Bossam: An Extended Rule Engine for OWL Inferencing

Minsu Jang and Joo-Chan Sohn

Intelligent Robot Division, Electronics & Telecommunications Research Institute,
Gajeong-dong 161, Yuseong-gu, Daejeon-si, 305-350, South Korea
{minsu, jcsohn}@etri.re.kr

Abstract. In this paper, we describe our effort to build an inference engine for OWL reasoning based on the rule engine paradigm. Rule engines are very practical and effective for their representational simplicity and optimized performance, but their limited expressiveness and web unfriendliness restrict their usability for OWL reasoning. We enumerate and succinctly describe extended features implemented in our rule engine, Bossam, and show that these features are necessary to promote the effectiveness of any ordinary rule engine's OWL reasoning capability. URI referencing and URI-based procedural attachment enhance web-friendliness. OWL importing, support for classical negation and relieved range restrictedness help correctly capture the semantics of OWL. Remote binding enables collaborated reasoning among multiple Bossam engines, which enhances the engine's usability on the distributed semantic web environment. By applying our engine to the W3C's OWL test cases, we got a plausible 70% average success rate for the three OWL species. Our contribution with this paper is to suggest a set of extended features that can enhance the reasoning capabilities of ordinary rule engines on the semantic web.

1 Introduction

The semantic web is an extension of the current web in which information is given well-defined meaning [1], by representing data formally and explicitly in a sharable way. According to the semantic web stack, ontology and rules are the two key components for formal and explicit data representation. To interpret data represented in ontology and rules and derive new information, an appropriate inference mechanism is necessary.

There are a number of reasoning mechanisms for the semantic web, but each of them has its own pros and cons. Dedicated description logic reasoning engines like FaCT[2] and Pellet[3] are great for their firm theoretical soundness, but they are not very useful for practical problems as they strictly stick to open-world assumption and logical perfection. We believe that practical solutions need to provide flexible reasoning ability that can deal with dynamic and very conflicting knowledge space. Another class of tools such as Hoolet [4] and Surnia [5] are based on automatic theorem provers and share the similar low practicality problem with description logic reasoning engines. These reasoning tools cannot deal with ECA rules and do not allow ref-

G. Antoniou and H. Boley (Eds.): RuleML 2004, LNCS 3323, pp. 128–138, 2004.
© Springer-Verlag Berlin Heidelberg 2004

erencing external objects in the rules. OWL reasoning engines like Jena [6], F-OWL [7], and [8] are based on logic programming or production rule system. With effective reasoning algorithms, they can process rules quite effectively. But [6] does not yet support negation in its rule language, and JESS-based tools like [8] support only negation-as-failure so they cannot correctly capture the semantics of OWL which is based on classical monotonic logic. On the other hand, [7] supports both negation-as-failure and classical negation, but it does not support procedural attachment. As procedural attachment is a necessary feature for practical applications, lacking the feature imposes many problems to utilize the tool in the real setting. Also, these tools put strict range-restrictedness on the head of rules. Range restrictedness dictates that every variable in the consequent part should also be present in the antecedent. But to properly implement entailments imposing OWL comprehension principle, it's necessary to introduce new variables in the consequent part.

We were motivated by the situation that there's no reasoning engine that provides sufficient expressiveness and extra-logical features we identified as required for effective OWL reasoning. We started to build Bossam as a typical rule engine and then added to it a set of extended feature elements to promote effectiveness of the engine's OWL reasoning capability.

Bossam is a RETE-based forward chaining engine, which is equipped with extended representational and extra-logical features:

- Support for both negation-as-failure and classical negation
- Relieved range-restrictedness in the rule heads
- Remote binding for cooperative inferencing among multiple rule engines

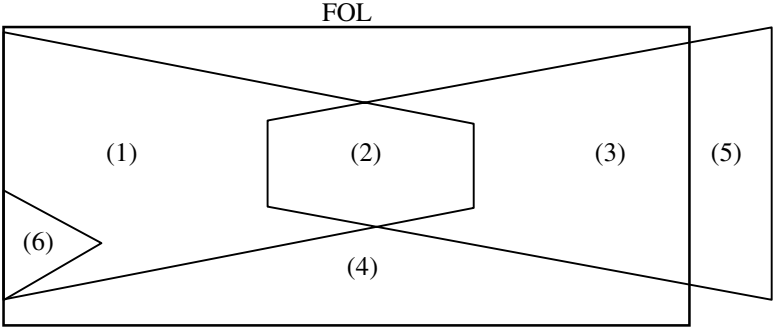
In the following two sections, we generally characterize Bossam by presenting its expressiveness and web-friendliness enhancements. In section 4 and 5, we describe Bossam's extended expressiveness elements and remote binding feature.

