about how a conclusion was reached. We have observed that hiding many of the primitive core rules in reasoners is useful for many users. For example, users may not want to see many applications of "modus ponens" in the JTP reasoner and may instead prefer to see one application of a courser grained inference such as class transitivity. The JTP reasoner was built using primitive rules because they were useful for efficient implementation of the reasoner, but not necessarily because they are useful for human understanding. Typically primitive rules are at the wrong level of granularity to present to an end user and also many times they are the wrong granularity to present to agents as well.

The rewriting of proofs based on primitive rules into proofs based on derived rules is one way of abstracting primitive rules. However, syntactic manipulations of proofs may also be insufficient for abstracting machine-generated proofs into some more understandable proofs [6]. Proofs can become more understandable if they are rewritten using IW **tactics**, that are rules derived from axioms from language descriptions such as the DAML [4] axiomatic set. In tactics, axioms are the elements responsible for aggregating steps together in order to make the rules more understandable.

The IWBase registrar includes an editor of derived rules that can be used to specify tactics. The IW abstractor algorithm generates explanations in a systematic way using IWBase derived rules. For any particular conclusion C, the IW abstractor may abstract away a number of node sets of proof of C. Using the IW explainer, the user may always ask for alternative explanations through follow-up questions and still obtain the proof of C, however the explanation of C and the presentation of provenance information provide abstracted explanations. The general result of using the IW abstractor in the IW explainer is to hide primitive rules and expose higher-level derived rules.

The ability to provide portions of proofs and provide support for follow-up questions has been found to be a critical component in creating usable explanation systems[8]. Inference Web follows this architectural design of being able to present stand alone components of proofs and then support follow-up questions that are appropriate in the context of any particular proof.

4.4 PML API and Proof Generation Services

A PML API is fully implemented in Java and will be publicly available soon. For now, most PML users rely on the *proof generation services (PGS)* available on any node of the IWBase including the IWBase Core node [11]. Thus, using the PGSs, inference engines can call the services passing node set and inference step attributes according to the PGS documentation, which return the PML documents to be stored under a web server. Compared with the PML API,

[11]PGSs documentation and links for the core node are available at http://iw.stanford.edu/documents_registering.html

PGSs are a better solution for generating PML since they already have support for querying the IWBase in order to include the meta-information annotating the proofs. Also, PGSs provide an uniform way of generating PML in case the PML specification evolves.

5 Related Work

Automated reasoners have many different ways to represent proofs [18]. Moreover, a single reasoner can have multiple ways of representing proofs. For example, a reasoner can have an internal representation of proofs that has a number of features to handle optimizations and an external representation of proofs used to present a trace of how conclusions are derived. Proofs exchanged between reasoners are usually external representation of proofs. Indeed, many proof elements such as optimization properties can be useless for other reasoners because the optimizations are tied to the internals of a particular engine.

External representations of proofs have been developed for several reasons. For example, most automated reasoners are able to produce a simple trace of their proofs in order to support author debugging. These traces though were rarely meant for end users or even programs but they can provide the raw material for generating explanations. The need to check proofs is a more sophisticated reason to have a external representation of proof. For example, Watson [19] created a technique to represent and check proofs produce by Isabelle [14].

Most external representations of proofs in use by more than one automated reasoning were developed within the context of hybrid reasoning systems and logical frameworks. But even in the context of hybrid reasoning systems, these external representations typically have goals of hybrid reasoning interoperability rather than our goal with PML of providing a general proof representation. PDS [1] is one example of a general representation of proofs that is used in hybrid reasoning systems such as MBase [7].

PML and PDS have some similarities and differences. They share the ability of representing proofs at different levels of abstraction. For instance, the PML ability to generate explanations from proofs corresponds to the PDS notion of "third dimension" for is proof representation. The MBase use of PDS is also of particular interest for PML since it demonstrated the need of a web artifact to represent proofs. Indeed, MBase uses an XML version of PDS. PML is more flexible than PDS for describing information manipulation in general however. For example, PML sentences may have multiple justifications and may be written in any language. While PML was originally designed with explanation of hybrid reasoners, it has had a major influence from work explaining information extraction and has evolved to facilitate explanations of those results (such as the work with the UIMA extractors). PML has had a primary emphasis of supporting portable proofs using web infrastructure aiming at interoperable web explanations. Another emphasis that separates our work on PML and Inference Web is our emphasis on explanation of provenance and our registration mechanism (IWBase) providing this support.

