EXTENDING LEGACY AGENT KNOWLEDGE BASE SYSTEMS WITH SEMANTIC WEB COMPATIBILITIES

Po-Chun Chen, Guruprasad Airy, Prasenjit Mitra and John Yen
College of Information Sciences and Technology, The Pennsylvania State University, University Park, PA, U.S.A.

Keywords: Knowledge base, Rule-based system, Semantic web, Agent.

Abstract: Knowledge bases with inference capabilities play a significant role in an intelligent agent system. Towards the vision of the semantic Web, the compatibility of knowledge representation is critically important. However, a legacy system that was developed without this consideration would have compatibility gaps between its own knowledge representation and the semantic Web standards. In order to solve this problem, we present a systematical approach to extend a legacy agent knowledge base to be able to handle and reason information encoded in standard semantic Web languages. The algorithms presented in this paper are applicable to compatible rule-based systems, and the methodology can be applied to other knowledge systems.

1 INTRODUCTION

Knowledge base systems with inference capabilities have been developed for decades. This kind of knowledge bases plays an important role as the “brain” of an intelligent agent system and is being widely leveraged in many decision support systems and other kinds of enterprise applications. These existing systems usually contain well-defined knowledge in terms of sets of rules that are developed and tested by knowledge engineers cooperating with domain experts. Each rule set is theoretically well-defined and sophisticated for the tasks it is designed to handle. However, due to the domain expertise-driven development process and potentially complex nature of rule-based systems, it is usually challenging to make major revision to an existing rule set. Moreover, it is even more difficult to migrate an existing rule set to another knowledge base system.

Just like other information systems, legacy knowledge base systems are also subject to being left behind as the world moves on. As the progress of research in the semantic Web, efforts in standardization of knowledge representation have made important steps toward the goal of the vision. The approach of using description logics to model concepts and relationships had been well recognized and led to semantic Web standards such as OWL (Web Ontology Language), OWL 2, SWRL (Semantic Web Rule Language), and RIF (Rule Interchange Format).

In order to process knowledge encoded in the semantic Web languages, there have been efforts and accomplishments in how to implement those semantics in rule-based systems (Meditskos and Bassiliades, 2008). In addition, the interoperability between SWRL and other rule-based systems has also been explored (O’Connor et al., 2005). However, little literature has demonstrated the detailed procedure of how to migrate an existing rule-based system toward the semantic Web environment without losing its originally designed functionalities.

To bridge the gap between legacy agent knowledge base systems and the semantic Web standards, we proposed an approach to extend existing knowledge base systems with semantic Web compatibility by facilitating knowledge exchangeability. With this extension, any standard-conformed knowledge engineering toolkit, ontology/rule editor, or other development tool can be used to work with the legacy system. Consequently, a wide range of supporting tools and possible extensions to the existing system become applicable and achievable, which also implies that the legacy system can benefit from the improved knowledge exchangeability.

2 TOWARDS SEMANTIC WEB COMPATIBILITIES

Figure 1 illustrates the framework of how a legacy knowledge base system can be extended with seman-
tic Web compatibilities. We assume there is a knowledge base embedded in the agent, and the knowledge is persistent in its legacy format. In order to be exchangeable with other semantic Web-compatible systems such as third-party editing tools, the knowledge in legacy formats needs to be translated into representations in semantic Web standards. The two core components in this framework are 1) a translator for converting knowledge in legacy formats into semantic Web standards such as OWL and SWRL; and 2) another translator that converts knowledge in the reverse direction. The two translators bridge the knowledge representation gap between the legacy system and semantic Web standards. The detailed design and implementation of these two translators are assumed to be system-specific and depend on the semantic Web languages being used and the format of knowledge representation of the legacy knowledge base system.

Below is an example FactType definition for the predicate Person\(\text{name,age,\text{gender}}\), where each argument is denoted by a prefix question mark “?”. Two example Facts instantiated according to the “Person” FactType definition are listed as well.

\[
(\text{FactType Person (\text{name,\text{gender},age=\text{male}})})
\]

\[
(\text{Fact Person (Alice age 30 male)})
\]

\[
(\text{Fact Person (Bob age 25 male)})
\]

A Rule is based on Horn logic and is composed of a number of antecedents and a consequent. If all of the antecedents are satisfied under some variable assignment, the rule will be fired to instantiate the consequent as an ImpliedFact using the values from this variable assignment.

