OWL-AA: Enriching OWL with Instance Recognition Semantics for Automated Semantic Annotation

Yihong Ding, David W. Embley
Department of Computer Science
Brigham Young University
Provo, Utah 84602
(ding, embley)@cs.byu.edu

Stephen W. Liddle
Department of Information Systems
Brigham Young University
Provo, Utah 84602
liddle@byu.edu

Abstract

Although OWL provides a solid basis for many semantic web applications, it lacks sufficient declarative semantics for instance recognition to support automated semantic annotation. This omission prevents OWL from being a satisfactory ontology language for automated semantic annotation. This problem can be solved by adding declarative instance recognition semantics to OWL. Our declarative instance recognition semantics include external representations and context recognition information of ontology concepts. The implementation shows that the new automated annotation prototype system using OWL ontologies with rich declarative instance semantics works well.

1 Introduction

Semantic annotation research is fundamental for the semantic web. A *semantic annotation* process adds formal metadata to web pages. This metadata links data in a web page to defined concepts in an ontology. Because machine agents are capable of interpreting data with respect to an ontology, annotated content becomes machine-processable.

Automated semantic annotation is the primary means of adding machine-processable metadata to existing web pages. Several researchers have suggested various ways to automate semantic annotation [1, 5, 12, 18, 20, 22]. Each of these approaches has adapted a data extraction engine to wrap and annotate existing web pages. None of the adapted data extraction engines, however, was originally designed to produce annotations linking extracted data to an ontology [18, 19]. To provide machine-processable semantics for annotated content, these approaches therefore need to do post-processing to map the extracted data to an ontology. Kiryakov and his colleagues referred to this problem as the "main drawback" of current approaches to automate annotation [18]. They suggest that extraction techniques for semantic annotation should "use ontolog[ies] more directly during the process of extraction" [18].

Our ontology-based semantic annotation research shows that this suggested solution does indeed work [7]. There is, however, a hidden problem in this solution that has not yet been well addressed. Any system that does not conform to semantic web standards will not be interoperable and thus will not be used. The current standard (W3C recommended) semantic web ontology language is OWL (Web Ontology Language) [W3Cowl]. But OWL lacks sufficient declarative semantics for instance recognition, which are needed to extract data directly with respect to ontologies. Our ontology language [7] supports rich declarative instance recognition semantics in our annotation approach, but it is not a standard.

To solve this problem, we propose in this paper an extension of OWL that contains enriched declarative instance recognition semantics. Because this extension is specifically for automated annotation, we call it OWL-AA (OWL for Automated Annotation). Instead of just proposing our OWL-AA syntax, the goal of this paper is to illustrate the essence of a sound solution to the problem.

The primary contribution of this work is that we have proposed OWL-AA as a way to extend OWL to provide for automated semantic annotation. Furthermore, as a significant consequence, OWL-AA separates the creation of domain knowledge from the implementation of a processor to use domain knowledge for the purpose of annotating web pages. With OWL-AA, domain experts need not know how to implement extraction and annotation programs, and program developers need not be domain experts.

To explain these contributions, Section 2 presents the details of our proposed instance recognition semantics and their role within the automated semantic annotation paradigm. Section 3 provides an overview of OWL with the purpose of explaining why it is insufficient to specify enough

instance semantics for automated semantic annotation. Section 4 introduces OWL-AA, and Section 5 describes the prototype system and experiments we have done to show the effectiveness of OWL-AA. Finally, we discuss related work in Section 6 and conclude the paper in Section 7.

2 Instance Recognition Semantics and Automated Semantic Annotation

2.1 Instance Recognition Semantics and Declarative Representation

Instance recognition semantics (IRecogniS) are machine-processable recognizers of instances that belong to the extention of a formal declaration, which usually could be an ontology concept. Normally, when people present the extention of an ontology concept, they list a set of data instances (setting A-Box). IRecogniS, however, present the ways of interpreting data instantiations of an ontology concept.

Despite of its name, people have used instance recognition semantics for decades to do data extraction and the other data recognition related work. Most commonly, software developers encode IRecogniS within programming procedures. The text analysis engines (TAE) in the IBM UIMA (Unstructured Information Management Architecture) project are typical examples [15]. Each TAE is a small reusable programming procedure that do a particular data analysis on source documents, for example, pulling out chemical names and their interactions, or identify events, locations, products, opinions about products, problems, methods etc. Users can aggregate multiple TAEs to achieve complicated document-level analysis requirements. Besides UIMA, many traditional data extraction tools also contain either manually programmed or machine-learned procedures to recognize data instances for individual extraction categories.

A significant characteristic of all these IRecogniS representations is that they are declared in procedures. Procedural representations of IRecogniS, or we simply call it *Procedural IRecogniS*, are hard to build, share and reuse. As we know, a complete automated semantic annotation process involves two types of very different knowledge formalizations. First, it requires formal declarations of IRecogniS, which should be created by domain experts. Second, it requires carefully implemented annotating programs to process IRecogniS, which should be coded by software developers. In general, these two groups of experts are not overlapped. We cannot expect every domain expert to be a professional software developer; and neither can we expect a professional software developer to be also an expert on many different domains. One major problem of using procedural IRecogniS is that it mixes knowledge specifications and knowledge processing. Hence it increases the difficulty of creating high quality procedural IRecogniS because it thus requires the two mentioned group of experts to closely cooperate together.

Despite of this cooperation problem, procedural IRecogniS lead to more troubles on knowledge sharing and communication. A procedure written in one programming language is generally not interpretable by a party that uses another programming language. Even if two parties use a same programming language such as C++, they may still have to re-compile procedure code when they are under different platforms such as Windows or Unix. This heterogeneity problem with programming languages and platforms makes it difficult to share procedural IRecogniS through the web.