2 Bossam's Expressiveness

Fig.1 illustrates the expressiveness of Bossam. The outermost rectangle is the boundary of first-order logic's expressiveness. Description logic and horn logic form two overlapping fragments inside FOL. Logic programming is largely a part of FOL, but it contains extra-logical features, such as negation-as-failure, procedural attachment, conflict resolution etc, which are not characterized inside FOL. The extra-logical features are essential even though they defile the clarity of a logic system's formal characterization. Because, in most real-life applications, interaction with external objects, rapid decision making etc are key requirements, which are possible only through employing aforementioned extra-logical features.

As explained in Fig.1, Bossam is based on LP, with two added expressiveness fragments: (4) and (6). Fragment (4) provides syntactic convenience useful for concise rule writing. Bossam breaks down the rules containing elements of (4) into several horn rules according to Lloyd-Topor transformation [9]. With fragment (4), it's easy to write OWL inference rules that create an RDF graph with multiple nodes as

an entailment from a premise document, which is not easy with horn rules. In Bossam, every atom contained in the conjunction at the consequent is derived into a fact in case of the rule firing. As for the fragment (6), we give detailed description in section 4.



Description Logic (DL)=(1)+(2)+(6), Horn Logic (HL)=(2)+(3),
Logic Programming (LP)=(2)+(3)+(5),
Description Logic Programming (DLP)=(2)
(4): A fragment which includes disjunctions in the body and conjunctions in the head,
(5): NAF, procedural attachment, conflict resolution etc.
(6): Classical negation.

Fig. 1. Expressiveness of Bossam corresponds to (2)+(3)+(4)+(5)+(6).

3 Web Friendliness Enhancements

Any web rules language should make it easy to write rules with URIs. Bossam offers some simple enhancements with its rule language and its interpreter so that URIs can be used intuitively in its rule-bases.

3.1 Seamless URI Integration

Bossam facilitates web friendliness by adding URI as its native symbol type. Actually, all the symbols – except variable symbols – in Bossam's rule-base are URIs. The following example shows a rule and a fact, which are written in Bossam rule language.

```
prefix family = http://family.com/Family#;
namespace = http://family.com/Johns#;
rule r1 is
   if
      family:isFatherOf(?x,?y)
      and family:isBrotherOf(?z,?x)
```

```
    then
       family:isUncleOf(?z,?y);
fact f1 is
    family:isFatherOf(John,Bob);
```

In the example above, `family:isFatherOf` and `family:isBrotherOf` may be the terms defined in a remote OWL ontology. Bossam can import remote OWL documents into its internal working memory and perform reasoning on them. The keyword `namespace` is used to define the base namespace of the rule-base. In the above example, the constant `John` specified in the fact `f1` is expanded to a full URI: http://family.com/Johns#John. As shown, it's easy in Bossam's rule language to specify and refer to namespaces and URIs. For a rule language to be used on the web, it should offer syntactic medium to seamlessly and intuitively integrate web resources into its rules.

3.2 OWL Importing

Many a web resource referred in Bossam rules are OWL vocabularies. For OWL reasoning, we wrote a set of OWL inference rules in Bossam rule language. Bossam imports and translates OWL ontology into a list of Bossam facts and then applies the OWL inference rules on them.

There're two approaches to OWL translation. The first is to translate OWL documents into a collection of RDF triples and then each triple into a plain fact with three terms [10] [11]. The second is to translate OWL documents into a set of sentences of the target logic system [4] [12]. The first approach is very simple and general, but some basic logical meanings contained in the original documents are not preserved in the translated result. That is, even the elementary logic constructs are axiomatized such that the implied logical relations in the original OWL constructs are not preserved in the translated result [15]. The second approach does guarantee semantics-preserving translation, but the translatability is limited by the target language's expressiveness.

