Integrating Metadata and Data Syntax Translation

Adam Martini

Computer and Information Science Department
University of Oregon
Eugene, Oregon 97403, USA
martini@cs.uoregon.edu

Abstract. In syntax translation between languages, the advantage of using a single internal language as a translation mediator is a reduction of the total number of translators required. To achieve bi-directional translation between $n$ languages requires $n(n-1)$ translators without a central mediator, as opposed to $2(n+1)$ translators using an internal language. Our goal is to simplify syntax translation of metadata and data between several languages relevant to semantic web technologies by using a single internal language. We choose Web-PDDL as a translation mediator and primarily focus on syntax translation of OWL Ontologies and OWL/RDF data. Web-PDDL provides a flexible and expressive logical framework, which we can mold to metadata schema, however, we encounter several challenges related to structural differences between languages and logical assumptions. Through generalization of our methods, we describe how translation error gives rise to semantic translation, which can effect the metadata–data relationship. The final product of our efforts is a functional bi-directional translator between OWL and Web-PDDL for both ontologies and their data. We can extend this application to provide translation between any pair of languages with only a constant increase is the number of translators.

Keywords: metadata syntax translation, data syntax translation, OWL ontology syntax translation, OWL/RDF data syntax translation

1 Introduction

The primary motivation for this project is to create an efficient and accurate bi-directional translator between the OWL[5] and Web-PDDL[13]. Because OWL has become the accepted ontology language, the AIM Lab[9] requires a working translator to port OWL ontologies and their data into Web-PDDL, the internal language of their existing applications. This allows for the continued use and extension of valuable Ontology Matching and Mapping tools developed over 10 years of research by Professor Dou and his students. These tools include OntoEngine[14], OntoMap[17], and OntoGrate[15].

Many of the experiments described in the papers cited above use Web-PDDL ontologies that were manually translated from OWL/RDF or painstakingly written by hand. It is clear to see that an automated bi-directional translator is
required to make these applications more accessible, and to allow for extensive
testing of their capabilities. In response to this need, this project creates a bi-
directional translation service, which demonstrates the effectiveness of using a
single internal language as a translation mediator.

In the following paper, we identify and discuss syntax translation challenges
associated with structural discrepancies between languages. We also discuss how
these challenges may lead to semantic translation, which can effect data trans-
lation and logical inference. Section 2 addresses the historical significance of the
OWL/RDF language and work related to ontology syntax translation. Section
3 describes the process of syntax mapping between OWL and Web-PDDL and
generalizes the complications that arise. We discuss the implementation details of
the OWL to Web-PDDL Translation Project in Section 4 and the translation re-
sults in Section 4.3. Finally, we conclude the paper with a discussion about how
syntax translation effects metadata-data semantics in Section 5, and directions
for future work in Section 6.

2 Background

2.1 OWL/RDF: Brief Overview

The specifications for the RDF language as a metadata model and an extension
of XML were originally published in 1999[8]. RDF data models are similar to
entity-relationship models[18], both being built from a set of statements about
resources. These statements take the form of subject-predicate-object expres-
sions, which are known as triples. RDF triples can be seen as an object oriented
abstraction of a classic entity-attribute-value model.

The final RDF Schema (RDFS) specifications, released in 2004, provide an
ontological structure for an underlying set of RDF triples. This structure groups
resources into classes or types and gives these categories a hierarchical structure
through subclass relationships. The schema also includes property specifications,
which define class connectivity with domain/range declarations. In this way,
basic inference can be applied to the underlying RDF data. To allow for more
complex inference, attempts to embed FOL logic into RDFS eventually produced
the Ontology Web Language (OWL). OWL, technically an extension to RDFS,
was adopted as the standard ontology language for the semantic web by the
W3C[12] in 2007. Ontologies built with OWL can express formal semantics as
a subset of FOL, which provide axioms for semantic reasoners (i.e., Pellet[6]) to
generate inference.

2.2 More About Web-PDDL

Before OWL became the standard ontology language, Dou and his advisor,
Professor Drew McDermott, created Web-PDDL to overcome the complications
of embedding logic into RDF[11]. Web-PDDL, “a strongly typed first order lan-
guage with Lisp like syntax” [13, pg. 1], extends the planning domain language
Integrating Metadata and Data Syntax Translation

PDDL with the ability to express XML namespaces. This extension allows for compatibility with RDF/OWL. The result is an expressive ontology language with the ability to encode axioms in a subset of first order logic (FOL), which can interface with semantic web technologies.