Less degree of shareability also leads to less degree of reusability. To reuse existing procedural IRecogniS, a user must carefully identify the original executional environment that the procedural IRecogniS are based, beyond the checking of an appropriate application domain. A very undesirable, but very much possible, problem is that a user cannot directly use some existing procedural IRecogniS in spite that they are appropriate for the application domain and correctly formalized. The reason is that the user may have to recode these procedural IRecogniS according to his own execution environment. And as we all know, such a kind of re-implementation is not trivial.

The last but not the least, it is not easy to make procedural IRecogniS machine-interpretable. Even after we have implemented procedural IRecogniS by platform-independent programming languages, such as Java, and stored them properly in an organized factory, they are, in essence, still not machine-interpretable. The TAEs in UIMA are typical examples. According to Ferrucci and Lally [9], they expected to reuse TAEs through composition so as to reduce redundant effort. They, however, also pointed out that such an "analysis engine assembler" was human-guided. Professional experts, who possibly know both the application domain and Java programming, need to decide how to select appropriate TAEs from a TAE factory and manually compose them.

All of these discussions lead to a perspective: we need declarative representations of IRecogniS, or *declarative IRecogniS*, for automated semantic annotation. Declarative IRecogniS are independent to procedures. Domain experts can use declarative languages to present IRecogniS without the need of knowing how to process them by implemented programs. Similarly, software developers can implement standard IRecogniS processors by any programming languages without the need of knowing which domain is going to process. Comparing to procedures, declarative IRecogniS are easy to share and reuse through the web.

Nevertheless, when such a declarative language is a type of ontology language, these declarative IRecogniS become machine-interpretable.

2.2 Declarative Instance Recognition Semantics for Automated Annotation

Our goal is to separate the specifications of domain-specific knowledge from the specifications of domain-independent knowledge. As we have seen, two different groups of experts must cooperate closely to build procedural IRecogniS. The underlying reason is that IRecogniS are domain-specific, and thus these procedures are domain-specific. Software developers must consult with domain experts to create IRecogniS procedures. This analysis give us the hint that we must make declarative IRecogniS take care of all the required domain-specific knowledge specification issues for automated semantic annotation. Then the procedures of IRecogniS processing become totally domain-independent. In such a new paradigm, the work of domain experts and software developers becomes independent to each other.

Data frame in information extraction ontology is an example of best practice about declarative IRecogniS. We have shown successful practices of using information extraction ontologies with data frames for automated information extraction [8] and for automated semantic annotation [7]. Therefore, starting with the discussion of data frame, we present the necessary intent of declarative IRecogniS for automated semantic annotation.

Figure 1 shows an example of presenting IRecogniS for the concepts *Price* and *Make* in the domain of car advertisements in the format of data frames used by data-extraction ontologies [8]. As Figure 1 shows, we use regular expressions to capture external representations. The *Price* data frame, for example, captures instances of this concept such as "4500" and "17,900". The left context phrase shows that it may has "$" directly adhere to the left of the extracted data. So "$4500" has a higher confidence to become a price. Although not shown in this figure, we can specify right context phrases as well as the left context phrase. A data frame's context keywords are also regular expressions. The *Price* data frame in Figure 1, for example, includes context keywords such as "price," "asking," "obo," and "negotiable". In the context of one of these keywords, if a number appears, it is likely that this number is a price (e.g. "asking 17,900 obo). Sometimes external representations are best described by lexicons or other reference sets. These lexicons or reference sets are also regular expressions, often simple lists of possible external representations, and can be used in place of or in combination with regular expressions. In Figure 1, *CarMake.lexicon* is a lexicon of car makes, which would include, for example, "Toyota", "Honda", and "Nissan" and potentially also misspellings (e.g. "Volkswagon") and abbreviations (e.g. "Chev" and "Chevy").

Price
  external representation: \d+ | \d?\d?\d,\d\d\d
  left context phrases: \$?
  context keywords: price | asking | obo | neg(\.|otiable)
  ...
end

Make
  external representation: CarMake.lexicon
  ...
end

Figure 1. Sample (partial) Declarative Instance Recognition Semantics for `Price` and `Make` in the domain of car ads.

This example illustrates two major components of declarative IRecogniS: external representations and contextual representations. External representation expresses extraction patterns, and contextual keywords or phrases describes typical context that helps to determine the presence of instances of a concept. It is not surprising that contextual representations are domain-specific. Nevertheless, the external representations are also domain-specific. For example, when we declare IRecogniS of a concept `Consumer Price`, its external representations are different in the domain of house-sale from which are in the domain of apartment-rental, even though both of the domains are about real-estate business. The consumer price in the former domain is rarely below $10,000 and the same defined price in the latter domain is very unlikely over $10,000. As another example, although the external representations of the concept `Name` are the same in the domain of student information and the domain of faculty introduction, their recognizing contexts are different. The `Name` in the former domain includes context keywords such as "student" and "enroll," but the `Name` in the latter domain includes context keywords such as "faculty" and "teach."

With the use of data frames in ontologies, our *Ontos* data extraction engine and the automated semantic annotator based upon *Ontos* are domain-independent [4]. Within the *Ontos* processor, we have implemented additional domain-independent knowledge to help data processing. For example, the recognition of a string overrides any recognitions of its substrings. That is, when the system simultaneously determines "Henry Ford" to be a *Person* and "Ford", which is part of the "Henry Ford," to be a *CarMake*, the system eliminates the latter one because its super-string is identified to be an instantiation of another concept. Because all

these types of knowledge are domain-independent, to create them is one-time effort by software developers. Also because they are domain-independent, software developers do not need to consult with domain experts to build them. More importantly, later on when domain experts build specifications of their interested domains, they no longer need to worry about these tedious domain-independent miscellanies. Therefore, they can pay their whole attentions to the interesting "domain-specific" knowledge specifications.