Table 1. OWL ontology translation examples

OWL statements	Bossam facts
`<owl:Class rdf:ID="Person"/>` `<Person rdf:about="#Sam"/>`	`owl:Class(Person);` `Person(Sam);`

`<owl:Individual rdf:about="#John">`
`<person:father rdf:resource="#Sam"/>`
`</owl:Individual>`
	`person:father(John,Sam);`

`<owl:Class rdf:about="#Human">`
`<owl:unionOf`
`rdf:parseType="Collection">`
`<owl:Class rdf:about="#Woman"/>`
`<owl:Class rdf:about="#Man" />`
`</owl:intersectionOf>`
`</owl:Class>`
	`owl:unionOf(Human, <Woman, Man>);`

With Bossam, we chose the first approach in favor of its simplicity. Bossam translates RDF triples involved in declaring OWL classes and restrictions into 1-ary predicates, and the triples declaring property values into 2-ary predicates. And RDF col-

lections are translated into Bossam's built-in list constants. Table 1 shows three basic examples of Bossam's OWL translation strategy.

3.3 Web-Friendly Procedural Attachment Mechanism

Even though web ontology is appropriate for expressing and sharing static knowledge, it's not adequate for denoting rapidly changing knowledge such as the values of sensors, stock quotes, etc. This kind of knowledge can be made readily accessible by calling external objects. Also, the ability to alter the status of external objects has been the common requirement for rule applications, which might be the same for rule applications on the web. Reading from and writing values onto external objects can be realized by procedural attachment mechanism.

In Bossam, we implemented a web-friendly procedural attachment mechanism, which can be extended to call any object exposed on the web. We defined a special URI structure for denoting calls to external java objects. Here's an example.

```
java://org.etri.sensor/Temperature#get(?x,?loc,?t)
```

The URI scheme, `java`, indicates that the URI is denoting a resource different from usual web resources; in this case, a java object. The path part denotes the package name and the class name. Then, the fragment ID, `get`, denotes the method name. Every external object in the reasoning context is checked for its type and bound to `?x` if it is of the type `org.etri.sensor.Temperature`. `?loc` is the input parameter to the method. The returned value or object from calling the method `get(?loc)` on `?x` is then bound to `?t`.

Extending URI structure in this way is an intuitive way of incorporating external data or objects on the web into reasoning, as it can easily be extended to denote web services, database tables, CORBA objects etc.

4 Extended Expressiveness

We describe in this section two extended expressiveness elements that are not supported in typical rule engines. The introduced expressiveness elements help correctly capture the semantics of OWL.

4.1 Support for Classical Negation, as Well as NAF

OWL semantics is based on open-world assumption, so classical negation should be available for correct representation of OWL semantics. For example, a disjoint class relation, $C1 = \neg C2$, can be written as two rules, *{if C1(?x) then neg C2(?x); if C2(?x) then neg C1(?x)},* where *neg* represents a classical negation. Ordinary rule engines are based on closed-world assumption and they cannot properly represent and process classical negation.

Bossam includes two symbols for denoting negations: `not` for NAF and `neg` for classical negation. Bossam can natively perform de Morgan's law on classical negation, and declares inconsistency by detecting the presence of both positive and negative facts inside its knowledge base.

In [13] and [14], some interesting examples of showing the usefulness of using both NAF and classical-negation on the web are introduced. One representative example involving both NAF and classical-negation is the *coherence principle*, which is the basis of common-sense reasoning [14]. The principle can be expressed in Bossam rule language as follows.

```
rule cp11 is if neg ?p(?x) then not ?p(?x);
rule cp12 is if neg ?p(?x,?y) then not ?p(?x,?y);
rule cp21 is if ?p(?x) then not neg ?p(?x);
rule cp22 is if ?p(?x, ?y) then not neg ?p(?x,?y);
```

`cp11` and `cp21` are for 1-ary predicates, and `cp12` and `cp22` are for 2-ary predicates. For some interesting examples and implications of the principle, the reader is referred to [14].