**Definition.** A Rule \(R = \{P, Q, X\}\) is composed of one or more ordered antecedents \(P = \{P_1, \ldots, P_n\}\), a consequent \(Q\), and a set of shared variable \(X = \{x_1, \ldots, x_o\}\) among \(P_1, \ldots, P_n\) and \(Q\). Each of \(P_1, \ldots, P_n\), and \(Q\) is a predicate with a particular assignment that contains only constant values or shared variables from \(X\). All variables that appear in the consequent should be bounded by being used in the antecedents. \(\forall x_i \in X, (x_i \text{ is used in } Q) \Rightarrow (x_i \text{ is used in } P)\). A rule is said to be “fired” when there is an assignment \(\Sigma'\) on \(X\) that satisfies \(P_1 \cap \ldots \cap P_n\), and then a Fact of \(Q\) will be instantiated based on this assignment \(\Sigma'\).

**Definition.** An ImpliedFact is a Fact that is instantiated due to firing a rule.

Below is an example of a rule \(R = \{P, Q, X\}\), where \(P = \{\text{Person(?\text{name,\text{gender},\text{age}},<?\text{age} \geq 12)}\}, Q = \text{Boy(?\text{name,\text{gender}})}, X = \{?\text{name,\text{gender}}\}\).

\[
(\text{Rule (Person (?\text{name,\text{gender,\text{age}}}) < (?\text{age} \geq 12))})
\]

\[
(\text{Boy (?\text{name,\text{gender}})})
\]

The syntactic specification in Extended Backus-Naur Form (EBNF) is outlined in Figure 2.

2.2 Translation Algorithms

2.2.1 OWL Elements and FactType/Fact

The three fundamental elements in an OWL ontology are Classes, Properties, and Individuals. A Property can be either a DatatypeProperty or an ObjectProperty associated with a Class, and Individuals are the instantiations of a Class. Based on this concept, our approach is to translate a Class with its Properties into a FactType in the knowledge base and Individuals of this Class into Facts of this FactType.

Due to performance consideration and inherited restrictions of the R-CAST knowledge base, the current design is focused on supporting knowledge rep-
represented in OWL-Lite and SWRL rules with axioms based on OWL-Lite. OWL-Lite is a computational feasible sublanguage of OWL, which only supports knowledge with cardinality being either 0 or 1.

Algorithm 1: Class/Property Definitions to FactTypes.

Require: A set of Class axioms \( C = \{c_1, \ldots, c_m\} \), a set of DatatypeProperty axioms \( DP = \{d_1, \ldots, d_n\} \), and a set of ObjectProperty axioms \( OP = \{o_1, \ldots, o_k\} \).

1: \( \text{function OWLToFactTypes}(C, DP, OP) \)
2: \( \text{for all } c_i \in C \) do
3: \( \text{factTypeName} \leftarrow \text{name of } c_i \)
4: \( \text{Create a FactType } f_i \text{ named } \text{factTypeName} \)
5: \( \text{AddPropertyToFactType}(f_i, \text{"InternalID"}) \)
6: \( \text{for all } d p_j \in DP, d p_j, \text{domain } = c_i \) do
7: \( \text{propertyName} \leftarrow \text{name of } d p_j \)
8: \( \text{AddPropertyToFactType}(f_i, \text{propertyName}) \)
9: \( \text{end for} \)
10: \( \text{for all } o p_j \in OP, o p_j, \text{domain } = c_i \) do
11: \( \text{propertyName} \leftarrow \text{name of } o p_j \)
12: \( \text{AddPropertyToFactType}(f_i, \text{propertyName}) \)
13: \( \text{end for} \)
14: \( \text{end for} \)

The algorithm for translating an OWL-Lite ontology into knowledge base representation is shown in Algorithm 1. Since every Class name in an OWL ontology is unique, each Class will be translated into exactly one FactType. Similarly, since in an OWL ontology every Property name is unique, each Property will also be converted into exactly one property within a FactType. Since each Class will be translated into a unique FactType with its associated Properties, the concepts in the OWL ontology will be preserved in the legacy knowledge base.

Besides, in each of the resulting FactType, we create an additional property named "InternalID" for storing internally-generated identifiers of individuals instantiated based on its corresponding Class. The InternalID is used as an internal reference in the knowledge base for a Fact to identify itself.

The algorithm for translating FactTypes to an OWL-Lite ontology is illustrated in Algorithm 2. Since the legacy system does not support cross reference from a FactType to another, every property of a FactType is translated into a DatatypeProperty.

Algorithm 2: FactTypes to Class/Property Definitions.