Different from many other IRecogniS representations (such as TAEs in UIMA), in data frames we do not support to declare document layout information (as Figure 1 shows). It is due to that layout is for displaying, not for describing data. We expect an annotation system based on processing declarative IRecogniS to continually work on a web page when its layout changes. A reality of current web is that web page layouts do change from time to time, and to automatically detect these changes is very difficult. Therefore, layout-independence is a preferable feature of automated semantic annotation, and our declarative IRecogniS satisfy this preference. This layout-independent property has been called *resiliency* [19].

Moreover, resiliency also means that an annotation process can automatically work correctly on new web pages only if they are for the same application domain. The layout of information does not matter, only appearance of data in expected conditions matters.

With both domain-independent and layout-independent features, our semantic annotator does hold improved degree of automation. Both of these independent properties reduce the needs for human intervention when either application domains or web page layouts change. Therefore, we may say that the use of declarative instance recognition semantics improves the degree of automation of automated semantic annotation systems. An important remaining problem is, however, that our declarative IRecogniS are not standard. In the other words, we need to make our declarative IRecogniS be compatible to semantic web standards, which is the rest of this paper.

3 Brief Overview of OWL

OWL is a standard of presenting ontologies in the semantic web [23]. Knowledge specification in OWL ontologies is very declarative. OWL provide varied syntaxes to specify rich semantics in target domains. In brief, we can categorize existing OWL constructs into three types. To illustrate our points, we use a well-known "standard" OWL ontology, the Food ontology in W3C web site.[1] Figure 2 shows partial of the Food ontology in its RDF/XML style.

First, OWL provides formal constructs of specifying classes, relations, and different types of constraints. As in

[1] http://www.w3.org/TR/2003/WD-owl-guide-20030331/food.owl

```
<owl:Ontology rdf:about="">
    <rdfs:comment>Derived from ... .</rdfs:comment>
</owl:Ontology>
<owl:Class rdf:ID="ConsumableThing" />
<owl:Class rdf:ID="MealCourse">
    <rdfs:subClassOf rdf:resource="#ConsumableThing" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasFood" />
            <owl:minCardinality
                rdf:datatype="&xsd;#nonNegativeInteger">1
            </owl:minCardinality>
        </owl:Restriction> ...
    <rdfs:subClassOf>
    ...
</owl:Class>
<owl:Class rdf:ID="Fish">
    <rdfs:subClassOf rdf:resource="#Seafood" />
    ...
</owl:Class>
<owl:Class rdf:ID="BlandFish">
    <rdfs:subClassOf rdf:resource="#Fish" />
    <owl:disjointWith rdf:resource="#NonBlandFish" />
</owl:Class>
...
<owl:ObjectProperty rdf:ID="hasFood">
    <rdfs:domain rdf:resource="#MealCourse" />
    <rdfs:range rdf:resource="#EdibleThing" />
</owl:ObjectProperty>
...
<BlandFish rdf:ID="Halibut" />
<BlandFish rdf:ID="Flounder" />
<NonBlandFish rdf:ID="Swordfish" />
```

Figure 2. Sample food.owl (adopted from W3C web site, slightly reformatted).

Figure 2, an OWL class can be as simple as just a declaration like *ConsumableThing*, or it can include detailed specifications of its related properties and constraints like *MealCourse*. A relation in OWL can be a property, like *hasFood*, with specified domains and ranges. Or it can be a hierarchical relationship such as *Fish* is *rdfs:subclassOf Seafood*. OWL also provides multiple specifications of constraints on relations, which are called *Restrictions*. In our example, we show a cardinality *Restriction* on the property *hasFood* that shows when *MealCourse hasFood*, it has at least one *EdibleThing*.

Besides domain description, or someone call it the T-Box specifications, the second type of specifications OWL supports is data instance binding with ontology concepts, which someone also call it the A-Box specifications. In

our Figure, both *Halibut* and *Flounder* are *BlandFish*; and *Swordfish* is *NonBlandFish*.

At last, OWL supports human to put human-interpretable comments into ontologies. These comments are for human readers to understand an ontology. They are, however, essentially not machine-interpretable. In Figure 2, the second line, for example, explains a brief history about from where this Food ontology is derived.

With this brief classification of OWL declarations, we can see that OWL only provide very limited supports of declarative IRecogniS. By using OWL, we can assign individuals to classes. But there are no formal constructs that allow us to specify a recognizer for a class. Moreover, although Figure 2 does show that OWL supports primary XML datatypes, and even enumerated datatypes (not shown in this figure), they are not enough for the purpose of automated semantic annotation. Some people may argue that whether we can use *rdfs:comment*, as in the second line, to declare IRecogniS, since in general we can put whatever we want inside *rdfs:comment*. This is technically applicable, but semantically incorrect. *rdfs:comment* is for human-readable comments, and it has been formally defined to hold this meaning. If we put IRecogniS inside this tag, machines are not going to distinguish an IRecogniS declaration from a normal comment. Therefore, we need an extension of OWL for automated annotation, which is OWL-AA.

