Ontology Re-Engineering: A Case Study from the Automotive Industry

Nestor Rychtyckyj and **Venkatesh Raman**
Ford Motor Company
Dearborn, MI, USA

Baskaran Sankaranarayanan,
P Sreenivasa Kumar, Deepak Khemani
Indian Institute of Technology Madras
Chennai, TN, India

Abstract

For over twenty five years Ford has been utilizing an AI-based system to manage process planning for vehicle assembly at our assembly plants around the world. The scope of the AI system, known originally as the Direct Labor Management System and now as the Global Study Process Allocation System (GSPAS), has increased over the years to include additional functionality on Ergonomics and Powertrain Assembly (Engines and Transmission plants). The knowledge about Ford's manufacturing processes is contained in an ontology originally developed using the KL-ONE representation language and methodology. To preserve the viability of the GSPAS ontology and to make it easily usable for other applications within Ford, we needed to re-engineer and convert the KL-ONE ontology into a semantic web OWL/RDF format. In this paper, we will discuss the process by which we re-engineered the existing GSPAS KL-ONE ontology and deployed Semantic Web technology in our application.

1 Introduction

The Direct Labor Management System (DLMS) (Rychtyckyj 1999) was initially developed and deployed in Ford's North American assembly plants back in the early 1990s. It was recognized that an ontology and a reasoner were required to represent the complex knowledge in the manufacturing process. This was done by creating an implementation of the KL-ONE language using the LISP programming language and developing a classifier that could reason with the ontology. This implementation turned out to be extremely successful and became the production version as the system was expanded to assembly plants first in Europe and then the rest of the world. Throughout this the KL-ONE architecture remained in place as the ontology was expanded and maintained through thousands of updates.

As the Semantic Web architecture and standards were developed it became obvious the GSPAS KL-ONE ontology would be much more usable and of better value to Ford if it could be rewritten into OWL/RDF. A semantic web ontology would be much easier to maintain and could be extended and utilized for other applications in the company. The main issue was in terms of time and resources: GSPAS was a production system with high value to the business customers and it was impossible to spare the people to redo the ontology and keep the existing system in production. An alternative solution was needed and we found it by partnering with the Indian Institute of Technology Madras (IITM) in Chennai, India. We elected to partner with IITM for several reasons. The university has an excellent reputation with a strong background in Artificial Intelligence. In addition, Ford already has significant operations in Chennai and we wanted to develop a strong relationship with the university.

The result of this project was very successful. The IITM team delivered a re-engineered OWL/RDF ontology that contained the knowledge in the existing KL-ONE ontology. The Ford team validated and updated the ontology to meet Ford's requirements and has deployed the lexical ontology into the GSPAS application. The rest of the paper is organized as follows: Section-2 will describe the structure and usage of the existing KL-ONE ontology. Section-3 describes the conversion approach and the conversion process, while Section-4 describes how the ontology was validated and then deployed into the GSPAS application. In this paper, we will refer to the GSPAS KL-ONE ontology as the GSPAS ontology or KL-ONE ontology, and refer to the new GSPAS OWL ontology as the new ontology or OWL ontology.

2 The existing KL-ONE ontology

We adapted the KL-ONE knowledge representation system during our initial development of DLMS. There were no KL-ONE tools or editors available so we built both a KL-ONE editor as well as the code for classification and reasoning (Rychtyckyj 1994). The Knowledge Base Update (KBU) module was a graphical user interface that allowed us to maintain the KL-ONE knowledge base and also performed error checking as part of the update process.

The KL-ONE knowledge representation system (Brachman and Schmolze 1985) was first developed in the late 1970's. KL-ONE was selected for use on the DLMS project because of its adaptability as well as the power of the KL-ONE classification algorithm.

The KL-ONE knowledge base as used in DLMS can be described as a network of concepts with the general concepts being closer to the root of the tree and the more specific concepts being the leaves of the tree. A concept in a KL-ONE knowledge base inherits attributes from the nodes that subsume it. The power of the KL-ONE system lies in the classification scheme. The system will place a new concept into its appropriate place in the taxonomy by utilizing the subsumption relation on the concept's attributes. A detailed description of the KL-ONE classification scheme can be found in (Schmolze and Lipkis 1983).

The existing KL-ONE ontology proved to be very robust and flexible as we made hundreds of changes to it on an annual basis. Both the business and the technology changed dramatically, but we managed to keep the system fully functional as its scope increased dramatically. However, it also became obvious that the KL-ONE framework was limiting the usefulness of the GSPAS ontology. It was difficult to extract and share knowledge with other applications because custom code was needed. The graphical user interface was rewritten several times as the application was migrated to new platforms and maintaining it was time-consuming. In the meantime semantic web technology had matured to a point where it was certainly feasible to move into this space.

3 Re-Engineering KL-ONE into OWL

The goal of this project is not only ontology translation but also redesign and restructuring, where the scope is limited to GSPAS and OWL frameworks. The GSPAS to OWL translation follows the 4-layered approach (with *lexical*, *syntactic*, *semantic* and *pragmatic* layers) from (Corcho and Gómez-Pérez 2005; Euzenat 2001). The lexical and syntactic layers, respectively, deal with character-set and KR-language syntax translation. The semantic and pragmatic layers, respectively, deal with interpretation and choice-of-modeling.