2.3 Related Work In Syntax Translation

Since the first use of ontologies in computer science for knowledge representation, ontology portability has been a major concern. This is primarily due to the fact that the creation and maintenance of an ontology is expensive, therefore reusability is paramount. The ideal goal is to define terms at the knowledge level, independent of a specific representational language. The acceptance of OWL as the W3 standard is the current solution to this problem, however, no standardized translators between OWL and other metadata/data models have been established.

The issue of portability was addressed formally as early as 1993 with the proposal of Ontolingua[16] as a knowledge translation mediator. The goals and approach of this translation project are very similar to that of Ontolingua. In fact, the internal language of Ontolingua (KIF) uses a Lisp-like notation that closely resembles Web-PDDL. This early work clearly documents that full translation between languages using different logical constructs is not possible, and that there may be consequences for target applications. This constraint is apparent in translation experiments for this project as well, which we discuss in Section 5.

3 Methodology

As stated above, the primary goal of this project is the efficient syntax translation of metadata and data models between representational languages. However as noted in section 2.3, it is difficult, if not impossible, to achieve full translation between models due to logical insufficiency or structural incompatibility. In this way, incomplete syntax translation engenders semantic translation error. In the following sections, we discuss our methods for syntax translation and highlight areas of difficulty.

3.1 Metadata Translation

The process of metadata syntax translation is ordered to allow for a proper sequence of object instantiation in the resulting knowledge model. This order corresponds to a natural hierarchy found in logical models. We outline the sequence we use for metadata translation below. To clarify the translation context, we include a brief description of the structural constructs involved in each step. We follow this contextual summary by a set of translation issues related to each step, if any exist.
1. **Domain Definition** – The domain of the metadata defines what subset of universal knowledge the schema refers to. This information gives us a way to categorize metadata into sets of knowledge. Each schema has a unique URI, which helps to distinguish it from other schema within the same domain.

2. **Namespaces Definition** – Namespaces are used to avoid name conflicts. Complex knowledge models, like ontologies, can draw upon resources from any number of unique namespaces. Accurate identification of all namespaces is critical for proper resource labeling within the resulting knowledge model. Lack of clearly defined namespaces can lead to semantic translation. If a metadata or data model attempts to use a resource that does not have a clearly defined namespace, then a translator will be unable to resolve the namespace to its proper prefix. This may force the translator to ignore the statement or make inaccurate guesses based on the capabilities of the target language. We will discuss this problem in more detail in section 4.2.

3. **Class and Class Hierarchy Identification** – The primary building blocks of metadata schema are classes, which correspond to the generic concept of types or categories. These categories are used to group individuals and to restrict the domain and range of their property relationships. Class identification and subclass declarations are straightforward to translate given proper namespace definition.

4. **Property and Property Hierarchy Parsing** – Properties are used to define relationships between classes and therefore individuals of those classes. A property is first defined in the metadata schema and is then used to declare any number of relationships between individuals in the dataset. Property parsing is a challenge for languages with embedded FOL, such as OWL, because domain and range definitions can contain nested restrictions on their value sets. These restrictions are technically logical axioms, which are handled during the final translation phase.

5. **Axiom Translation** – Axiom translation can be a challenging part of the syntax translation process. Due to differences in logical representation or language specific assumptions, it is not always possible to completely or accurately translate embedded logic, even if your target language is more expressive. These challenges engender semantic translation via syntax translation error, which propagates to inaccurate logical inference and/or data translation. Typically, the more complex the logic, the higher the potential for error. The problem is exacerbated when working with origin languages that are flexible (in terms of the number of ways to define a single axiom) and highly expressive (in terms of logical complexity). Further discussion on this topic can be found in section 5.

### 3.2 Data Translation

Data syntax translation can be seen as a secondary process to metadata translation. However in practice, it is often easier, or even necessary, to separate metadata and data translation. This separation is due to the division of metadata/data models into separate files and a lack of syntactic standardization,
which defines how these files relate to each other. It is also possible that meta-
data and data models use different languages, more expressive languages for
metadata and less expressive for data (i.e., an OWL ontology with RDF data).