4 OWL-AA: OWL for Automated Annotation

There are three designing perspectives on OWL-AA. First, it must be fully compatible to the current OWL declarations. It means that any interpretations in OWL-AA are not going to be conflict to existing OWL declarations. It also means that the new OWL-AA representations should follow the representation style of normal OWL. Second, its declarations should be easily attachable and detachable. That is, the addition of OWL-AA will not affect any existing OWL declarations. Our projection is that we can add OWL-AA declarations into any existing OWL ontologies without deviating any original interpretations. Also, by simply detaching OWL-AA declarations, any OWL-AA ontology becomes a valid normal OWL ontology. Third, it should not introduce additional complexity of decidability. As we know, a major designing problem of OWL is its decidability issue. To resolve it, researchers have proposed three sublanguages, namely OWL-Full, OWL-DL, and OWL-Lite, that are compromised to different degree of computational complexity. We do not want to increase the language's degree of computational complexity because of OWL-AA.

With the three perspectives, we define the RDF schema for the OWL-AA extension as Figure 3 shows. At the beginning, we specify three XML namespaces used in the

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2001/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#">
  <rdfs:Class rdf:ID="ExternalRepresentation"/>
  <rdfs:Class rdf:ID="ContextualRepresentation"/>
  <rdfs:Class rdf:ID="RegularExpression">
    <rdfs:subClassOf
      rdf:resource="#ExternalRepresentation"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="LexiconList">
    <rdfs:subClassOf
      rdf:resource="#ExternalRepresentation"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="ContextPhrase">
    <rdfs:subClassOf
      rdf:resource="#ContextualRepresentation"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="ContextKeyword">
    <rdfs:subClassOf
      rdf:resource="#ContextualRepresentation"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="LeftContextPhrase">
    <rdfs:subClassOf rdf:resource="#ContextPhrase"/>
  </rdfs:Class>
  <rdfs:Class rdf:ID="RightContextPhrase">
    <rdfs:subClassOf rdf:resource="#ContextPhrase"/>
  </rdfs:Class>
  <rdf:Property rdf:ID="ExtractionPattern">
    <rdfs:domain rdf:resource="owl:Class"/>
    <rdfs:range rdf:resource="#ExternalRepresentation"/>
  </rdf:Property>
  <rdf:Property rdf:ID="ExtractionContext">
    <rdfs:domain rdf:resource="#ExternalRepresentation"/>
    <rdfs:range rdf:resource="#ContextPhrase"/>
  </rdf:Property>
  <rdf:Property rdf:ID="ExtractionKeyword">
    <rdfs:domain rdf:resource="#ExternalRepresentation"/>
    <rdfs:range rdf:resource="#ContextKeyword"/>
  </rdf:Property>
</rdf:RDF>
```

Figure 3. RDF Schema Defines Extension of OWL.

definition: rdf, rdfs, and owl. According to our discussions of data frame in Section 2, we declare two basic rdfs classes *ExternalRepresentation* and *ContextualRepresentation* to be the main body of IRecogniS declarations. *ExternalRepresentation* has two subclasses, which are *RegularExpression* and *LexiconList*. Accordingly, they represent the types of external representations as the examples of *Price* and *Make* in Figure 1. *ContextualRepresentation* also has two subclasses, which are *ContextPhrase* and *ContextKeyword*. They, respectively, present the context phrases and context keywords as Figure 1 shows. In detail, as we mentioned before, we distinguish left context phrases and right context phrases. So they are the subclasses of *ContextPhrase*.

In this definition, we also defined three properties. As Figure 3 shows, for each *owl:Class*, we can declare *ExternalRepresentation* as its *ExtractionPattern*. For each *ExternalRepresentation*, we can declare *ContextPhrase* as its *ExtractionContext*, and *ContextKeyword* as its *ExtractionKeyword*.

Figure 4 shows the instance recognition semantics in Figure 1 declared by OWL-AA. To be able to use formal OWL-AA definitions, first we need to add an *owlaa* namespace declaration at the beginning of an ontology file, as the first line in Figure 4. Originally, we use *owl:Class* to declare a class in a normal OWL ontology. So do we using OWL-AA. The only difference is that we now need to add a new property of *owlaa:ExtractionPattern* within the declaration of a class if an ontology developer decides to add IRecogniS declarations for this class. In our example, the external representations of *Price* is stored in a new class *PriceER*. Note that although the declaration of *PriceER* is totally within the scope of OWL-AA, most of its representations, as Figure 4 shows, are normal OWL representations. As we have already known, *PriceER* should be presented as *owlaa:RegularExpression*. Also, it may have multiple *owlaa:ExtractionContext* and *owlaa:ExtractionKeyword* properties. Following the example, we specify the details of both external representations and contextual information of *Price*. They have exactly the same semantic meanings as Figure 1 shows.

Similarly, we can declare IRecogniS for *Make* in Figure 1. As in the end of Figure 4 shows, the only difference from the declared IRecogniS for *Price* is that the external representation of *Make* is from a dictionary file instead of regular expressions. So when we declare an instance of *MakeER*, we present the URL of the dictionary file, instead of exact values.

A special addition is the internal datatypes of external representations and context representations. In the figure, we declared *ER-value*, *CT-value*, and *KW-value*. All of them are bound to XML String datatype. We can, however, omit these declarations and simplify the presentations be-

