Ontology Reengineering: A Case Study from the Automotive Industry

Nestor Rychtyckyj, Venkatesh Raman, Baskaran Sankaranarayanan, P. Sreenivasa Kumar, Deepak Khemani

■ *For more than 25 years Ford Motor Company has been utilizing an AI-based system to manage process planning for vehicle assembly at its 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 reengineer and convert the KL-ONE ontology into a semantic web OWL/RDF format. In this article, we will discuss the process by which we reengineered the existing GSPAS KL-ONE ontology and deployed semantic web technology in our application.*

The Direct Labor Management System (DLMS) (Rychtyckyj 1999) was initially developed and deployed in Ford Motor Company'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 that the Global Study Process Allocation System (GSPAS) KL-ONE ontology would be much more usable and of better value to Ford if it could be rewritten into OWL/RDF. An ontology based on modern semantic web standards 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 Ford found it by partnering with the Indian Institute of Technology Madras (IITM) in Chennai, India. Ford elected to partner with IITM because the university has an excellent reputation with a strong background in artificial intelligence (Khemani 2013), and moreover, Ford wanted to develop a strong relationship with the university.

The results of this project were very successful. The IITM team delivered a reengineered 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

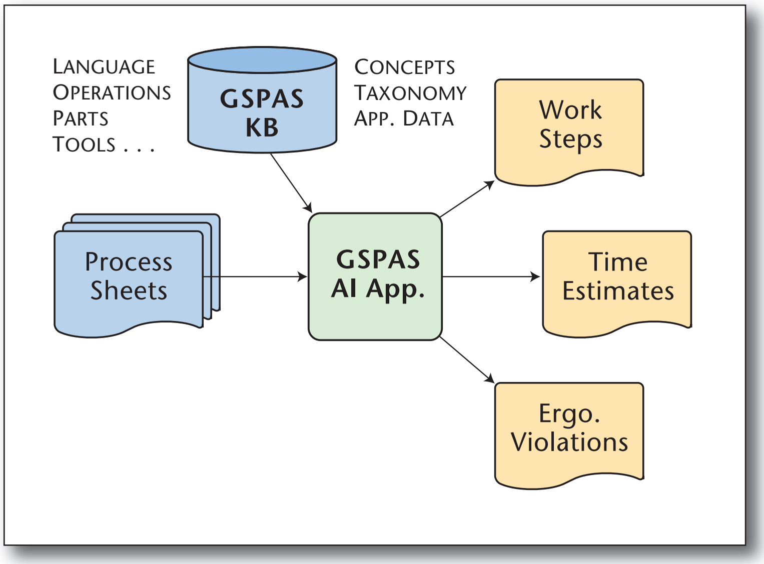
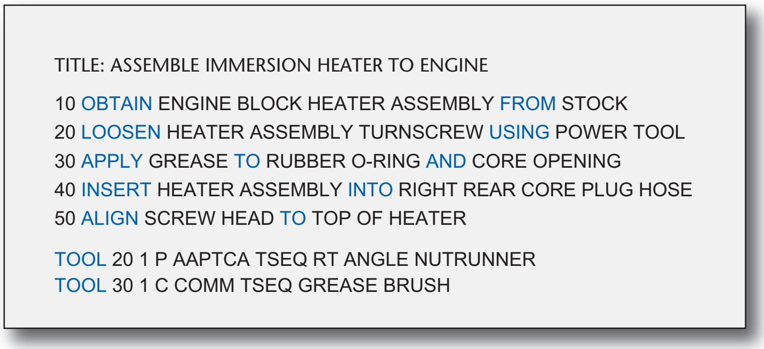


Figure 1. GSPAS AI Application.



TITLE: ASSEMBLE IMMERSION HEATER TO ENGINE

10 OBTAIN ENGINE BLOCK HEATER ASSEMBLY FROM STOCK
20 LOOSEN HEATER ASSEMBLY TURNSCREW USING POWER TOOL
30 APPLY GREASE TO RUBBER O-RING AND CORE OPENING
40 INSERT HEATER ASSEMBLY INTO RIGHT REAR CORE PLUG HOSE
50 ALIGN SCREW HEAD TO TOP OF HEATER

TOOL 20 1 P AAPTCA TSEQ RT ANGLE NUTRUNNER
TOOL 30 1 C COMM TSEQ GREASE BRUSH

Figure 2. Process Sheet.

requirements and has deployed the lexical ontology into the GSPAS application. In the rest of the article we will describe the structure and usage of the existing KL-ONE ontology, and then describe the conversion approach and the conversion process.