We conjecture that there're two kinds of knowledge that will be circulating on the semantic web. The first is the static knowledge such as genealogy, monetary system, membership representation schema etc that contains general truths that do not change often. And the second is the dynamic knowledge such as membership management rules, payment strategies, business contract rules etc that contains strategic and business-centric truths and policies that do change often according to the business and strategic needs. Monotonic reasoning based on open-world assumption is appropriate for processing static knowledge to guarantee correct and safe propagation of truths. But for dynamic knowledge, flexible and context-sensitive non-monotonic reasoning is more appropriate to efficiently draw practical conclusions. We think it should become a common requirement for an inference mechanism on the semantic web that it has to effectively deal with a mixed set of static and dynamic knowledge. To satisfy the requirement, an inference mechanism should be able to correctly represent and perform reasoning with knowledge involving both negation-as-failure and classical-negation.

4.2 Relieved Range Restrictedness

Most rule engines put a strict restriction on the rules: every variable in the consequent part should appear in the antecedent part. This is called range restrictedness [9]. Range restrictedness guarantees the safeness of rules.

But some OWL entailments require creation of new RDF resources in the consequent part of rules. OWL *comprehension principle* is the representative example. As a sample, consider a cardinality restriction with a cardinality value 1. This OWL restriction entails a pair of a minimum and a maximum cardinality restriction both with a cardinality value 1. That is, *restriction(p, cardinality(1))* entails {*restriction(p, minCardinality(1)) and restriction(p, maxCardinality(1))*}. This entailment requires two new restrictions be created as a conclusion. To express this in a production rule, two

new variables should be introduced at the consequent, which is not possible with typical rule engines. Bossam supports this by relieving range-restrictedness.

We extended typical production algorithm so that range restrictedness can be alleviated in a specific case. In Bossam, you can introduce new variables in *type-declaring 1-ary predicates* in the consequent part. The introduced variables can be referenced in subsequent predicates in the same consequent. The following is a simple Bossam rule that implements the aforementioned comprehension principle regarding cardinality restriction.

```
rule CardinalityEntailment001 is
  if
    owl:Restriction(?r) and owl:onProperty(?r,?p)
    and owl:cardinality(?r,?n)
  then
    owl:Restriction(?r1) and owl:onProperty(?r1,?p)
    and owl:minCardinality(?r1,?n)
    and owl:Restriction(?r2) and owl:onProperty(?r2,?p)
    and owl:maxCardinality(?r2,?n);
```

In the consequent part of the rule above, two new variables, `r1` and `r2`, are introduced for the predicate `owl:Restriction`. Upon encountering predicates of the form like this, Bossam internally creates new anonymous resources and binds each of them to the corresponding new variable.

5 Remote Binding for Distributed Reasoning

Remote binding is a simple mechanism to enable cooperative reasoning among multiple Bossam engines. On the semantic web, knowledge bases, whether they are specified in ontology or rules, are distributed and managed independently. To perform reasoning on them, it is necessary to collect and combine knowledge from various sources. One direct way to accomplish this is to read all the required ontologies and rule-bases from remote hosts and combine them into one big local knowledge base. This approach is easy to implement, but it should be noted that as the size of knowledge base gets bigger, the performance of reasoning mechanisms downgrades very quickly. It's important to keep the size of a knowledge base reasonable.

Distributed (or collaborated) reasoning, when implemented in a proper way, enables sharing knowledge between inference engines and each of them can be free from knowledge saturation. Bossam provides a collaboration mechanism that enables a simple form of distributed reasoning. Every instance of Bossam engine maintains a *knowledge catalog* that maps namespaces to engines. When a Bossam engine encounters a vocabulary it does not maintain locally, it looks up the catalog to find a list of relevant engines based on the namespace of the vocabulary. Upon finishing the lookup, the engine issues a query to each relevant engine one by one until a satisfactory answer is achieved.

Fig 2 shows the overall structure of remote binding mechanism. The knowledge catalog contains a map of namespaces to physical URIs. Each URI refers to the point

of contact to send queries about the corresponding namespace. In the current version of Bossam, knowledge catalogs should be specified and provided by the user.