```
<rdf:RDF ... xmlns:owlaa="http://www.deg.byu.edu/owlaa-rdfs#">
  <owl:Class rdf:ID="Price">
    ... (standard OWL declaration of owl:Class Price)
    <rdfs:subClassOf> <owl:Restriction>
      <owl:onProperty rdf:resource="owlaa:ExtractionPattern" />
      <owl:allValuesFrom rdf:resource="PriceER" />
    </owl:Restriction> </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="PriceER"/>
    <rdfs:subClassOf rdf:resource="owlaa:RegularExpression"/>
    <rdfs:subClassOf> <owl:Restriction>
      <owl:onProperty rdf:resource="owlaa:ExtractionContext" />
      <owl:someValuesFrom rdf:resource="PriceER-CT-L" />
    </owl:Restriction> </rdfs:subClassOf>
    <rdfs:subClassOf> <owl:Restriction>
      <owl:onProperty rdf:resource="owlaa:ExtractionKeyword" />
      <owl:allValuesFrom rdf:resource="PriceER-KW" />
    </owl:Restriction> </rdfs:subClassOf>
    ...
  </owl:Class>
  <owl:Class rdf:ID="PriceER-CT-L"/>
    <rdfs:subClassOf rdf:resource="owlaa:LeftContextPhrase"/>
  </owl:Class>
  <owl:Class rdf:ID="PriceER-KW"/>
    <rdfs:subClassOf rdf:resource="owlaa:ContextKeywor"/>
  </owl:Class>
  <owl:DatatypeProperty rdf:about="ER-value">
    <rdfs:domain rdf:resource="owlaa:ExternalRepresentation"/>
    <rdfs:range rdf:resource="&xsd;#string"/> </...>
  <owl:DatatypeProperty rdf:about="CT-value">
    <rdfs:domain rdf:resource="owlaa:ContextPhrase"/>
    <rdfs:range rdf:resource="&xsd;#string"/> </...>
  <owl:DatatypeProperty rdf:about="KW-value">
    <rdfs:domain rdf:resource="owlaa:ContextKeyword"/>
    <rdfs:range rdf:resource="&xsd;#string"/> </...>
  <PriceER rdf:ID="PriceER-1"/>
    <ER-value rdf:datatype="&xsd;#string"/>
      \d+ | \d?\d?\d,\d\d\d</ER-value>
    <owlaa:ExtractionContext rdf:resource="#PriceER-CT-L-1"/>
    <owlaa:ExtractionKeyword rdf:resource="#PriceER-KW-1"/>
  </PriceER>
  <PriceER-CT-L rdf:ID="PriceER-CT-L-1">
    <CT-value rdf:datatype="&xsd;#string"/>
      \$?</CT-value> </...>
  <PriceER-KW rdf:ID="PriceER-KW-1">
    <KW-value rdf:datatype="&xsd;#string"/>
      price | asking | obo | neg(\.|otiable)</KW-value> </...>
  <owl:Class rdf:ID="MakeER"/>
    <rdfs:subClassOf rdf:resource="owlaa:LexiconList"/>
  </owl:Class>
  <MakeER rdf:ID="MakeER-1"/>
    <ER-value rdf:datatype="&xsd;#string"/>
    CarMake.lexicon</ER-value>
  </MakeER>
</rdf:RDF>
```

Figure 4. IRecogniS Declaration of Figure 1 in OWL-AA.

cause currently all of them are simple strings. But we finally decide to present them in this way because we want to show that OWL-AA is able to support other, possibly more complicated, datatypes as well. Although based on our practice, using regular expressions to represent extraction patterns works well. We do not object to use any other complicated object datatypes to represent extraction patterns or contextual patterns. On the contrary, we believe that complicated datatypes of extraction patterns could improve the accuracies of data recognition results.

Finally, we do a quick review of the three perspectives we mentioned at the beginning. This OWL-AA presentation is fully compatible to current OWL. We have added a few new formal constructs without modifying any existing ones. Also, these new constructs are defined upon RDFS on which OWL is defined. So there is no problem seamlessly put them together.

Even more important, as Figure 4 shows, the declaration of IRecogniS using OWL-AA is completely detachable. Figure 4 shows only the OWL-AA part of a complete ontology. For users who do not require IRecogniS information and believe that the IRecogniS declarations increase the burden of knowledge sharing, it is straightforward to remove all the OWL-AA declarations and they will get a plain OWL ontology as usual. Similarly, when users do expect to add IRecogniS declarations in their existing OWL ontology, it is quite easy because they do not need to make a change of any lines of their existing OWL ontology.

As a direct consequence of this easy-to-detach property, OWL-AA does not introduce any new decidability complexity into OWL. OWL-AA additions is an independent attachment to a normal OWL ontology. Hence any existing OWL inference engine can process an OWL-AA ontology just as processing a normal OWL ontology. OWL-AA, however, does provide richer semantics that could be helpful to derive more delicate inferential conclusions. We are going to explore some of its usage beyond automated annotation in Section 6.

When we only focus on the OWL-AA attachment part itself, it is still fully decidable. In the context of this paper, we are not going to present a theoretic proof on its decidability. As a simple explanation, both of our external representations and contextual representations can be presented in regular expressions, which are basically finite automata. And we know that we can decide the output of a finite automata in polynomial time.

5 Automated Annotation System

This work is a part of a big project of build a practical, automated semantic annotation system. Figure 5 shows the architecture of the entire system. Our projection of such a system is to make it annotate web pages in a domain-independent and layout-independent way, while we expect the system to have high-speed performance when processing large size tasks. Also, the system is going to provide facilities for users so that they are able to assemble unique domain ontologies for their own special applications. A more details description of the entire system can be found in [6] for readers who are interested in this project.

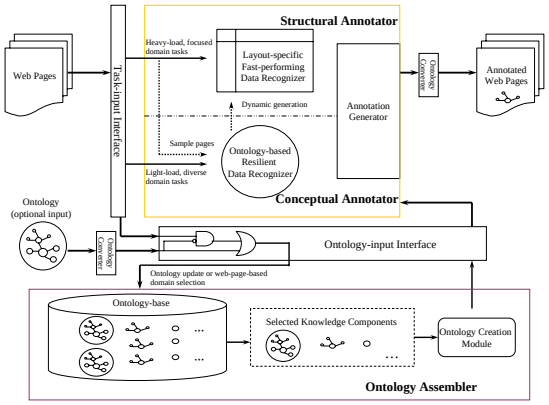

Figure 5. System Architecture