Our approach to re-engineering (redesign and translation) is shown in Fig-1. We follow a spiral development model and make several iterations through the various phases. The Framework-Mapping phase covers the semantic and pragmatic aspects of ontology translation. The lexical and syntactic transformations are implemented in the translator, we do not present the details due to space limit. The remainder of this section describes our re-engineering approach.

3.1 Study Phase

Here, the goal is to understand the GSPAS and OWL (Bechhofer et al. 2004) frameworks, their similarities and differences, and understand the use-cases, design and organization of the GSPAS ontology, and identify areas for improvement.

To accomplish this goal, the IITM team studied the GSPAS, KL-ONE, DL and OWL frameworks, and with the help of the Ford team analyzed the GSPAS ontology. Then the IITM team developed a document that presented their understanding of (*i.*) the KR frameworks, (*ii.*) a potential mapping between GSPAS and OWL, (*iii.*) the design, organization and use-cases of GSPAS ontology, and (*iv.*) a high-level approach to GSPAS ontology re-engineering.

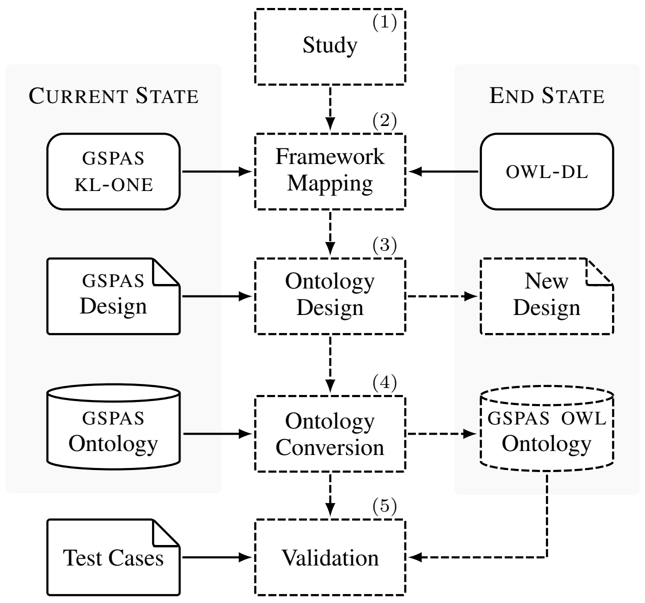

Figure 1: GSPAS to OWL conversion. Shows the current and end states of the ontology, the life-cycle of ontology re-engineering, the inputs to re-engineering (solid line), and the various phases and deliverables (dashed line).

The Ford team then reviewed the understanding-document and worked with the IITM team to validate their understanding of the ontology and to address the questions and fill in the blanks where needed. There was a significant amount of "obsolete" knowledge that was no longer needed but existed in the ontology, furthermore, ontology "cleanup" was never a high priority due to limited time and resources, so concepts like carburetors or tape decks still exist in the ontology.

3.2 Framework Mapping

Here we aim to establish a correspondence between GSPAS (a subset of KL-ONE) and OWL frameworks. So we study and compare the three elements of the frameworks, namely, *vocabulary*, *representation* and *reasoning*.

Vocabulary: GSPAS, KL-ONE, DL and OWL (though related) use different names to refer to a given idea or conceptualization. We document the various vocabularies and their correspondences, see Table-1. It also shows the GSPAS features that are (un)supported in other frameworks.

Representation: we study the KR primitives in GSPAS and determine how GSPAS ontology can be losslessly encoded in OWL. GSPAS implements only a subset of the KL-ONE framework. For each GSPAS KR primitive we find a representation in OWL, such that the subsumption relation is preserved after translation. Table-2 shows the KR primitives and their OWL translation. Here, a primitive concept is represented as a subclass axiom, a defined concept as an equivalence axiom, a classifiable attribute as an object property, a simple attribute as an annotation property, and *a value restriction as an existential restriction*.

Table 1: Vocabulary Mapping. Only terms relevant to GSPAS framework are listed here.

	GSPAS	**KL-ONE**	**DL**	**OWL**
1	Concept	Concept	Concept	Class
2	Primitive Concept	Primitive Concept	Atomic Inclusion	Partial Concept
3	Generic Concept	Defined Concept	Definition	Complete Concept
4	Individual	Individual Concept	Individual	Object
5	Classifiable Attribute	Role	Role	Object Property
6	Attribute	Non-definitional Role	*n/a*	Annotation Property
7	Inheritable Attribute	*n/a*	*n/a*	*n/a*
8	Role Restriction	Role Restriction	Role Restriction	Property Restriction
9	Value Restriction	Value Restriction	Value Restriction	Value Restriction
10	Number Restriction	Number Restriction	Number Restriction	Cardinality restriction
11	Classifier	Classifier	Reasoner	Reasoner

Table 2: GSPAS KR primitives and their OWL translation.
Where, A is concept name; $\mathrm{C}, \mathrm{C}_1, \mathrm{C}_2$ are concept expressions; R, S are role names.