In this article, we refer to the GSPAS KL-ONE ontology as GSPAS KB or as GSPAS ontology or as KL-ONE ontology, and refer to the reengineered GSPAS OWL ontology as new ontology or as OWL ontology.

GSPAS and the KL-ONE Ontology

Ford's DLMS was developed to standardize vehicle assembly, improve efficiency, and reduce cost throughout the entire manufacturing process planning system. DLMS was then integrated into Ford's Global Study Process Allocation System, which is currently used across all of Ford's global vehicle assembly and powertrain plants.

Artificial intelligence in GSPAS is used for several different purposes: (1) Validate the correctness of process sheets that describe assembly operations. (2) Develop a list of operator work instructions and associated MODAPTS (modular arrangement of predetermined time standards) codes (Sullivan, Carey, and Farrell 2001) for each assembly operation in the process sheet. (3) Check the process sheet for ergonomic concerns. (4) Translate the process sheets into the language used at a particular assembly plant.

Figure 1 shows the architecture of the GSPAS AI application. Figure 2 shows a sample process sheet with five build steps and two tool specifications; at such granularity, thousands of process sheets are used to document the build steps for a whole vehicle. The core of the GSPAS AI application is an ontology that contains relevant knowledge about Ford's manufacturing processes including the labor requirements for the assembly operations, part and tooling information, workplace ergonomic concerns, linguistic representation of Standard Language (Rychtyckyj 2006) and other concepts. Figure 3 shows how this ontology is used to generate operator work instructions and MODAPTS codes. Each build step in the process sheet is parsed and transformed into a KL-ONE description, which is then classified to find the matching concepts in GSPAS KB. The matching concepts provide meaning to a build step and also supply the necessary work steps and MODAPTS codes.

The ontology was developed inhouse using the KL-ONE knowledge representation language and includes a graphic user interface for ontology editing as well as a classifier. The GSPAS ontology has been updated frequently to keep in sync with all of the changes that have occurred to Ford and the automobile industry in general. The automotive business has evolved dramatically and Ford itself bought and then sold off companies such as Jaguar, Land Rover, and Volvo. The manufacturing process, technology, and tooling have all changed dramatically over the last few years, and all of these changes needed to be reflected in the GSPAS ontology. Technology and parts for new products like electric and hybrid-electric vehicles, in-vehicle infotainment, and aluminum bodies all became part of the Ford manufacturing process and consequently needed to be added into Standard Language and the GSPAS ontology. On the other hand, different concepts in the ontology became obsolete and were no longer needed. Throughout the intervening years and all of the changes, the KL-ONE ontology model and classifier proved to be robust enough to support GSPAS and Ford's manufacturing plants.

Ford adapted the KL-ONE knowledge representation system during its initial development of DLMS. There were no KL-ONE tools or editors available so Ford built both a KL-ONE editor as well as the code for classification and reasoning (Rychtyckyj 1994). The knowledge base update module, an in-house developed graphic user interface, 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 1970s. 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 (Lipkis 1981).

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 the papers by Lipkis (1981) and Schmolze and Lipkis (1983).

The existing KL-ONE ontology proved to be very robust and flexible as Ford made hundreds of changes to it on an annual basis. Both the business and the technology changed dramatically, but Ford managed to keep the system fully functional as its scope increased. 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 graphic user interface was rewritten several times as the application 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. We had previously explored using an automated learning approach to reengineer our KL-ONE ontology, but the results showed that the new ontology was not as intuitive and understandable to users and developers.

Reengineering GSPAS into OWL

The goal is to reengineer the GSPAS ontology into an OWL ontology that will preserve the existing relations and links. This reengineering involves ontology translation, which maps GSPAS ontology to an OWL ontology, and ontology modeling, which identifies a design for the OWL ontology while resolving some of the issues in the existing design.