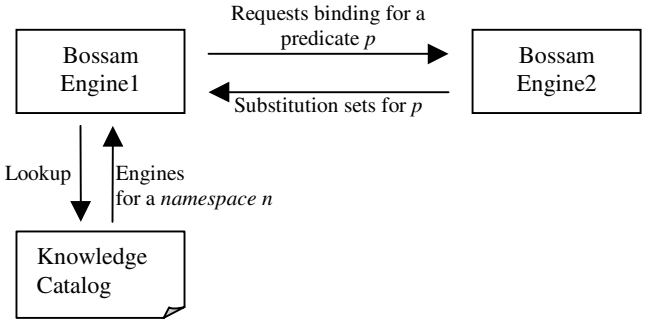


Fig. 2. Overall structure of remote binding

In remote binding, the content of a query is the request for substitution set for a predicate *p*. The answer to the query is a set of bindings, which in turn is fed by the asking engine into its ongoing process of producing further implied models.

In RETE network, unification is performed at alpha nodes. In Bossam, when a RETE network is built, an alpha node capable of remote binding is created whenever a predicate with a predicate symbol that has a foreign namespace is encountered. The remote-binding alpha nodes always contact remote inference engines to perform unification.

6 OWL Inference Test Results

In this section, we present Bossam's OWL inference test results. We tested Bossam against OWL test cases defined by W3C [17], in part, to validate the effectiveness of our approach. Out of the 10 categories of OWL tests, we applied our engine only to positive entailment tests that are more relevant to the inferencing capability.

6.1 On Processing Positive Entailment Tests

Each OWL positive entailment test is composed of two OWL documents: one premise document and one conclusion document. OWL reasoning engine should be able to entail the conclusion document when given with the premise document as an input. For each positive entailment test, we executed an inference session on Bossam with the given test's premise document and then queried the engine with the conclusion document to find out if the conclusion document holds in the models Bossam produced from the premise document. To do this, we converted each conclusion docu-

ment into a Bossam query, which is then transformed into a Bossam rule for further processing.

The following is an example that illustrates the basic approach of the conversion. This example document was depicted from the OWL test case at http://www.w3.org/2002/03owlt/FunctionalProperty/Manifest005#test. The namespace prefix `eg` corresponds to http://www.example.org/, and `foo` corresponds to http://www.example.org/foo#.

(d1) Original conclusion document in N3 [16]*:*

```
eg:foo#object rdf:type owl:Thing.
_:a rdf:type owl:Restriction.
eg:foo#prop rdf:type owl:FunctionalProperty.
_:a owl:onProperty eg:foo#prop.
_:a owl:maxCardinality "1"^^xsd:nonNegativeInteger.
eg:foo#object rdf:type _:a.
```

(d2) Bossam query generated from (d1):

```
(((((?a(foo:object) and  owl:Restriction(?a)
    ) and owl:onProperty(?a,foo:prop)
   ) and owl:FunctionalProperty(foo:prop)
  ) and owl:Thing(foo:object)
 ) and owl:maxCardinality(?a,1)
)
```

(d3) Bossam rule transformed from (d2):

```
rule q is
  if
     ?a(foo:object) and owl:Restriction(?a)
     and owl:onProperty(?a,foo:prop)
     and owl:FunctionalProperty(foo:prop)
     and owl:Thing(foo:object)
     and owl:maxCardinality(?a,1)
  then
     Result(?a);
```

As can be seen, anonymous RDF nodes in the conclusion document are converted into Bossam variables. Bossam tries to answer the query by finding some successful bindings to the variables. If a conclusion document does not contain anonymous RDF nodes, it's converted into a query composed of only ground predicates. As Bossam is a forward-chaining engine that is data-driven, it internally converts queries into rules and applies the rules to the forward-chaining process to see if the rules fire with successful bindings. If any rule fires, then the corresponding query is declared to be true.

6.2 Test Results

Table.2 summarizes the success rates of Bossam and other OWL inference engines. The number of tests for each species of OWL is 23 for OWL Lite, 29 for OWL DL, and 41 for OWL Full [17]. The success rate of Bossam marked middle to high among representative OWL inference engines.