An important part of this semantic annotator project is the ontology converter (as Figure 5 shows). The converter is to do transformation between OWL-AA representations and OSMX representations, which is what we discussed in Section 2. Our adopted ontology-based data recognizer takes OSMX ontologies as its input, and so does our annotator. But OSMX is not a standard, and so a system based on OSMX ontologies is difficult to be used by normal semantic web users. The ontology converter is to resolve this problem. Since OWL-AA is full compatible to OWL and its presentations is totally OWL-style, it is very straightforward for normal OWL users to augment any existing OWL ontologies to be OWL-AA ontologies. Through the ontology converter, our system therefore can take OWL-AA, besides OSMX, ontologies as its input. Also, on the output side, the ontology converter will convert OSMX representations to OWL representations. Therefore, our annotation results are going to be presented with respect to standard OWL ontologies instead of OSMX ontologies. This change also makes our annotation results be more useable by normal semantic web consumers.

Except of IRecogniS declarations, an OSMX ontology [7, 8] is semantically equivalent to an OWL ontology, although syntactically they are quite different from each other. For example, both of the languages construct a graph of domain knowledge with classes (or object sets in OSMX), properties (or relationship sets in OSMX), restrictions (such as participation constraints in OSMX), and hierarchical declarations (such as is-a declaration in OSMX).

A direct syntax-to-syntax transformation, however, is both tedious and error-prone. A complete syntax-to-syntax conversion must base on very carefully theoretical proof of the semantic equivalence. Otherwise, a careless ontology conversion is easy to cause the problem of information leak, i.e., the converted domain is not exactly the domain before it is converted. OWL is built upon a very delicate analysis of description logic. Unfortunately, however, we are not such experts that know every detail about the logic foundation of OWL.

To solve this problem, we have used Jena [16], a standard semantic web framework. Jena is a carefully designed semantic web project primarily by HP research. In essence, Jena can take an input of OWL ontologies and convert them to be a Jena graph. Also, it has well-designed functions that can output correct OWL ontologies based on any constructed Jena graph. This nice feature really solves our problem of avoiding information leak during ontology conversion. As long as we can correctly present our OSMX domain declarations in Jena graphs, we can promise that there should be no information leak because these Jena developers are the group of people that help to build the logic foundation of OWL.

Fortunately, our OSMX ontologies are also conceptual graphs [8], which is very much alike the conceptual graphs OWL ontologies represent. Moreover, Jena graph, by its designers' words, is not only for OWL ontologies, but also for ontologies no matter of their representing languages. We found that it was much easier for us to convert our OSMX ontologies to Jena graphs than through directly syntax-to-syntax conversion to OWL ontologies.

A problem in this conversion process is the IRecogniS declarations because neither OWL nor Jena graph has prereserved rooms for IRecogniS. But again, we have designed our OWL-AA extensions to be a fully detachable part of OWL ontologies. Hence we have added some new classes in Jena so that each named ontology class object can hold an additional data structure that contains IRecogniS declarations. This part of ontology conversion code is written by ourselves. Again, since they can be sit totally separated to the other OWL representations, our new implementation does not affect the correctness of original Jena ontology model.

6 Related Work and Discussion

There are three categories of related work: *instance recognition semantics declarations*, *extension of OWL*, and *ontology conversion*. Although the principle of IRecogniS is not new, very few explorative studies have ever done on declarative IRecogniS, or even procedural IRecogniS. Many users have coded IRecogniS into their procedures. For example, a common part of most data extraction tools is wrapper, which wraps representations of desired data in target documents [19]. These wrappers therefore present IRecogniS. In general, more or less these wrappers contain document layout information. Although processing extraction based on layouts does help to improve a system's performance of speed, it is not desirable when the focus is knowledge sharing and reusing. As we discussed earlier, the inclusion of layout information makes IRecogniS generally inapplicable to a web page from another resource because layouts are usually different with respect to different resources. Consequently, even layout-independent part of IRecogniS in these wrappers also becomes very hard to reuse since it is difficult to separate them from layout-dependent part of IRecogniS.

IBM UIMA [9] is a typical approach that has seriously addressed the issues of sharing and reusing procedural IRecogniS. Besides, machine learning researchers used to train machines to learn domain-specific data extraction rules, which belong to IRecogniS. Some of these learned rules are claimed to be applicable in different extraction tasks. So they are also examples of IRecogniS reuse. The difference between our declarative IRecogniS and all these previous attempts is that we are trying to formalize the IRecogniS declarations so that they fit better to the requirements of semantic web applications. Semantic annotation is a typical application field that we can gain benefits by using declarative IRecogniS. Nevertheless, later on in this section we are going to address several other semantic web applications to which declarative IRecogniS also contribute.