GSPAS to OWL translation follows a four-layered translation model (Corcho and Gómez-Pérez 2005, Euzenat 2001) consisting of lexical, syntactic, semantic, and pragmatic levels. This model covers all aspects of ontology translation, including semantics preservation, which is a key requirement that is not always easy to satisfy.

In this model, the lexical and syntactic levels deal with the translation of characters, words, values, strings, and sentences between knowledge representation (KR) languages. The semantic level deals with KR framework translation and semantics preservation. The pragmatic level deals with the choice of modeling and encoding that relates to scalability, maintainability, and ontology usage. For example, an entity can be modeled as a class (*red* as a class of *color*) or as an individual *(red* is a *color*). And a binary relation can be modeled as a subclass relation *(obtain* as a class of *verb)* or as a role restriction (*obtain* has part-of-speech some *verb*) or as a role assertion *(obtain* has part-of-speech *verb)*. The choice is between storing information in the taxonomy versus storing it in role links. Further, one can find attach application data to classes, individuals, or roles; and interpret it before or after building the taxonomy.

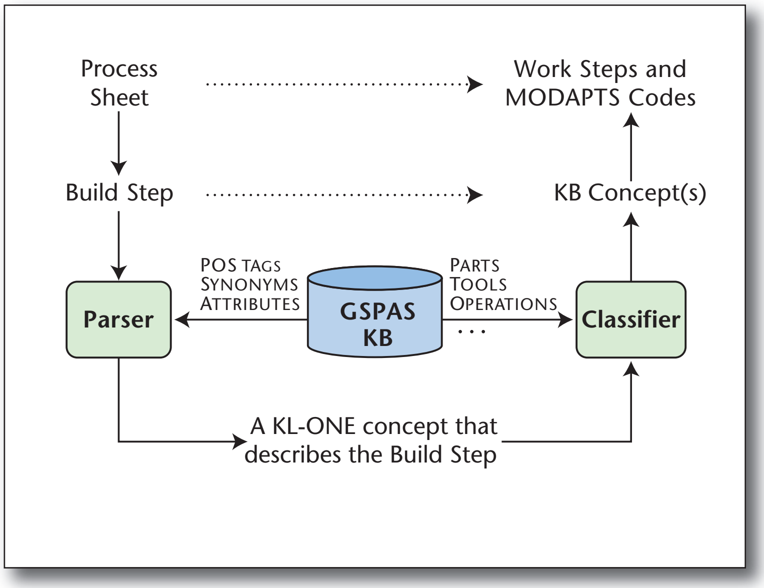


Figure 3. Ontology Use Cases.

As shown in figure 4, our approach to reengineering (modeling and translation) starts with the study phase and works through three levels of abstraction, namely, framework, design, and ontology levels, and finally, ends with the validation phase. We follow a spiral development model, which makes several iterations through the various phases. The framework-mapping and design phases incorporate the semantic and pragmatic aspects from the four-layered model. The ontology conversion tool implements, among other things, the lexical and syntactic translations. The remainder of this section describes the various phases in figure 4.

Study Phase

In this phase, the goal is to study the GSPAS and OWL (Bechhofer et al. 2004) frameworks and the GSPAS ontology and further understand the reengineering problem and identify areas that need improvement.

To accomplish this goal, the IITM team studied the GSPAS, KL-ONE, Description Logics (DL), and OWL