	GSPAS	**KL-ONE**	**DL**	**OWL**
1	Primitive Concept	Primitive Concept	$\mathrm{A} \sqsubseteq \mathrm{C}$	`rdfs:subClassOf`
2	Generic Concept	Defined Concept	$\mathrm{A} \equiv \mathrm{C}$	`owl:equivalentClass`
3	Classifiable Attribute	Role	Role	`owl:ObjectProperty`
4	Attribute	Non-definitional Role	*n/a*	`owl:AnnotationProperty`
5	Inheritable Attribute	*n/a*	*n/a*	*n/a*
6	Value Restriction	Value Restriction	$\exists \mathrm{R.A}$	`owl:someValuesFrom`
7	Conjunction	Conjunction	$\mathrm{C}_1 \sqcap \mathrm{C}_2$	`owl:intersectionOf`
8	Sub Role	Sub Role	$\mathrm{R} \sqsubseteq \mathrm{S}$	`rdfs:subPropertyOf`

In GSPAS ontology, roles are functional and value restrictions can be seen as $\forall R.A \sqcap \exists R$. In the new ontology, we remodel the value restriction as existential restriction $\exists R.A$ (note that $\forall R.A \sqcap \exists R \sqsubseteq \exists R.A$). This helps to preserve the subsumption relation and also makes the new ontology more tractable from a computational complexity point of view.

In Table-2, inheritable attributes are non-definitional but are inherited by subclasses, this is a GSPAS specific feature, neither KL-ONE nor OWL support this type of attribute directly. In the new ontology, for pragmatic reasons, we model inheritable attributes as annotation properties (without inheritance) and allow the application that uses the ontology to handle attribute inheritance.

Reasoning: we show how subsumptions in the GSPAS ontology are preserved in the new ontology. The GSPAS classifier (a derivative of the KL-ONE classifier) uses structure-matching to compute subsumption relation, whereas, OWL reasoners use logic-based tableau algorithms for this purpose. It is known that structural subsumption is sound but incomplete with respect to logical subsumption (Baader et al. 2003), i.e., for a knowledge base, logical subsumption will find all inferences that structural subsumption can find and possibly more; let us denote this as property P_1.

Now, the profile of GSPAS (a subset of KL-ONE) is:

$$A \sqsubseteq C; \quad A \equiv C; \qquad \text{axioms} \quad (1a)$$

$$C \rightarrow A \mid \forall R.A \sqcap \exists R \mid C_1 \sqcap C_2; \qquad \text{constructors} \quad (1b)$$

The profile of the new (translated) ontology is:

$$A \sqsubseteq C; \quad A \equiv C; \quad R \sqsubseteq S; \qquad \text{axioms} \quad (2a)$$

$$\text{domain}(R) \sqsubseteq C; \quad \text{range}(R) \sqsubseteq C; \qquad \text{axioms} \quad (2b)$$

$$C \rightarrow A \mid \exists R.A \mid C_1 \sqcap C_2; \qquad \text{constructors} \quad (2c)$$

where, A is concept name; C, C_1, C_2 are concept expressions; R, S are role names.

Observe that models of Eqn-1, when value restriction is translated into existential restriction, are models of Eqn-2. From property P_1 and ($\forall R.A \sqcap \exists R \sqsubseteq \exists R.A$), we can conclude that the subsumptions in GSPAS ontology will be preserved in the new ontology. Furthermore, Eqn-2 forms a sub-language of DL called $\mathcal{EL}^{++}$ (Baader, Brandt, and Lutz 2008; Motik et al. 2012) which runs in polynomial time for common reasoning tasks. Thereby, the new ontology stays well within the OWL-DL subset.

We experimented with other DL profiles and selected $\mathcal{EL}^{++}$ because it provides a good balance between expressiveness and performance for the current ontology requirements.

3.3 Ontology Design and Organization

The existing GSPAS ontology was designed to support two use cases, namely, to parse vehicle-build-instructions written in standard language (Rychtyckyj 2006), and to interpret (assign meaning to) parsed instructions. As a result of this, there are two sets of terminology in the ontology—one that describes words in the standard language and the other that describes build-instructions, parts, tools, etc.

The new design aims to organize the ontology in a modular fashion, i.e., keep related terms together and unrelated terms apart. Accordingly, the new ontology is broadly divided into language and manufacturing terms (Fig-2). Each of this is further divided into smaller areas (like verbs, parts, tools, etc.), and so on to arbitrary depth.

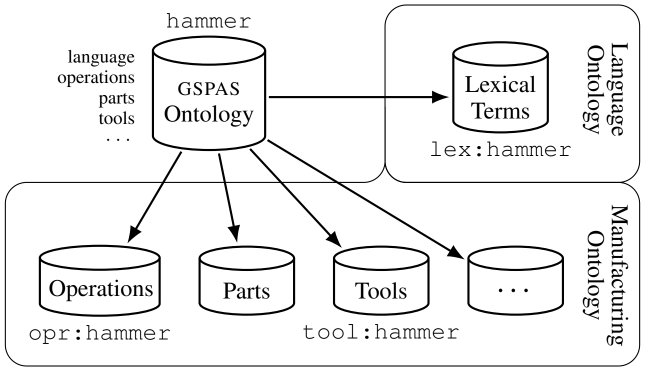

Figure 2: Re-engineered Ontology.