In order to show some of the power of our explanation work with Inference Web and PML, we have provided a few examples of web services utilizing PML proofs. The KSL Wine Agent, the DAML Query Language Client[12], and the OWL Query Language Client all use PML proofs in order to provide interoperable explanations of their answers. In this paper, we only showed a simple example from the KSL wine agent in its typical task of suggesting wine and food combinations for particular meals. In that example, we just showed a portion of the explanation for the type of a particular meal (TonysSpeciality). The implemented system provides explanations for food types as well as wine combinations as well as wine suggestions and places to buy them. The wine agent loads a knowledge base into the JTP reasoner, then asks JTP for answers to questions such as what wine should be served with a particular meal, then presents the answers along with their PML proofs through a specialized interface using the Inference Web browser and explainer. In the simple example in this paper, we just showed a portion of a proof that would have been dumped for the wine description for the meal that served TonysSpeciality. Since that wine description suggested a white wine, a user might have asked for the type of the food (it was a crabdish which is also of course a seafood dish and the knowledge base contains information in it that suggests white wines with seafood dishes). Using this and other web services, we have provided implemented examples of how one can use PML and the inference web to exploit knowledge bases in languages such as OWL and reasoner that can provide justifications for answers in PML to provide explanations of their answers using the Inference Web.

The IW Browser[13] is a web service that renders PML proofs and presents them in multiple formats for humans. All of these agents presented in this paper use the Stanford JTP hybrid reasoner but other reasoners and question answering systems are being registered in Inference Web and are supporting PML dumps. We have extended a version of SRI's SNARK theorem prover[14] to produce PML proofs and we have also worked with ISI to integrate ISI's Prometheus[15] query planner with Inference web. We have also worked with IBM to integrate a number of their information extractors to produce PML and thus to be able to use inference web to explain their findings. Trento's semantic match system using JSAT has also been integrated with Inference Web and PML and can now provide explanations of semantic matches[12]. We are also pursuing discussions with designers of other reasoning systems including W3C's CWM[16] and UT's KM[17] and SRI's SPARK.

In these discussions and in our integration efforts, we have gathered requirements for a portable proof markup language that will support standard first order logic theorem provers as well as a broad array of systems that return answers to question such as extractors that take in natural language documents

[12]http://onto.stanford.edu:8080/dql/servlet/DQLFrontEnd
[13]http://iw.stanford.edu/iwbrowser
[14]http://www.ai.sri.com/~stickel/snark.html
[15]http://www.isi.edu/info-agents/Prometheus/
[16]http://www.w3.org/2000/10/swap/doc/cwm.html
[17]http://www.cs.utexas.edu/users/mfkb/km.html

and extract sets of structured facts or query planners that take in a query and determine how to answer the query by posing subqueries to resources that are expected to have information relevant to the query.

6 Conclusion and Future Work

In this article we have introduced a proof markup language – PML – that is used to support web services inter-operation and trust. Since PML is an interlingua and since a number of question answering systems can justify their answers using PML, inter-operation is facilitated. Since trust is increase (for humans or agents) when they can ask for the reasons that an answer is believed, and further find meta information about the information and the assumptions on which any information manipulation step is based, trust is increased.

PML has been integrated into the Inference Web infrastructure and is used as the interlingua that allows IW to present, combine, summarize, abstract, and explain answers generated by web services. In our discussion of hybrid reasoning and our integration efforts with JTP, we presented how PML supports hybrid reasoning and provided one implemented example of how the integration with a reasoner works. This implementation then provides the basis for utilizing PML and Inference web to provide explanations of a number of web services that may serve as pedagogical examples. We provided a brief description of how the KSL wine agent uses PML and Inference Web to explain its suggestions and conclusions.