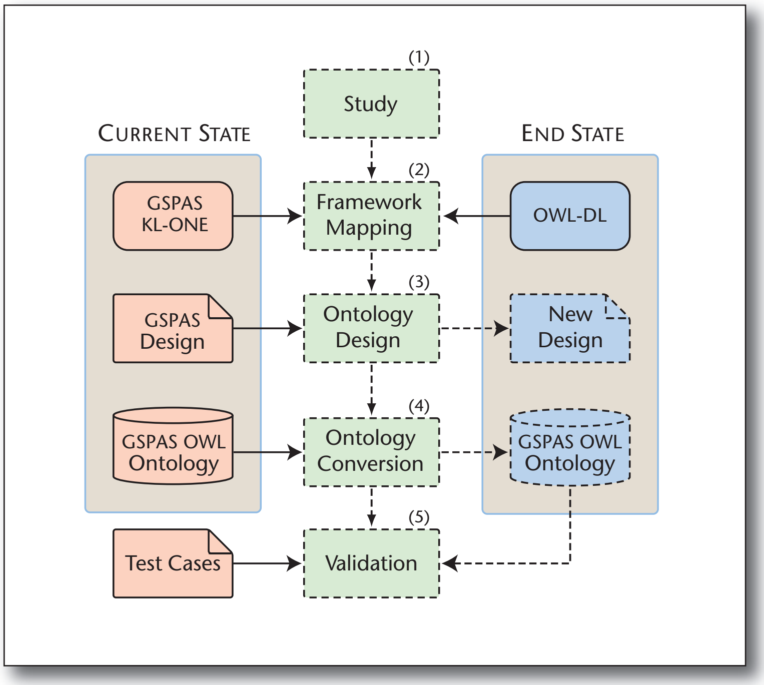


Figure 4. Ontology Reengineering.

The figure shows the current and end states of the ontology, the inputs to reengineering (solid line), and the various phases and deliverables (dashed line).

frameworks, and with the help of the Ford team analyzed the GSPAS ontology. Then the IITM team developed a document that presented (1) their understanding of the KR frameworks, (2) a potential mapping between GSPAS and OWL, (3) their understanding of the design, organization, and use cases of GSPAS ontology, and (4) a high-level approach to GSPAS ontology reengineering.

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.

Framework Mapping

An ontology describes terms in a domain and captures their association with other terms in that domain. A structure-preserving transformation maps each term and its associations and subsumptions from a source ontology to a term with corresponding associations and subsumptions in a target ontology, and thereby preserves the semantics of these terms.

GSPAS implements a subset of KL-ONE that satisfies Ford's AI needs. We worked with this subset instead of the full KL-ONE. Accordingly, the goal of framework mapping is to create a semantics-preserving mapping between GSPAS (a subset of KL-ONE) and OWL frameworks. This mapping is created for each of vocabulary, representation, and reasoning components of these frameworks.

Vocabulary

GSPAS, KL-ONE, DL, and OWL frameworks, though related, were developed by different groups across space and time. This naturally led to the use of different names to refer to a given idea. Table 1 documents the various vocabularies and their correspondences. It also shows the GSPAS features (un) supported in other frameworks.

Representation

To encode knowledge, the GSPAS ontology uses two kinds of concepts (primitive and defined) and two concept-forming operators (value-restriction and conjunctions), further, it uses classifiable attributes (roles) to define value restrictions, and two kinds of nonclassifiable attributes (nondefinitional roles) to store application data, where one is inherited by subclasses and the other is noninheritable. In KL-ONE and so in GSPAS, a primitive-concept provides necessary conditions for membership, whereas a defined-concept provides both necessary and sufficient conditions for membership. And a value restriction restricts all fillers of a role to a given type or concept, and allows us to describe concepts based on these restrictions, like *things whose tires are slick*. Consider the statement, *Formula One car has slick tires*. If this is taken to provide a necessary condition about F1 cars (a primitive concept) then it states that *tires of F1 car are slick tires*. Instead, if it is taken to provide both necessary and sufficient condition about F1 cars (a defined concept) then it states that *tires of F1 car are slick tires, and things whose tires are slick are F1 cars.* In short, the GSPAS KR language permits the following:

$A \sqsubseteq C$ (primitive concepts); $A \equiv C$ (defined concepts)

where A is any concept name, and C is a concept forming expression which can be a concept-name or a value-restriction or a conjunction, as shown below. Here A_1, A_2 are concept names, R is role name, and C_1, C_2 are concept forming expressions.

$C \rightarrow A_1 \quad C \rightarrow (\forall R.A_2 \sqcap \exists R) \quad C \rightarrow C_1 \sqcap C_2$

Using this notation, we can describe F1-Car as a primitive concept: $\text{F1-Car} \sqsubseteq \text{Car} \sqcap (\forall \text{tire.Slick-Tire} \sqcap \exists \text{tire})$, which states that F1-Car is a car and all its tires are slick tires and has some tires. See how the textual description resembles the expression.

For a lossless translation, we have to map the GSPAS KR language to OWL constructs that will preserve the meaning of domain terms and their subsumptions. One such mapping (table 2) is discussed next.