We see model separation and lack of syntactic standardization as a major
roadblock for translation consistency. Our experience shows that highly flexible
and expressive languages without clear Best Practice rules have more inconsis-
tent translation results. We will discuss this issue further in section 5.

When parsed separately from metadata, data syntax translation is similar to
metadata syntax translation in that you need an ordered process to insure proper
object instantiation. The following sequence of steps is used when metadata is
unavailable:

1. Domain Definition – Similar to in section 3.1
2. Namespaces Definition – Again, similar to section 3.1
3. Class and Class Hierarchy Identification – One pass through the dataset is
   sufficient to populate the class objects and simple subclass relations.
4. Extraction of Individuals – Association of individuals with their class or type
can also be accomplished during the first pass.
5. Extraction of Facts – A second pass through the dataset is necessary to
   define relationships between individuals. It is important to ignore irrelevant
   statements that relate to metadata definition. Annotation facts for human
   readability may also be ignored unless this information is required by target
   applications or if full source language data recreation is required.

When data is translated concurrently with metadata, the first three steps
above are handled during metadata syntax translation. Steps 4 and 5 of data
syntax translation can occur at any point after the property parsing phase of
metadata translation. The combination metadata and data into a single docu-
ment helps to insure that all relationships found between individuals are con-
sistent with the metadata schema. However, if complex property logic leads to
semantic translation, fact declarations may become inaccurate due to skewed or
missing axioms. Likewise, if axioms related to class hierarchy are compromised,
this may affect relationship definition between individuals in the resulting trans-
lation. We discuss the consequences of translation error and semantic translation
on data definition and inference in section 5.

3.3 Extensions to Web-PDDL

During our syntax translation between OWL to Web-PDDL we found that Web-
PDDL lacks necessary keywords to represent certain logical restrictions. In par-
ticular, keywords defining cardinality. OWL uses the keywords “cardinality”,
“minCardinality”, and “maxCardinality” to restrict the range of a given prop-
erty. These words are added as global predicates to the Web-PDDL language.
When translating into target languages from Web-PDDL, special handling is used
to address these restriction predicates.
4 OWL To Web-PDDL Syntax Translation Project

This section presents the details of work thus far on the OWL to Web-PDDL Syntax Translation Project. Details about the testing corpus used are presented in section 4.1. Section 4.2 presents the implementation details for both metadata and data syntax translation. The translation results are presented in section 4.3.

4.1 Metadata and Data Corpus

The OWL ontologies used for this project are sourced from the web. One primary resource is the Protege Ontology Library[10], which provides ontologies of varying complexity. We chose simple ontologies for initial testing and gradually increased complexity while expanding the software’s capabilities. We extensively tested and debugged a total of 11 OWL ontologies.

After 2-way metadata translation was complete, we searched the web for OWL/RDF data to use for data syntax translation. It was surprisingly difficult to find recently created or updated RDF datasets. The DataSetRDFDumps page [2] available through the W3C has traditionally been a primary resource for RDF data. Unfortunately, the majority of the links on this page are currently broken. Our experience of working with datasets that are currently available may shine light on the significance of the apparent lack of actively maintained RDF datasets; a topic we cover briefly in section 5. We chose two recently updated datasets from the DataDumps site that we feel are good representations of modern OWL/RDF dataset syntax.

4.2 Implementation Details

All code for this project is written in Java using Apache Jena[1] to parse and output OWL/RDF documents. In this way, the first hurdle we faced in this project was learning how to use Jena to perform deep parsing of an ontology. Concurrent with this task was beginning to understand what OWL/RDF grammar is capable of communicating so that the information could be intelligently manipulated. It is fair to say there was a steep learning curve for this stage of the project, which was a clear indication of the complexity of this task.

The following subsections will discuss the implementation details of the metadata and data syntax translation. An outline of the program details, along with challenges we faced and our solutions, is presented for each phase of implementation. We will use the steps outlined in section 3 to help structure the discussion. Challenges not related to these translation steps are described in sections titled Other Challenges.
Program Details – Phase one of this project was the bi-directional metadata syntax translation from OWL to Web-PDDL. We will discuss each direction of the translation individually.