PML provides the representational foundation and Inference Web provides the tools and infrastructural support for enabling proof annotation, knowledge provenance presentation, and explanation browsing for agents and humans. This work provides the foundation from which web services may realize their promise of providing remote, interoperable access across components of distributed software systems.

Acknowledgments. The authors would like to thank Pat Hayes for many thought provoking conversations concerning explanation and for the PML name.

References

[1] Lassaad Cheikhrouhou and Volker Sorge. PDS – A Three-Dimensional Data Structure for Proof Plans. In *Proceedings of the International Conference on Artificial and Computational Intelligence (ACIDCA'2000)*, Monastir, Tunisia, March 2000.

[2] D. Ferrucci and A. Lally. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Journal of Natural Language Engineering*, June 2004. To appear.

[3] Richard Fikes, Jessica Jenkins, and Gleb Frank. JTP: A System Architecture and Component Library for Hybrid Reasoning. Technical Report KSL-03-01, Knowledge Systems Laboratory, Stanford University, Stanford, CA, USA, 2003.

[4] Richard Fikes and Deborah L. McGuinness. An Axiomatic Semantics for RDF, RDF-S, and DAML+OIL (March 2001). Technical Report Note 18, W3C, December 2001.

[5] Michael R. Genesereth and Richard Fikes. Knowledge interchange format, version 3.0 reference manual. Technical Report Logic-92-1, Computer Science Department, Stanford University, Stanford, CA, USA, 1992.

[6] Xiaorong Huang. Reconstructing Proofs at the Assertion Level. In *Proceedings of CADE-94*, LNAI-814, pages 738–752. Springer, 1994.

[7] Michael Kohlhase and Andreas Franke. MBase: Representing Knowledge and Context for the Integration of Mathematical Software Systems. *Journal of Symbolic Computation*, 32(4):365–402, September 2001.

[8] Deborah L. McGuinness. *Explaining Reasoning in Description Logics*. PhD thesis, Rutgers University, 1996.

[9] Deborah L. McGuinness and Paulo Pinheiro da Silva. Infrastructure for Web Explanations. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, LNCS-2870, pages 113–129, Sanibel, FL, USA, October 2003. Springer.

[10] Deborah L. McGuinness and Paulo Pinheiro da Silva. Registry-Based Support for Information Integration. In *Proceedings of IJCAI-2003 Workshop on Information Integration on the Web (IIWeb-03)*, pages 117–122, Acapulco, Mexico, August 2003.

[11] Deborah L. McGuinness and Paulo Pinheiro da Silva. Explaining Answers from the Semantic Web. *Journal of Web Semantics*, 2004. To appear.

[12] Deborah L. McGuinness, Pavel Shvaiko, Fausto Giunchiglia, and Paulo Pinheiro da Silva. Towards Explainig Semantic Matching. In Volker Haarslev and Ralf Möller, editors, *Proceedings of the 2004 International Workshop on Description Logics*. CEUR-Workshop Proceedings, 2004. To appear.

[13] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. Technical report, World Wide Web Consortium (W3C), February 10 2004. Recommendation.

[14] Lawrence C. Paulson. Isabelle: the next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361–386. Academic Press, 1990.

[15] Paulo Pinheiro da Silva, Patrick Hayes, Deborah L. McGuinness, and Richard Fikes. PPDR: A Proof Protocol for Deductive Reasoning. Technical Report KSL-04-04, Knowledge Systems Laboratory, Stanford University, Stanford, CA, USA, March 2004.

[16] Paulo Pinheiro da Silva, Deborah L. McGuinness, and Rob McCool. Knowledge Provenance Infrastructure. *IEEE Data Engineering Bulletin*, December 2003. To appear.

[17] J. Reynolds. Unpublished seminar notes. Stanford University, Stanford, CA, 1966.

[18] Geoffrey N. Watson. Proof Representations in Theorem Provers. Technical Report 98-13, Software Verification Research Centre, The University of Queensland, Queensland, Australia, September 1998.

[19] Geoffrey N. Watson. *A Generic Proof Checker*. PhD thesis, The University of Queensland, Queensland, Australia, 2002.