Fig-2 highlights the differences between the GSPAS ontology and the new ontology. The new ontology is a collection of terms.[1] Every term, apart from its description, is assigned a namespace, a label and a unique identifier. The unique identifier is generated from namespace and label.[2] Namespaces have a hierarchical structure, this structure allows top-down organization of the ontology to arbitrary depth.

Most of the design of GSPAS ontology is stable and reusable. So we reuse the working-parts of GSPAS design and remodel only the problematic cases. Accordingly, in the new design we remodel, among other things, homonyms (one-spelling many-meanings), synonyms (many-spellings one-meaning) and part-of-speech information.

Homonyms: By design, terms in the GSPAS ontology reside in a single namespace and a term is identified by its name (label). As a result, a term like *hammer* which occurs as a word in the standard language, as a tool and an operation will carry all three details in its description. So *hammer* will have a single representation overloaded with three different meanings (resulting in the homonym problem). Such terms will specialize unrelated parents and will be specialized by unrelated children. This causes interleaving of unrelated hierarchies, and leads to spurious (or unintended) inferences. For example, from "*hammer* is a *tool*" and "*hammer* is an *operation*" and "*power-hammer* is a *hammer*," we can infer that "*power-hammer* is an *operation*", a spurious inference.

Homonyms can lead to description errors. For example, if *hammer* as a word is a primitive-concept and as an operation it is a defined-concept then only one of these types can be used. Using one will cause description error for the other.

Homonyms also cause punning. OWL-DL requires the identifiers of objects, classes and properties to be mutually disjoint. Punning is the result of violating this constraint. For example, prepositions like *using* and *with* occur as concepts in the language ontology and as properties in the manufacturing ontology.

The new design adopts "one-term one-meaning" principle, where a term is allowed only one meaning (or sense). Therefore, each sense of a homonym will be defined in a separate namespace. Now, *hammer* will be split into 3 new terms, each with a single meaning and a distinct namespace.

```
lex:hammer   opr:hammer   tool:hammer
```

This solves the homonym problem and enforces "one-term one-meaning" principle. Now, homonyms will have matching labels, but different IRIs, and will not cause spurious inferences.

Synonyms: In the GSPAS ontology, name variations (like synonyms, acronyms, abbreviations, misspellings, regional variations, names given by external source, etc.) are treated as synonyms (we call them GSPAS-synonyms). GSPAS synonyms are stored as data-values in the associated term and so the classifier does not process then. The same approach is used in the new design where GSPAS synonyms are stored in a multi-valued OWL annotation property. Below, we provide an alternate design and give reasons for rejecting it.

GSPAS synonyms for classes and objects can be modeled using predefined properties `owl:equivalentClass` and `owl:sameAs`, respectively. Now, GSPAS synonyms become logical-terms and the classifier will process them. But this has a few side effects. First, we cannot tell apart a term and its synonym because both become first class terms, so the synonym relation has no explicit representation. This is not wrong, but the synonym relationship goes out of sight. Second, the GSPAS synonym relation is neither symmetric nor transitive, but class-equivalence and same-as are both symmetric and transitive, and so will induce spurious synonym relationships. Third, the GSPAS synonyms become new terms in the namespace and may cause homonym problem. This can be solved, but at the expense of introducing spurious homonyms. For these reasons we reject this approach and *treat synonyms as data-values*.

Part-of-speech information: In GSPAS ontology, part-of-speech (POS) information is modeled in two ways: POS tags (like noun, verb, etc.) appear as concepts in the taxonomy (so words in standard language can specialize them), and POS tags are stored as data-values in non-definitional attributes. In the new design, we model POS tags as concepts in the taxonomy. The POS tags stored in the attributes are remodeled into the taxonomy by creating suitable POS concepts and subsumption links.

3.4 Ontology Conversion

Conceptually, ontology conversion takes a GSPAS term and creates one or more new terms from it, and in the process it resolves homonyms and implements the various design choices. For example, a GSPAS term description like

$$\mathrm{C} \equiv \mathrm{A} \sqcap \mathrm{B} \sqcap \forall \mathrm{R.U} \sqcap \exists \mathrm{R} \sqcap \forall \mathrm{S.V} \sqcap \exists \mathrm{S} \qquad (3)$$

may result in new term descriptions like

$$\mathrm{C}_1 \equiv \mathrm{A}_1 \sqcap \exists \mathrm{R}_1.\mathrm{U}_1; \qquad \mathrm{C}_2 \equiv \mathrm{B}_2 \sqcap \exists \mathrm{S}_2.\mathrm{V}_2 \qquad (4)$$

[1]Term refers to any of Concept, Individual, Role or Attribute.

[2]International Resource Identifier (IRI) is the unique identifier.

where, alphabets are term names, subscripts are namespaces, each new term gets one namespace, the left-side of a term description is a name, and the right-side is an expression that refers to terms defined elsewhere in the ontology.