Require: A set of FactType \( FT = \{f_1, \ldots, f_m\} \)
1: \( \text{function FactTypesToOWL}(FT) \)
2: \( \text{for all } f_i \in FT \) do
3: \( \text{className} \leftarrow \text{name of } f_i \)
4: \( \text{Create an OWL Class } c_i \text{ named } \text{className} \)
5: \( \text{Get the property list } p_j = \langle p_{i1}, \ldots, p_{in} \rangle \text{ from } f_i \)
6: \( \text{for all } p_{ik} \in p_i \) do
7: \( \text{propertyName} \leftarrow \text{name of } p_{ik} \)
8: \( \text{uniquePN} \leftarrow \text{className} \text{"""} \text{propertyName} \)
9: \( \text{Make a DatatypeProperty } d p_{ik} \text{ named } \text{uniquePN} \)
10: \( \langle d p_{ik}.\text{domain} \rangle \leftarrow c_i \)
11: \( \langle d p_{ik}.\text{range} \rangle \leftarrow \text{String} \)
12: \( \text{end for} \)
13: \( \text{end for} \)

For Individuals and Facts, we convert OWL-Lite Individuals into Facts in the legacy knowledge base by generating one Fact for each Individual. When creating a Fact, the URI (Uniform Resource Identifier) of the Individual will be used as its InternalID since the URI is unique in an OWL ontology. If there is no URI designated to an Individual, a unique identifier will be generated for it. To illustrate the usage of InternalIDs, Figure 3 shows an example with two classes where one is referred by the other through an ObjectProperty. The two FactTypes listed below are generated for it. To illustrate the usage of InternalIDs, Figure 3 shows an example with two classes where one is referred by the other through an ObjectProperty. The two FactTypes listed below are generated for it. To illustrate the usage of InternalIDs, Figure 3 shows an example with two classes where one is referred by the other through an ObjectProperty.

For the translation from Facts to OWL-Lite Individuals, it is straightforward to create an OWL Individual and then assign all the property values by using its associated OWL Property definitions.

Figure 3: An example ontology in OWL-Lite.
the antecedents and a head element for the consequent. The content of a head or a body is an AtomList, which is composed of a number of Atoms. There are four primary types of Atoms in SWRL: ClassAtom, DatavaluedPropertyAtom, IndividualPropertyAtom, and BuiltinAtom. ClassAtom is a unary predicate for declaring the class of an object, where the object can be either an identifier or a variable. DatavaluedPropertyAtom and IndividualPropertyAtom are binary predicates that are used to associate an object with a value or another object. A BuiltinAtom is an element embedded with a functional predicate that supports a predefined operation.

According to the specification shown in Figure 2, an antecedent can be either a predicate based on a FactType definition or a functional predicate which implements an operation. The translation consists of three parts. The first part translates antecedent predicates based on FactType definitions, or non-functional predicates, into SWRL ClassAtoms and PropertyAtoms. For example, the first antecedent of the example rule in 2.1 "(Person (?name ?age male))" will be translated into four Atoms as follows.\footnote{The "#" sign is for indicating that it is an URI in the ontology file. The namespace part is ignored here.}

\begin{verbatim}
ClassAtom(#C1, #Person)
DatavaluedPropertyAtom(#C1, #name)
DatavaluedPropertyAtom(#C1, #age)
DatavaluedPropertyAtom(#C1, "male")
\end{verbatim}

The second part translates functional predicates in the antecedent such as arithmetic and comparison operators. It creates a BuiltinAtom with its built-in element referring to the URI of the corresponding SWRL built-in ontology. The arguments to a BuiltinAtom is a RDF List, and the sequence of the arguments should be carefully organized if the parameters are defined differently between the functional predicate and the SWRL built-in. For example, the second antecedent of the example rule in 2.1 "((age 12))" will be translated into a BuiltinAtom as follows.

\begin{verbatim}
BuiltinAtom(builtin=swrlb:lessThan, arguments=(#age, 12))
\end{verbatim}

The last part translates the consequent into a SWRL ClassAtom and several PropertyAtoms. It is necessary to identify whether this rule is designed to update existing individuals or is designed to create new individuals. If the rule is designed to create new individuals, we need to declare a new SWRL variable in the antecedents and use it in the consequent.

The translation from SWRL to legacy rules has a similar structure. Due to the restriction of using OWL-Lite, each object declared with ClassAtom should have non-duplicated PropertyAtoms in order to satisfy the cardinality requirement of OWL-Lite.

3 EXPERIMENT AND RESULTS

In order to verify this approach, we applied the translators of both directions to two existing decision support systems (Airy et al., 2006), with 7 different legacy knowledge sets in the first system and 9 for the other one. The tasks of each system include reasoning information based on its own rule set, generating the values of predefined decision factors that describe the situation, and providing recommendations.

We translated the legacy knowledge sets into OWL and SWRL and then translated them back to the legacy format. We also checked the consistency between the situation descriptions generated by the original systems and those generated by systems based on the forward-back translated knowledge given the same information. All of the test sets show consistent results, thus confirming that these translators preserve the functionalities of the legacy systems.

4 CONCLUSIONS

In this paper, a framework for extending legacy knowledge base systems with semantic Web compatibilities is presented. Although the implementation is assumed to be system-specific, the methodology can be applicable to other knowledge base systems.

Future work will include moving forward to support OWL-DL. Since OWL-DL allows cardinality to be larger than 1, the current data structure and the translation algorithms would need to be redesigned.

REFERENCES