Although OWL is the current W3C recommendation of web ontology languages, researchers have pointed out that OWL is not sufficient for every semantic web work. Hence it is not surprising to see several proposed extensions of OWL. For example, Bouquet and et. al. proposed C-OWL (Context OWL) to localize content of ontologies and to support explicit ontology mappings which allow for limited and totally controlled forms of global visibility [3]. According to the authors, such a type of problem "could not otherwise be dealt with" except of C-OWL. Nevertheless, Pan and Horrocks proposed another extension of OWL, which is named OWL-Eu, to enrich OWL with customized datatypes [21]. The authors pointed out that "many potential users will not adopt OWL unless [the datatype support problem] is overcome." With our experiences, we also believe that the support of customized datatypes could improve the performance of data extraction. So we have reserved a space to declare complex datatypes when using OWL-AA, as discussed in Section 4. Our OWL-AA is not only compatible to OWL, but also automatically compatible to OWL-Eu.

Before semantic web, ontology had already been designed and used for different purposes for years. One trend of the semantic web is to uniform these different ontology

representations so that machine communications could be simpler. Hence there has been several work of ontology conversion. Among them, we are particularly interested in the conversion studies between OWL and another ontology language.

Kashyap and Borgida presented a work of representing UMLS (Unified Medical Language System) in OWL [17]. In this work, the authors showed a syntax-to-syntax ontology conversion based on a detailed theoretical analysis of their logic foundations. Although this type of theoretical proofs ensures the integrity of ontology conversion, proofs are unavoidably difficult. Hence many other ontology conversion researchers tried alternative ways to avoid such a sophisticated theoretical analysis of logic foundations. Gasevic and et. al. published a study of doing transformation between OWL and MDA (Model-Driven Architectures) through technological spaces, which are working contexts with a set of associated concepts, body of knowledge, tools, required skills, and possibilities [10]. Their implementation is based on the similarity of UML diagrams of two ontological representations and uses XSLT to perform a syntax-to-syntax transformation. Heimbigner worked on another project of transforming a specific software engineering ontology CIM (Common Information Model) into OWL [13]. He did an exhaustive syntactical analysis of both languages and performed a syntax-to-syntax conversion. Instead of through theoretical proofs to confirm the integrity of these syntactical transformations, Heimbigner processes a converted OWL ontology through Jena reasoner to ensure that it derives the same inferential results as the original CIM ontology does. This is, however, a necessary but incomplete integrity checking. In practice, however, it usually brings satisfactory results. In the same year, van Assem and et. al. presented another work of converting MeSH (Medical Subject Headings) and WordNet, two well-known thesauri, to RDF/OWL. They proposed a three-step conversion, which consists of knowledge understanding preparation, syntactical conversion, and semantic checking. The main contribution of their work is that it formalizes the generic steps people need to follow to ensure a precise ontology conversion. Finally, but not the last one, Hepp presented an approach, namely the gen/tax approach, to represent hierarchy of industrial taxonomies in OWL [14]. Hepp proposed a novel method of deriving two concepts for each taxonomy category, one reflecting the generic concept and another reflecting the taxonomy concept. With this gen/tax technology, a conversion from industrial taxonomies to OWL does not require reasoning capabilities beyond *rdfs:subClassOf*.

Although OWL-AA is proposed for automated semantic annotation, we can reduce the difficulty of several other hard semantic web problems by using OWL-AA. A key observation is that if we can think of solutions before a problem comes to reality, usually the solutions could be much easier than later. Within the semantic web scenario, almost all the other applications are based on semantic annotations. Hence the choices of semantic annotation approaches and annotation representations not only affect the annotation problem itself, but also significantly affect the performance of the other semantic web applications that depend on annotations. Declarative IRecogniS provide facilities to solve these hard problems. Two typical examples are the studies of ontology mapping and annotation integrity.

Semantic-web machine agents communicate each other through ontology mapping just like humans communicate through talking. The most significant problem on ontology mapping is that a machine agent does not know how to interpret another ontology based on its own knowledge. Earlier research has shown that IRecogniS help to achieve high accurate, automated schema and ontology mappings [**?**, **?**]. However, in general ontology mapping researchers need to reproduce these IRecogniS information for their mapping purposes because IRecogniS usually are missing, even in populated ontologies. If we can keep the used declarative IRecogniS (during annotation processes) in produced annotations through OWL-AA, they become a valuable resource that may provide great help for potential ontology mapping requests in the future.

Annotation integrity is another important problem in semantic web research. Without confident integrity checking, semantic annotations will never be accepted by serious industrial users. Declarative IRecogniS can provide help on solving this annotation integrity problem too. For example, by using OWL-AA, we would not mistakenly accept "Taurus" with its annotation to be *CarMake*, but rather correctly interpret it as a *CarModel*, in the domain of automobile. Declarative IRecogniS allow annotation consumers to verify the correct bindings of data instances to their ontological definitions. Humans, as well as ontologies armed with IRecogniS, can catch these kinds of errors, but they are likely to be hard, if not impossible, for independent machine agents to catch and resolve.