Technically, ontology conversion reduces to the problem of assigning namespace(s) to each name in a term description and then extracting new descriptions from that term description. For example, term C after namespace assignment is shown below, from this, new descriptions $\mathrm{C_1, C_2}$ (in Eqn-4) are extracted after resolving namespace ambiguities.

$$\mathrm{C_{1,2} \equiv A_{1,2} \sqcap B_{2,3} \sqcap \forall R_1.U_{1,3} \sqcap \exists R_1 \sqcap \forall S_2.V_{2,3} \sqcap \exists S_2}$$

In the presence of namespace ambiguity, ontology conversion becomes an inverse problem.[3] There are several solutions to $\mathrm{C_1}$ and $\mathrm{C_2}$, one solution is Eqn-4, another solution is given below, and many other solutions exist.

$$\mathrm{C_1 \equiv A_2 \sqcap \exists R_1.U_3; \qquad C_2 \equiv B_3 \sqcap \exists S_2.V_3}$$

We need a set of rules to select the correct solution from the possible set of solutions. The choice of solution depends on the choice of namespaces, the particular instance of GSPAS ontology, and homonyms in the ontology.

Below, we describe the conversion process with the help of Fig-3, two term-mapping functions and three choice-functions. In Fig-3, *parent* refers to named primitive concept, *role* refers to the role participating in role restrictions, and *filler* refers to value restriction. For example, in Eqn-3, parents of C is $\mathrm{\{A, B\}}$, roles of C is $\mathrm{\{R, S\}}$, and fillers of R in C is $\mathrm{\{U\}}$.

The term-mapping functions are used to track the relationship between GSPAS terms (sources) and new terms (targets). Here, sof maps a target to its source, each target has only one source, and tof (inverse of sof) maps a source to a set of targets. For example,

$$\mathrm{sof(C_1) = C; \quad sof(C_2) = C; \quad tof(C) = \{C_1, C_2\}.}$$

The choice functions are used to disambiguate homonyms and to choose admissible terms. Given a context and set of candidate new-terms, choice functions return a set of new-terms valid in that context. Here, chooseP returns the valid parents for a given concept; chooseR returns the valid roles for a given concept; chooseF returns the valid fillers for a given concept and role pair. For example,

$$\begin{aligned}\mathrm{chooseP(C_1, \{A_1, A_2\})} &= \mathrm{\{A_1\}}\\ \mathrm{chooseR(C_1, \{R_1, S_2\})} &= \mathrm{\{R_1\}}\\ \mathrm{chooseF(C_1, R_1, \{U_1, U_3\})} &= \mathrm{\{U_1\}}\end{aligned}$$

Ontology conversion creates new terms from GSPAS terms, this is done in 4 steps: (A.) create new terms, with empty descriptions, from GSPAS terms; (B.) add parents to the newly created terms, (C.) then roles, (D.) and role fillers (value restrictions). These steps are elaborated below.

[3]The corresponding forward problem is to recover the GSPAS ontology from the new ontology, i.e., drop namespaces and merge descriptions. GSPAS conversion is lossless if we can recover GSPAS ontology from the new ontology.

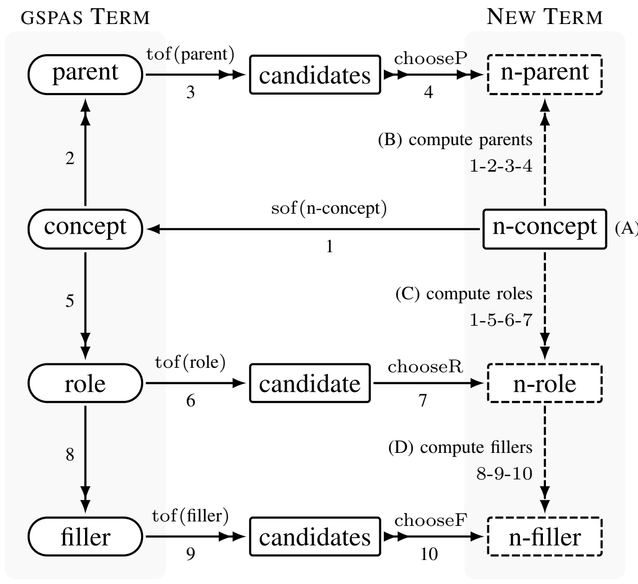

Figure 3: Conversion work flow. Shows GSPAS term and a new term, and the conversion steps A to D. Numbers indicate flow sequence. Double arrow indicates set-valued input/output, otherwise it is scalar input/output. Items to be computed are in dashed-lines.

(A.) Create new-terms (concepts, roles and attributes). First, determine the choice of namespaces for the new ontology, then, for each namespace, identify the terms that belong to it. Homonyms will show up in multiple namespaces. Next, create new-terms out of GSPAS-terms and namespaces, and track the association using sof and tof functions.

```
1  for each $ns$ in $Namespaces$
2      $ns\text{-}terms$ = identify all terms that belong to $ns$
3      for each $term$ in $ns\text{-}terms$
4          $new\text{-}term = \text{Create-Term}(ns, term)$
5          add $term$ to $\text{sof}(new\text{-}term)$
6          add $new\text{-}term$ to $\text{tof}(term)$
```

At this point we have new-terms with empty descriptions, each new-term will map to exactly one GSPAS-term, and each GSPAS-term will map to one or more new-terms.