OWL – Web-PDDL:

1. Domain Definition – Defining the domain name for a Web-PDDL document is a crucial first step for the translation process. Generally, the prefix of the base URI is used for this definition. The easiest way to do this is to identify the base URI among the namespaces declared in the ontology and its corresponding namespace. However, there are several ways to legally define namespaces, only one of which Jena can consistently parse correctly. This is an example of how insufficient standardization in combination with high language flexibility can lead to document error and/or negatively effect parsing. Overall, the relatively simple task of finding the name of the domain and the base URI associated with that domain proved to be a significant challenge.

   Challenges and Resolutions – We used different methods based on cause to solve this issue:
   (a) Missing base URI due to naming convention – If Jena is unable to identify one of the namespaces as a base URI/prefix pair, then we need to try clever ways of locating this information. Fortunately, the filename (less the suffix) very often is identical to, or contains, the prefix name. Comparing variations of the filename to prefixes mapped to namespaces in the Jena’s OntModel is the first place to look for the missing domain name.
   (b) Missing base URI due to non-standard namespace declaration – Using DOCTYPE syntax for namespace declarations is a common technique used to make an ontology header more human readable. Unfortunately, this hinders Jena’s ability to identify the base URI. Our solution is to use JDOM [3] to parse the DOCTYPE header into a String, which we iterate through in an attempt to match the filename to potential prefixes.
   (c) Base URI found but without matching prefix – In this case, an attempt is made to resolve the missing prefix based on the NsPrefix map in the OntModel. However at this point, the actual prefix name is not important. Knowing the base URI is enough information to map resources with this namespace to whatever prefix we choose for the domain.

2. Namespaces Definition – Namespace conventions for OWL ontologies are well defined and straightforward. These conventions recommend that all namespaces used in an ontology are declared explicitly with a matching prefix. As a result, there was no problem getting a set of namespace prefix pairs from the Jena model for the ontologies we tested.

3. Class and Class Hierarchy Identification – This process involves two iterations through Classes defined in the ontology. During the first pass the Classes are translated into Web-PDDL Types. The direct super class relationships are added during the second pass. While iterating through the classes, classes defined to as intersections, unions, restrictions and/or equivalent classes are marked as needing further parsing during axiom translation.
4. **Property and Property Hierarchy Parsing** – This process is complex because each property requires extensive case checking to correctly identify its domain and range. This complexity is related to poor standardization of the OWL language. We outline the most challenging situations below. Properties declared as symmetric, transitive, inverseOf, and functional are marked for future handling.

![Diagram of disjointWith declarations](image)

**Challenges and Resolutions**

(a) Missing both domain and range declarations – If a property is missing domain and range definitions, it may have an inverse property with domain and range declarations, or it may be a hierarchical property with. If the property has an inverse property, case checking is required to find the appropriate domain and range. If no inverse property is found, then the property is created using the universal “Object” class for both the domain and range.

(b) Restriction class in the domain or range – We solve this problem with an attempt to declaring the property in its standard form by extracting the class from within the restriction. We then attempt to create an axiom that defines the restriction during the axiom translation step.

5. **Axiom Translation** – During this translation step, an attempt is made to resolve the complex logic constraints of each marked class and property into axioms. Deeply nested logic presents the greatest challenge because it is difficult to develop recursive handling that generalizes well between grammars. In some cases axioms cannot be translated correctly. We will
describe axiom translation challenges along with our solutions (if one exists) below.

Challenges and Resolutions
(a) Variable scope with someValuesFrom, allValuesFrom and cardinality restrictions – The concept of variable scope is necessary to write correct FOL axioms. SomeValuesFrom and cardinality restrictions require the use of existential variables while allValuesFrom restrictions imply universal scope. Example 1 shows a maxCardinality restriction, which used an existential variable from the property domain. The structure is similar to that of a someValuesFrom restriction. Example 2 shows how a nested allValuesFrom restriction, which uses a secondary universal Object variable to enforce scope.

(b) Intersection classes – An intersection corresponds to a logical “AND”, which Web-PDDL supports naturally in its axioms with the conjunctive keyword “and”. Example 1 shows how we translate an intersection class composed of several internal restriction classes by conjunctively joining conclusion side atomic formulas.