Table 2. OWL Test Results (data excerpted from W3C site, as of Dec. 2003)

Engine / Species	*Bossam*	Hoolet	Cerebra	Pellet	Euler	FOWL	FaCT	Surnia	Jena2
OWL Lite	*95%*	82%	73%	82%	100%	65%	4%	26%	69%
OWL DL	*51%*	62%	51%	89%	100%	6%	10%	3%	17%
OWL Full	*68%*	N/A	12%	82%	100%	48%	N/A	41%	68%

One thing to be noted is that other engines listed in Table 2 do not offer the practical features that Bossam do offer. For example, there's no engine except Bossam in Table 2 that supports procedural attachment, two negations and remote binding. Also, it needs to be commented that description logic based engines like Pellet, FaCT and Cerebra are not capable of dealing with rules.

As the semantic web technology development progresses further into the higher layer of the semantic web stack, rule-processing capability will be much required from reasoning engines. And, as the semantic web technology gets wide acceptance by the business fields, practical – extra-logical – reasoning features will become the deciding factor for choosing the solutions for web reasoning.

7 Concluding Remarks

If a reasoning tool were to be utilized pervasively in the real world settings, it should provide rich practical features. The strength of successful reasoning mechanisms, especially rule engines, is related to their highly efficient reasoning performance and rich extra-logical features. On the semantic web, we believe that the reasoning tools inheriting the pros of (currently commercial) rule engines will survive as the most viable reasoning mechanism.

In this paper, we described Bossam, a rule engine extended with various features to improve web friendliness, OWL reasoning capability, and usability on the web. We plan to extend and refine reasoning capability of Bossam to make it a more reliable and competitive reasoning tool for the semantic web, and to investigate the possibility of applying it to some real semantic web applications. Especially, we're trying to extend Bossam's remote binding mechanism so that knowledge catalog can be automatically created and maintained.

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (2001)
2. Horrocks, I.: The FaCT System. LNAI, Vol. 1397. Springer-Verlag, Berlin (1998) 307-312
3. Maryland Information and Network Dynamics Lab.: Pellet OWL Reasoner. (2003) http://www.mindswap.org/2003/pellet/index.shtml
4. Bechhofer, S.: Hoolet OWL Reasoner (2003) http://owl.man.ac.uk/hoolet/
5. Hawke, S.: Surnia (2003) http://www.w3.org/2003/08/surnia/
6. Hewlett-Packard: Jena Semantic Web Framework (2003) http://jena.sourceforge.net/
7. UMBC: F-OWL: An OWL Inference Engine in Flora-2 http://fowl.sourceforge.net/
8. Gandon, F. L., Sadeh, N.: OWL inference engine using XSLT and JESS. http://mycampus.sadehlab.cs.cmu.edu/public_pages/OWLEngine.html
9. Grosof, B., Gandhe, M., Finin, T.: SweetJess: Inferencing in Situated Courteous RuleML via Translation to and from Jess Rules. (2003) http://ebusiness.mit.edu/bgrosof/paps/sweetjess-wp-050203.pdf
10. Kopena, J., Regli, W.: DAMLJessKB: A Tool for Reasoning with the Semantic Web, LNCS, Vol. 2870. Springer-Verlag, Berlin Heidelberg New York (2003) 628-643
11. Fikes, R., Frank, G., Jenkins, J.: JTP: A Query Answering System For Knowledge Represented in DAML. (2002)
12. Grosof, B., Horrocks, I., Volz, R., Decker, S.: Description logic programs: Combining logic programs with description logic. In Proceedings of WWW2003. (2003) 48
13. Wagner, G.: Web Rules Need Two Kinds of Negation. LNCS, Vol. 2901. Springer-Verlag, Berlin Heidelberg New York (2003) 33-50
14. Alferes, J., Damicio, C., Pereira, L.: Semantic Web Logic Programming Tools. LNCS 2901 (2003) 16-32
15. Horrocks, I., Volz, R.: Rule Language. A Deliverable from IST Project 2001-33052 WonderWeb. (2003)
16. Berners-Lee, T.: Primer: Getting into RDF and Semantic Web using N3. (2004) http://www.w3.org/2000/10/swap/Primer.html
17. Jeremy J. Carroll, Jos De Roo: OWL Web Ontology Language Test Cases. W3C Recommendation 10 February 2004 (2004) http://www.w3.org/TR/owl-test/