(B.) Populate parents (follow the path 1-2-3-4 in Fig-3). For each new-concept fetch its GSPAS-parents. For each GSPAS-parent fetch the candidate set. If a parent is a homonym it will return multiple candidates. Now, use the choice function to select valid parents from the candidate set. Add selected parents to the new-concept.

```
7   for each $new\text{-}concept$
8       $concept = \text{sof}(new\text{-}concept)$
9       for each $parent$ of $concept$
10          $candidates = \text{tof}(parent)$
11          $new\text{-}parents = \text{chooseP}(candidates)$
12          add $new\text{-}parents$ to $new\text{-}concept$
```

(C.) Populate roles (follow the path 1-5-6-7 in Fig-3). For each new-concept fetch its GSPAS-roles. For each GSPAS-role fetch the candidate set. Each role has only one meaning in GSPAS ontology, so GSPAS-role and candidate-role will be in 1-to-1 correspondence. So candidate set will be a singleton set. Use the choice function to select valid role from the candidate set. Add selected role to new-concept.

```
13 **for each** $new\text{-}concept$
14     $concept = \mathrm{sof}(new\text{-}concept)$
15     **for each** $role$ **of** $concept$
16         $candidates = \mathrm{tof}(role)$
17         $new\text{-}role = \mathrm{chooseR}(candidates)$
18         **add** $new\text{-}role$ **to** $new\text{-}concept$
19         // code to populate fillers is given below
```

Now, populate attributes in a similar manner.

(D.) Populate role fillers (continue from previous step and follow the path 8-9-10 in Fig-3). For a GSPAS-role fetch its fillers. For each GSPAS-filler fetch the candidate set. Use the choice function to select the valid filler from the candidate set. Add selected fillers to new-concept.

```
19         // code to populate fillers is given below
20         **for each** $filler$ **of** $role$
21             $candidates = \mathrm{tof}(filler)$
22             $new\text{-}fillers = \mathrm{chooseF}(candidates)$
23             **add** $new\text{-}fillers$ **to** $new\text{-}role$
                 **of** $new\text{-}concept$
```

Now, populate attribute fillers in a similar manner.

At the end of step D, all term descriptions are complete and we have a re-engineered namespace-aware ontology which can be serialized in OWL/RDF format.

In the conversion process, the choice function is the workhorse, and the remaining is routine processing. The choice function uses a set of cascading rules to disambiguate terms. For example, given a term and a set of candidate parents, chooseP returns parents from the term's namespace, otherwise returns parents that prefer children from the term's namespace, otherwise returns the candidate set.

For each role, its namespace and the namespaces in which it can be used is predetermined during design phase. Also, its domain and range are predetermined. Given a term and a candidate role, chooseR returns the role if it is usable in the term's namespace otherwise none.

Given a term, a role and a set of candidate fillers, chooseF filters the candidate list progressively until only one candidate is left. First, it selects fillers that are subtype of the role's range, next selects fillers from the term's namespace, and finally selects fillers from the role range's namespace.

The choice functions and their rules were discovered by profiling the GSPAS ontology and by experimentation. The rules are specific to GSPAS ontology, its design-and-organization, the choice of namespaces, homonyms, etc. The rules are tuned to the particular ontology instance that was used for conversion and testing.

3.5 Verification

Verification is done at 3 levels: framework level, ontology level and application level.

At the framework level, (*i.*) we verify the validity of the mapping between KR primitives (Table-2). It is verified by comparing the asserted hierarchies of the new and GSPAS ontology, and then comparing the respective inferred hierarchies. The asserted hierarchy in the new ontology had 4 missing subsumption links out of 12,600+ direct links. These were fixed manually. Next, we manually compared the inferred hierarchies, most of the hierarchy matched, there were about 20 cases where a sub-concept became equivalent to its parent. These types of cases were manually corrected in the new ontology. (*ii.*) Further, we verify the profile of the new ontology. We used Pellet info tool to compute OWL and DL profiles of the new ontology. It turned out to be OWL 2 EL and $\mathcal{EL}^{++}$ (see Table-4) as expected.

At the ontology level, (*i.*) we verify that every GSPAS term has a representation in the new ontology and every new term description is part of some GSPAS term description. This is done by recreating the GSPAS ontology from the new ontology by dropping the namespaces and merging the terms. We manually compared the two versions of GSPAS ontology and found no significant differences. This verification by itself does not establish the validity of the new ontology, but checks whether the conversion is lossless. It is a good first line of defense, and helps in accounting for terms in the new ontology. (*ii.*) Further, we checked for cases of punning using Pellet lint tool, and found one violation which was fixed manually.

The application level verification provides the final validation of the new ontology. It is described in sec-4.

3.6 Performance Testing

In the GSPAS ontology, all terms are modeled as concepts, there are no individuals. But primitive concepts that occur as leafs in the taxonomy and without any role restrictions qualify as individuals. To explore alternate models of GSPAS ontology, qualifying individuals in the part-of-speech hierarchy and object hierarchy are modeled as individuals.

We created five OWL ontologies from GSPAS ontology (see Table-3), each differ in the number of individuals they contain. The first four cases were created for performance testing. The last one was created as a result of performance tuning. We tested three reasoners on the five ontologies using Intel i7-4770 with 16GB RAM running 64bit Ubuntu 12.04; the details are reported in Table-4.