(c) Restriction classes – There are several types of restriction classes defined in OWL. When handled individually, restrictions are relatively simple to translate. However, when restrictions give rise to nested logic, recursive parsing is required. In the ontologies we tested, the deepest nesting is three levels. Because our translator assumes a particular logical hierarchy during recursive handling, it is not robust to high language variation. This is unfortunate, and will continue to be a weakness in our implementation until enough variations have been identified and incorporated.

(d) Union classes – A union corresponds to a logical “OR”, which can be represented easily in Web-PDDL in most cases. Generally, OWL uses unionOf classes to define a set of ranges or domains for a given property. We translate this as individual predicate definitions for each domain/range pair in the set(s).

(e) Equivalent classes with nested logic – Equivalent classes correspond to if-and-only-if logic. Trivial equivalence between classes is expressed using the Web-PDDL symbol “T->”, but equivalence with nested logic requires FOL axioms. Web-PDDL uses the keyword “iff” to express equivalence logic as shown in example 3.

(f) oneOf Enumerations – oneOf constraints define a set of values that an instance of a class can take. The most compact way to represent this kind of statement is to negate the axiom to create its conditional normal form (CNF). Web-PDDL supports the use of logical “OR” statements in CNF axioms as shown in example 2.

(g) DisjointWith Definitions – Because Web-PDDL assumes disjunction between its type statements, we simply ignore DisjointWith declarations. When the model is translated from Web-PDDL into other languages, or back into OWL, this level of granularity is lost. This may have implications for inference over data. See figure 1 for a graphical representation of this translation.
Unresolved Challenges
(a) Nested Enumerations – Our translator fails to handle nested oneOf con-
straints. This is due to unimplemented functionality in the original Web-
PDDL parser. We can create Web-PDDL documents with nested CNF
axioms, but the Web-PDDL parser requires updating.

Example 1:
For the first example we present both the Web-PDDL and FOL translations.
For all of the following examples we give only the FOL representation for
readability.
The original OWL class definition:

```xml
<owl:Class rdf:ID="Zinfandel">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Wine" />
    <owl:Restriction>
      <owl:onProperty rdf:resource="#madeFromGrape" />
      <owl:hasValue rdf:resource="#ZinfandelGrape" />
    </owl:Restriction>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#madeFromGrape" />
      <owl:maxCardinality rdf:datatype="#xsd:nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

The resulting Web-PDDL axiom:

```
(forall ( o - Object)
  (if (@pddl:isa o Zinfandel)
    (exists ( w - WineGrape)
      (and (@pddl:isa o Wine)
        (madeFromGrape o ZinfandelGrape)
        (madeFromGrape o w)
        (@pddl:maxcardinality w 1))))
```

The resulting FOL axiom:

```
∀x Object(x) ∧ isa(x, Zinfandel) ⇒ ∃y WineGrape(y) ∧ isa(x, Wine) ∧ madeFromGrape(x, ZinfandelGrape) ∧ madeFromGrape(x, y) ∧ maxcardinality(y, 1)
```
Example 2:

The original OWL class definition:

```xml
<owl:Class rdf:about="#WhiteLoire"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="#madeFromGrape"/>
  <owl:allValuesFrom>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#CheninBlancGrape"/>
        <owl:Thing rdf:about="#PinotBlancGrape"/>
        <owl:Thing rdf:about="#SauvignonBlancGrape"/>
      </owl:oneOf>
    </owl:Class>
  </owl:allValuesFrom>
</owl:Restriction>
</owl:Class>
```

The resulting FOL axiom:

\[
\forall x, y \ (\text{Object}(x) \land \text{Object}(y)) \\
\quad \Rightarrow \ \neg \text{isa}(x, \text{WhiteLoire}) \\
\quad \lor \ (\text{madeFromGrape}(x, y) \land \text{isa}(y, \text{CheninBlancGrape})) \\
\quad \lor \ (\text{madeFromGrape}(x, y) \land \text{isa}(y, \text{PinotBlancGrape})) \\
\quad \lor \ (\text{madeFromGrape}(x, y) \land \text{isa}(y, \text{SauvignonBlancGrape}))
\]

Example 3:

The original OWL class definition:

```xml
<owl:Class rdf:ID="TinyPolarAminoAcid"/>
<owl:equivalentClass>
  <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:ID="AminoAcid"/>
      <owl:Restriction>
        <owl:onProperty rdf:ID="hasSize"/>
      </owl:Restriction>
    </owl:Intersection>
    <owl:Class rdf:ID="Tiny"/>
  </owl:Class>
</owl:equivalentClass>
```
The resulting FOL axiom:

$$\forall x \text{Object}(x) \land \text{isa}(x, \text{TinyPolarAminoAcid})$$

$$\iff \exists y, z \text{Size}(y) \land \text{Polarity}(z)$$

$$\land \text{isa}(x, \text{AminoAcid}) \land \text{hasSize}(x, y) \land \text{hasPolarity}(x, z)$$

**Example 4:**

This example shows the namespace header and initial resource statement of an OWL/RDF dataset. The lack of required metadata specification creates unnecessary ambiguity in the language. Also, notice how the initial entry does not have a corresponding namespace prefix definition. This gives rise to an unresolved issue because Web-PDDL requires a defined namespace prefix for all resources.

```xml
<rdf:RDF xmlns:code="http://telegraphis.net/ontology/measurement/code#"
         xmlns:geographis="http://telegraphis.net/ontology/geography/geography#"
         xmlns:gn="http://www.geonames.org/ontology#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <gn:Capital rdf:about="http://telegraphis.net/data/capitals/AD/Andorra_la_Vella#AndorralaVella"/>
  ...
</gn:Capital>
```

**Web-PDDL – OWL:**

The translation back into OWL from Web-PDDL was a significantly less daunting task. This was partly due to our intimate familiarity with the Jena API, and
partly due an increased understanding of the syntax translation process as a whole. However, the process was slowed due to several tangents to fix OWL – Web-PDDL bugs that became apparent while viewing the original OWL file side by side with its recreation.

There were few serious problems with the implementation of this code. We will outline issues that we had alongside the processing steps.

1. **Domain Definition** – We explicitly defined the base URI and its prefix during our translation into Web-PDDL using a Web-PDDL node titled “base”. This made the extraction of the base URI trivial.

2. **Namespaces Definition** – Again, a trivial task due to Web-PDDL’s clearly defined syntax. However, we did run into trouble with the getting Jena to store and print the namespaces correctly.

**Challenges and Resolutions**

(a) Jena unable to index identical URI’s – Because Jena uses a Java Map object to store its namespace/prefix pairs, it cannot have duplicates in either set of values. This makes it impossible to explicitly add base URI multiple times with different prefixes as is the accepted namespace convention. Jena’s solution to this is to have a parameter for its OWL/RDF writer, which informs the writer what URI to use for base URI namespace conventions. Unfortunately, only Jena’s most basic XML writer generates documents true to the logic of the original ontology. As a last resort, we solve this problem by reloading the final document as a String and inserting the appropriate statements into the namespace header.

3. **Property and Property Hierarchy Parsing** – Again, no significant issues.

4. **Axiom Translation** – In general, the translation of Web-PDDL axioms back into embedded OWL logic was simple. This is largely due to the fact that the axioms were directly translated from OWL syntax. We use case checking on a set of key features to differentiate axioms into categories. These features include the number of universal and existential variables, the number of expressions on the premise and conclusion sides of the axiom, and in some cases, String matching.

We used Protege 4.2[7] to view both the original OWL file and its recreation simultaneous to find translation errors. It was during this process that we began to uncover serious problems with our original translations, which were debugged and fixed. There was only one major problem that could not be resolved.

**Unresolved Challenges**

(a) Recapturing Union Classes Unsuccessful – After breaking Union class definitions into individual statements, we were unable to identify which statements to belonged originally to unionOf constraints

5. **Other Challenges** As mentioned above, writing OWL documents correctly with Jena proved to be a challenge. The main problem was that while the pretty writer in Jena to produce more human readable documents, it did not preserve class hierarchy. Extensive testing showed that the only Jena writer
that produces accurate OWL models is the default XML writer. Unfortu-
nately, the resulting documents have low human readability.

**OWL/RDF Data – Web-PDDL Data – OWL/RDF Data**

*Program Details* The second phase of the project was bi-directional syntax trans-
lation of OWL/RDF and Web-PDDL Data. In section 3 we break this problem
into two types: 1) data translation that occurs as a subsequence of metadata
translation and 2) data translation that occurs independently from metadata
translation. These problems are handled in very different ways, so we will dis-
cuss the their implementation separately.

Data Translation In Conjunction With Metadata Translation:

*OWL/RDF Data – Web-PDDL Data:* In this problem, the successful syn-