7 Concluding Remarks

Our, as well as some previous, research studies show that declarative instance recognition semantics are important for automated semantic annotation. However, current semantic web ontology language standard—OWL—does not well support IRecogniS declarations. Hence it is not a satisfactory ontology language for automated semantic annotation. The OWL-AA extension of OWL is to solve this problem. OWL-AA is full compatible to original OWL, fully attachable to and detachable from normal OWL ontologies, and it does not introduce any new complexity of decidability into OWL declarations. Our implementation and practices show

that this OWL-AA extensions work well in our automated semantic annotation system.

Declarative instance recognition semantics are very useful not only for automated annotation work. In this paper, we also discuss several external problems that OWL-AA can help to solve. For example, OWL-AA could be very much useful to deal with the problems of ontology mapping and annotation integrity.

Until now, the discussion of using declarative IRecogniS is still limited. Its usage, however, could be very explorative. In this paper, we have proposed OWL-AA. But it does not mean that this is the best representation of declarative IRecogniS everyone should follow. Instead, we want to emphasize that we should pay the most attentions to the use of IRecogniS for the semantic web practices itself. We expect that this research work, as well as our practices on exploring the use of declarative IRecogniS, is going to lead more constructive discussions of exploring the use of ontologies for the semantic web.

References

[1] L. Arlotta, V. Crescenzi, G. Mecca, and P. Merialdo, "Automatic annotation of data extracted from large web sites," *Proc. Sixth International Workshop on the Web and Databases (WebDB 2003)*, pp. 7-12, San Diego, California, June 2003.

[2] M. van Assem, M.R. Menken, G. Schreiber, J. Wielemaker, and B. Wielinga, " A Method for Converting Thesauri to RDF/OWL," *Proc. Third International Semantic Web Conference (ISWC 2004)*, pp. 17-31, Hiroshima, Japan, November 2004.

[3] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt, "Contextualizing Ontologies," *Journal of Web Semantics*, vol. 1, no. 4, pp. 325–343, October 2004.

[4] Data Extraction Research Group, Brigham Young University, URL: http://www.deg.byu.edu/.

[5] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, K.S. McCurley, S. Rajagopalan, A. Tomkins, J.A. Tomlin, and J.Y. Zien, "A Case for Automated Large Scale Semantic Annotations," *Journal of Web Semantics*, vol. 1, no. 1, pp. 115–132, December 2003.

[6] Y. Ding and D.W. Embley, "Using Data-Extraction Ontologies to Foster Automating Semantic Annotation," *Proc. PhD Workshop in 22nd International Conference on Data Engineering (ICDE 2006)*, Atlanta, Georgia, April 2006.

[7] Y. Ding, D.W. Embley, and S.W. Liddle, "Automatic Creation and Simplified Querying of Semantic Web Content: An Approach Based on Information-Extraction Ontologies," 2006. (submitted to review)

[8] D.W. Embley, D.M. Campbell, Y.S. Jiang, S.W. Liddle, D.W. Lonsdale, Y.-K. Ng, and R.D. Smith, "Conceptual-model-based data extraction from multiple-record web pages," *Data & Knowledge Engineering*, vol. 31, no. 3, pp. 227-251, November 1999.

[9] D. Ferrucci and A. Lally, "Building an example application with the Unstructured Information Management Architecture," *IBM Systems Journal*, vol. 43, No. 3, pp. 455–475, March 2004.

[10] D. Gasevic, D. Djuric, V. Devedzic, and V. Damjanovic, "Approaching OWL and MDA Through Technological Spaces," *Proc. 3rd Workshop in Software Model Engineering (WiSME2004) in conjunction with UML-2004*, Lisbon, Portugal, October 2004.

[11] T.R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199-220, 1993.

[12] S. Handschuh, S. Staab, and F. Ciravegna, "S-CREAM Semi-automatic CREAtion of Metadata," *Proc. European Conference on Knowledge Acquisition and Management (EKAW-2002)*, pp. 358–372, Madrid, Spain, October, 2002.

[13] D. Heimbigner, "DMTF - CIM to OWL: A Case Study in Ontology Conversion," *Proc. Ontology in Action Workshop in conjunction with SEKE'04*, Banff, Alberta, Canada, June 2004.

[14] M. Hepp, "Representing the Hierarchy of Industrial Taxonomies in OWL: The gen/tax Approach," *Proc. ISWC Workshop on Semantic Web Case Studies and Best Practices for eBusiness (SWCASE'05)*, pp. 49–56, Galway, Irland, November 2005.

[15] IBM Research, *Unstructured Information Management Architecture (UIMA)*, URL: http://www.research.ibm.com/UIMA/.

[16] Jena, A Semantic Web Framework for Java, URL: http://jena.sourceforge.net/.

[17] V. Kashyap and A. Borgida, "Representing the UMLS Semantic Network using OWL," *Proc. Second International Semantic Web Conference (ISWC 2003)*, pp. 1–16, Sanibel Island, Florida, October 2003.

[18] A. Kiryakov, B. Popov, I. Terziev, D. Manov, and D. Ognyanoff, “Semantic Annotation, Indexing, and Retrieval,” *Journal of Web Semantics*, vol. 2, no. 1, pp. 49–79, December 2004.

[19] A.H.F. Laender, B.A. Ribeiro-Neto, A.S. da Silva, and J.S. Teixeira, “A brief survey of web data extraction tools,” *SIGMOD Record*, vol. 31, no. 2, pp. 84-93, June 2002.

[20] S. Mukherjee, G. Yang, and I.V. Ramakrishnan, “Automatic Annotation of Content-Rich HTML Documents: Structural and Semantic Analysis,” *Proc. Second International Semantic Web Conference (ISWC 2003)*, pp. 533–549, Sanibel Island, Florida, October, 2003.

[21] J. Z. Pan and I. Horrocks, “OWL-Eu: Adding customised datatypes into OWL,” *Journal of Web Semantics*, vol. 4, no. 1, pp. 29–39, January 2006.

[22] M. Vargas-Vera, E. Motta, J. Domingue, M. Lanzoni, A. Stutt, and F. Ciravegna, “MnM: Ontology Driven Tool for Semantic Markup,” *Proc. Workshop Semantic Authoring, Annotation & Knowledge Markup (SAAKM 2002)*, pp. 43–47, Lyon, France, July, 2002.

[23] W3C (World Wide Web Consortium) *OWL Web Ontology Language Reference*, URL: http://www.w3.org/TR/owl-ref/.