We make the following observations. (*i.*) Of the reasoners, FacT++ has the best overall performance, followed by HermiT and Pellet. (*ii.*) Of the ontologies, LEX-1 has the best overall performance, it has a 1:21 class to individual ratio, and ONT-1 has good overall performance and has no individuals. (*iii.*) The performance, though within acceptable limits, begins to degrade for ONT-2 and ONT-3. HermiT and Pellet are up to 2 orders of magnitude slower than FacT++ on these ontologies.

To understand where the reasoner was spending time we profiled ONT-3 using Pellet[4] and computed the classification time for each concept. From this we prepared a Pareto Chart (term-count vs classification-time), which showed that

[4]In Pellet, concept classification is done by a series of subsumption tests. Pellet reports the execution time for each test, we sum up these times to compute the classification time for a concept.

Table 3: Ontology test cases. LEX-1 is the language ontology with leafs modeled as individuals. ONT-1 is the full ontology with all terms modeled as concepts.
ONT-2 is ONT-1 with lexical-leaves modeled as individuals.
ONT-3 is ONT-2 with object-leaves modeled as individuals.
ONT-4 is ONT-2 with ordinals rolledback to concepts.

Case	Individuals	Individuals	Classes
LEX-1	lex leafs	6,780	317
ONT-1	none	0	12,815
ONT-2	lex leafs	5,679	7,136
ONT-3	lex and obj leafs	6,898	5,917
ONT-4	lex leafs minus ordinals	5,136	7,679

Table 4: Classification time. Reported by Protégé v4.3.0 using FacT++ v1.6.3, Pellet v2.2.0 and HermiT v1.3.8.
The OWL and DL profiles were derived using Pellet Info tool. In DL profile, 'AL' is Attributive Language, 'E' is Existential Restriction, 'H' is Sub-role and 'O' is Ordinals.

	Language Profile		Classification Time (sec)		
Case	OWL	DL	FacT++	HermiT	Pellet
LEX-1	OWL2EL	AL	0.2	0.8	0.7
ONT-1	OWL2EL	ALEH	1.6	12	4
ONT-2	OWL2EL	ALEHO	2.3	74	564
ONT-3	OWL2EL	ALEHO	2.7	352	716
ONT-4	OWL2EL	ALEH	1.7	13	4

Pellet was spending 96% of the execution time in classifying 20% of the terms. We analyzed these terms and found that most of these had `owl:hasValue` restriction in its definition. To verify the impact of `owl:hasValue` on performance, we created ONT-4 from ONT-2 by changing fillers of `owl:hasValue` into concepts and rewriting `owl:hasValue` as existential restriction. Now, ONT-4 out performs ONT-2 and ONT-3, and has a comparable performance to ONT-1.

From this we conclude that creating individuals have less impact on performance, as seen in LEX-1, but using them in `owl:hasValue` restriction degrades performance, as seen in ONT-2, ONT-3. This is true for HermiT and Pellet. In our test, FacT++ consistently outperforms HermiT and Pellet, and for our ontology FacT++ is unaffected by ordinals.

This performance test is solely based on execution time, we did not compare the inferences made by these reasoners, so we do not know if there is any qualitative difference in the inferences produced by these reasoners.

4 Deployment and Maintenance

We (Ford) verified the completeness of the new OWL ontology by developing a tool to compare it to the KL-ONE version. The delivered OWL ontology needed to be validated and verified as the first step toward deployment. This process consisted of several steps. Initially, the OWL ontology was loaded into an Allegrograph server and we wrote various SPARQL queries to determine if the results returned were as expected. In cases where the results were not satisfactory, we then examined the ontology and made modifications if they were required. This manual validation went on for a period of several weeks until we were certain that the OWL ontology was complete and usable.

The next phase of the validation process utilized an automated set of regression tests that were run against the new OWL ontology. This is a set of over 1000 use cases that access the OWL ontology to parse and process the assembly build instructions. In this case, we replaced the KL-ONE ontology with the OWL ontology and ran the entire suite of regression tests and compared the results with the baseline. As with the manual tests we found a number of differences that needed to be analyzed and addressed. These differences fell into the following categories:

- OWL representation was different than KL-ONE, but was part of the re-engineering process. In this case we adjusted the regression tests to reflect how the knowledge was represented in OWL.
- Discrepancies were caused because of formatting, punctuation, special characters and related syntax errors. In these cases, we wrote a routine that would fix these errors as part of the OWL retrieval process, but our intention is to go back and fix these in OWL.
- In some cases, the OWL representation was not what we wanted. In this case we went back to OWL and made the appropriate fixes.

At this point we were confident that the *lexical ontology* was fairly complete and would be usable after the changes made above were completed.

The next step was to build an image using the new OWL ontology and deploy it for user acceptance testing. This testing pointed out some performance issues that were addressed by rewriting the code to make the OWL interface work more efficiently. After these performance issues were fixed the new AI system with the OWL ontology was deployed into the testing environment. No other major issues were discovered during the user acceptance testing phase and the application with the embedded lexical OWL ontology was deployed for use.

We were able to take advantage of the extensibility of the OWL ontology by developing a script that could load a class of parts known as *wire assemblies* directly from an external database. This allows us to add additional knowledge into OWL much more quickly. Another of the main advantages of using OWL was the capability to use standard tools for ontology maintenance. In our case, we are using the Top Braid Composer tool to maintain our OWL ontology which provides much additional capability and allowed us to retire our own tool.