tax translation of data is entirely dependent on successful translation of its
metadata. In fact, most of the OWL ontologies we tested contain both data
and metadata, so we handled data syntax parsing concurrently with meta-
data handling. To do this, we added the creation of Web-PDDL constants
to/from OWL individuals and the translation of Web-PDDL facts to/from
RDF triples that defined relationships between individuals.

*Web-PDDL Data – OWL/RDF Data:* Generating RDF data documents
from Web-PDDL documents that contain metadata and data was a trivial
task. Constants translate effortlessly into Individuals and the data structures
containing Facts are easily parsed into triples.

Data Translation Independent From Metadata Translation:

In contrast to single file metadata and data translation, we encountered sev-
eral problems translating data separately from its metadata. These problems
lead us to direct parsing of RDF triples for reasons explained below.

*OWL/RDF Data – Web-PDDL Data:* OWL/RDF data files have a very
loose syntax that consists of mostly RDF statements with elements of OWL
for syntactic sugar. The original OWL parser we wrote breaks when handed
documents with less rigid syntax constraints, therefore, we could not reuse
this code. There are two main reasons for this failure:

1. Jena is unable to parse the document into data structures our translator
   is expecting – The result of this problem as that we were unable to reuse
   our original translator code. Translating a datasets OWL metadata be-
   fore attempting is a partial solution to this issue. However, choosing an
   ontology to load is an inherently ambiguous task. There is no special
   syntax, such as an “imports” keyword, to define a corresponding meta-
   data schema. The namespace header in Example 4 demonstrates this
   ambiguity.
2. RDF’s allowance of URI’s that have no corresponding prefix defined in
the namespace section – This problem has no solution at this point. It
is difficult to say whether resources without prefix definitions or miss-
ing/misplaced “#” symbols for instance separation is due to human er-
or. However, the RDF grammar places no restrictions on resource con-
tents. Web-PDDL does not allow for resources without valid namespace
prefixes and therefore cannot represent resources as raw URI’s. This
is an example of a language syntax constraint that makes Web-PDDL
more consistent and easier to manage, but leads to incompatibility with
more flexible languages. Example 4 shows an OWL/RDF data file with
a resource that has no defined prefix due to a questionably placed “#”
symbol.

To work around the issues above, we fell back on direct data translation
to/from RDF triples. This approach, while simple, complicates handling of
nested logic. Because most information contained in data files is flat, recur-
sive handling and case checking for nested logic is left as future work.

Web-PDDL Data – OWL/RDF Data: As we mentioned earlier, translation
of constants and facts into RDF Data is very simple. However, one prob-
lem occurs when a Web-PDDL document contains a “(” or “)” character in
the String portion of a fact declaration. Web-PDDL relies on these charac-
ters for parsing, and extra parens causes grammar tree parsing to fail. The
real solution for this problem is to add escape characters for parens to the
Web-PDDL grammar, but our temporary solution is to parse Web-PDDL
datasets using a Java Scanner object to naively iterate through the file. The
implementation of escape characters is left for future work.

4.3 Results

Our final application can successfully translate all of the OWL ontologies and
OWL/RDF datasets from our test set into Web-PDDL and back to their original
format with only minor errors. Because our implementation lacks the exhaustive
testing necessary to become robust, the current version is considered a work in
progress. More specific information about our translation results are as follows:

Ontology Translation The Web-PDDL translation of the most complex of our
test ontologies contains 87 classes, 13 properties, and about 200 axioms. Only
about 4 constraints were left unhandled due to complexity and the original OWL
translation is almost identical to its recreation when viewed in Protege 4.2\[7\].

Data Translation For data represented within an OWL ontology, translation is
flawless. For OWL/RDF datasets translated separately from their metadata, the
translation accuracy is difficult to measure due to the size of the files and vastly
different pre/post syntax. The original files are 20MB and 70MB respectively,
each containing hundreds of thousands of triples. Bi-direction translation was
successful, however, resources without defined namespace prefixes or nested logic
were ignored.
5 Discussion

Our work demonstrates that there are three main factors that influence translation error: 1) language ambiguity, and 2) differences in the logical constructs or assumptions between languages. Each factor directly or indirectly leads to semantic translation, which affects the accuracy of the metadata–data relationship.