After deployment of the lexical OWL ontology our next goal is to deploy the manufacturing ontology. The OWL ontology is also available for use through Allegrograph and is being utilized by other applications that need the information. Figure-4 shows the structure of our semantic web architecture.

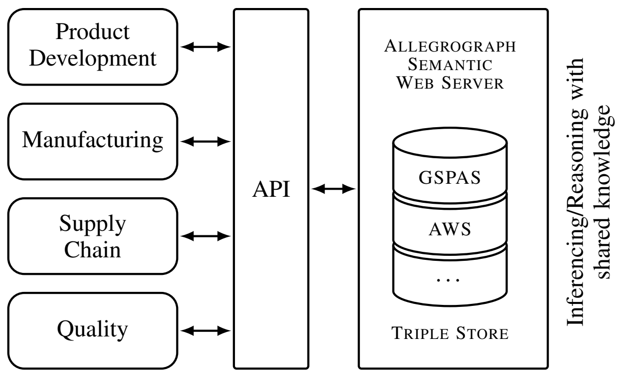

Figure 4: Ford Semantic Web Framework.

5 Conclusions and Future Steps

In this paper we described a project where Ford collaborated with the Indian Institute of Technology Madras to re-engineer and convert an existing ontology into a semantic web OWL/RDF architecture. After a thorough validation procedure the lexical ontology has been deployed as part of the GSPAS system at Ford. The manufacturing ontology will also undergo the same rigorous validation before deployment into production.

There were a number of compelling reasons that motivated the re-engineering of the ontology from KL-ONE to OWL. The most important ones were based around maintainability and extensibility. The original software was written before any software tools for ontology maintenance were available. The KL-ONE ontology could only be maintained using a specialized tool. This tool has had to be re-written several times as operating systems and hardware were being upgraded and it was becoming a bottleneck for future ontology development. The KL-ONE ontology was not usable outside the application without designing custom code to extract specific knowledge. In the meantime business requirements for the ontology were rapidly increasing and the existing architecture could not support them. The conversion of the ontology to OWL was a critical requirement for the future usage of the AI application. Our experience was somewhat unique in that we have been using KL-ONE since the 1990s and much of the work in semantic web had taken place after we had a deployed application.

The conversion from KL-ONE to OWL required a significant amount of work, but the advantages from moving into a semantic web architecture made this a worthwhile investment. It enables us to take advantage of existing tools and processes and to make our ontology reusable and extensible using existing standards. Queries can easily be developed using SPARQL that allow other applications to access our ontology. The semantic web infrastructure also gives us the capability to link to other ontologies and take advantage of the linked open data world. Therefore, the ROI on this project is based on the following: increased functionality with OWL vs. KL-ONE and reduced expenses in terms of maintenance costs.

Our future work will include the deployment of our manufacturing ontology into production as well as the use of semantic web tools for ontology development and maintenance. This project has helped us create a center of excellence around semantic technology to support other semantic web work at Ford.

References

Baader, F.; Calvanese, D.; McGuinness, D. L.; Nardi, D.; and Patel-Schneider, P. F., eds. 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Baader, F.; Brandt, S.; and Lutz, C. 2008. Pushing the EL Envelope Further. In *Proc. of the OWLED 2008 DC Workshop on OWL: Experiences and Directions*.

Bechhofer, S.; van Harmelen, F.; Hendler, J.; Horrocks, I.; McGuinness, D. L.; Patel-Schneider, P. F.; and Stein, L. A. 2004. OWL Web Ontology Language Reference. W3C Recommendation. http://www.w3.org/TR/2004/REC-owl-ref-20040210/.

Brachman, R. J., and Schmolze, J. G. 1985. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science* 9(2):171–216.

Corcho, Ó., and Gómez-Pérez, A. 2005. A Layered Model for Building Ontology Translation Systems. *Int. J. Semantic Web Inf. Syst.* 1(2):22–48.

Euzenat, J. 2001. Towards a Principled Approach to Semantic Interoperability. In *Proc. of IJCAI Workshop on Ontologies and Information Sharing*, IJCAI'01, 19–25.

Motik, B.; Grau, B. C.; Horrocks, I.; Fokoue, A.; and Wu, Z. 2012. OWL 2 Web Ontology Language Profiles (2nd Ed.). W3C Recommendation. http://www.w3.org/TR/2012/REC-owl2-profiles-20121211/.

Rychtyckyj, N. 1994. Classification in DLMS Utilizing a KL-ONE Representation Language. In *6th Int. Conf. on Tools with AI*, ICTAI'94, 339–345.

Rychtyckyj, N. 1999. DLMS: Ten Years of AI for Vehicle Assembly Process Planning. In *Proc. of AAAI'99/IAAI'99*, 821–828.

Rychtyckyj, N. 2006. Standard Language at Ford Motor Company: A Case Study in Controlled Language Development and Deployment. In *Proc. of CLAW'06*.

Schmolze, J. G., and Lipkis, T. A. 1983. Classification in the KL-ONE Knowledge Representation System. In *Proc. of IJCAI'83*, 330–332.