From our experience with the syntax translation of OWL and OWL/RDF, we argue that language ambiguity unnecessarily complicates parsing. Grammatical ambiguity is considered a poor design decision in most computer languages. In general, an unambiguous grammar facilitates language portability and minimizes potential for human error. The overall result is enforcement of language consistency and elimination of “best guess” algorithms during processing. An example of language ambiguity effecting this project is a lack of domain identification and namespace convention enforcement.

Differences in logical constructs effects syntax translation at a fundamental level. Some issues that we faced in implementation relate to inadequacies in our target language, Web-PDDL. Other issues arise from the inherent difficulty of processes sing a language with high flexibility (in terms of the number of ways to express an idea). Flexibility is also related to grammar design. In this case, the language design goals of expressiveness (in terms of logic) are achieved in OWL with a simultaneous increase in flexibility. Increased flexibility not only complicates syntax translation, but also overall usability and portability at the application level.

Differing logical assumptions between languages also effects translation decisions. These assumptions manifest as syntactic shortcuts, which express more complex semantics. The detail or granularity of semantic information is essentially lost with logical assumptions. In this project, the result is incomplete recreation of metadata in its original language.

The primary effect of handling the challenges above during syntax translation is translation error. At the metadata level, this implies undesirable semantic translation. At the data level, this implies missing facts. When syntax translation changes metadata semantics, there exists a potential for inaccurate inference over its corresponding data. It is important to reduce translation error at each translation step to avoid error propagation and semantic degradation.

6 Future Work

Listed below are several direction for future work relevant to this project. The items can be seen as ordered by importance.

1. Application Integration Our first task is to integrate the translation service into existing AIM Lab applications, such as OntoMap and OntoGrate. Use of the software with existing applications will facilitate testing and debugging.
2. *Continued Maintenance and Improvements* Maintaining and improving this project will be a continual process over the next year. The program will become more robust through testing and debugging. For data translation, extending triples translation to handle nested logic is necessary to achieve accurate data translation.

3. *More Languages* Further expansion of the translator to SQL, XLS, etc. metadata and data is needed to achieve the final vision of using Web-PDDL as a translation mediator. More translators will allow for more input flexibility for future AIM Lab projects.

4. *Annotations and Labels* Annotations and labels intended for human interpretation are yet to be translated into Web-PDDL (and therefore back to OWL). This is a low priority because it is not relevant information for our target applications.

7 Conclusion

Version 1.0 of our metadata and data syntax translation software is complete. A working version of our translation service [4] is available for testing online. While this program needs refinement before reaching maturity, it is clear that we have accomplished our goals of bi-directional translation between OWL/RDF and Web-PDDL. As we add languages to our implementation, we will be able to translate between all of the languages using a single internal language. This effectively reduces the number of translators required from \( n(n - 1) \) to \( 2(n + 1) \).

Our experience gives insight into the process of syntax translation and its challenges. We document a direct correlation between language ambiguity and differing logical constructs/assumptions, and translation error. The effect of translation error is semantic translation, which fundamentally changes the metadata–data relationship. In this way, syntax translation can promote inaccurate inference in applications that use metadata and data translations.

Acknowledgments

We would especially like to thank Professor Dejing Dou for his strong support and patient guidance throughout this project. We would also like to thank the developers of the Jena API for the hard work that goes into providing consistently good software for RDF/OWL parsing.

References

1. Apache Jena Homepage.
   http://jena.apache.org/.
2. DataSetDumps: Linked Data Sets.
   http://www.w3.org/wiki/DataSetRDFDumps.
3. JDOM Homepage.
   http://www.jdom.org/.
4. OWL to Web-PDDL Translation Service.  
   http://aimlab.cs.uoregon.edu/owl2pddl/
5. Owl web ontology language.  
   http://www.w3.org/TR/owl-features/
6. Pellet Hompage.  
   http://clarkparsia.com/pellet/
7. Protege Homepage.  
   http://protege.stanford.edu/
   http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/
9. The AIM Lab Homepage.  
   http://aimlab.cs.uoregon.edu/
10. The Protege Ontology Library.  
    http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library
11. W3 RDF Homepage.  
    http://www.w3.org/RDF/
    http://www.w3.org/