First, the primitive concepts are mapped to partial concepts in OWL and are encoded as subclass axioms. And defined concepts are mapped to complete concepts in OWL and are encoded as class-equivalence axioms. Further, concept names and concept conjunctions are mapped, respectively, to class names and class intersections in OWL. These four mappings are exact.

Next, GSPAS roles are mapped to object properties.

	GSPAS	KL-ONE	DL	OWL
1	THING	THING	Top concept ‘T’	owl:Thing
2	Concept	Concept	Concept	Class
3	Primitive Concept	Primitive Concept	Atomic Inclusion	Partial Concept
4	Generic Concept	Defined Concept	Definition	Complete Concept
5	Individual	Individual Concept	Individual	Object
6	Role Restriction	Role Restriction	Role Restriction	Property Restriction
7	Value Restriction	Value Restriction	Value Restriction	Value Restricti
8	Number Restriction	Number Restriction	Number Restriction	Cardinality restriction
9	Classifiable Attribute	Role	Role	Object Property
10	Nondefinitional Attribute	Nondefinitional Role	n/a	Annotation Property
11	Nondefinitional Inheritable Attribute	n/a	n/a	n/a
12	Classifier	Classifier	Reasoner	Reasoner

Table 1. Vocabulary Mapping.

	GSPAS	KL-ONE	DL*	OWL
1	Primitive Concept	Primitive Concept	A ! C	rdfs:subClassOf
2	Generic Concept	Defined Concept	A " C	owl:equivalentClass
3	Value Restriction (#R.A $ ∃R)	Value Restriction	∃R.A	owl:someValuesFrom
4	Conjunction	Conjunction	$C_1 \$ C_2$	owl:intersectionOf
5	Classifiable Attribute	Role	Role	owl:ObjectProperty
6	Nondefinitional Attribute	Non-definitional Role	n/a	owl:AnnotationProperty
7	Nondefinitional Inheritable Attribute	n/a	n/a	n/a

Table 2. GSPAS KR Primitives (Modeling Elements) and Their OWL Translation.

*In DL expression, A is concept name; C, C_1, C_2 are concept forming expressions; R is role name.

And nondefinitional attributes are mapped to annotation properties. The inheritable nondefinitional attributes (not supported in KL-ONE and OWL) are modeled as annotation properties and the attribute inheritance is handled in the application. These three mappings are lossless, and the logical (roles and concepts) versus nonlogical (annotation properties and application data) separation remains intact.

Finally, GSPAS value restriction (∀R.A ⊓ ∃R), which restricts all fillers of R to concept A, is remodeled as existential restriction ∃R.A in the OWL ontology, which restricts R to have some fillers from concept A and, optionally, other fillers from other concepts. It is our observation that, in the GSPAS ontology, concepts that are best modeled using existential restriction are modeled using value restriction.

Observe that (∀R.A ⊓ ∃R) is a subclass of ∃R.A, and so, the existential restriction admits more models than the corresponding value restriction. This is a widening or relaxing transformation that preserves subsumption structure (subclass or *is-a* relation). We will justify this for both assertion and inference links.

Consider two value restrictions in figure 5, and their translation given by *is-a*$_1$ and *is-a*$_2$. If *is-a*$_3$ is asserted in the GSPAS ontology then *is-a*$_4$ will be asserted during ontology conversion. By *is-a*$_3$ and *is-a*$_1$ all individuals of ($\forall R.A_2 \sqcap \exists R$) will belong to $\exists R.A_1$, making *is-a*$_5$ true. By similar argument, *is-a*$_2$ and *is-a*$_4$ also make *is-a*$_5$ true. As a result, the asserted *is-a*$_4$ agrees with the assertion *is-a*$_3$ (figure 5).

The sufficient conditions for inferring *is-a* link between a concept *Sub* and a concept *Super* is stated in Lipkis (1981). Two of the relevant conditions are (1) Each role of *Super* is modified by a role of *Sub.* (2) Each value description of each role of *Super* subsumes a value description of the corresponding role of *Sub.* Accordingly, if A_1 subsumes A_2, then *is-a*$_3$ will be inferred, and correspondingly, *is-a*$_4$ will be inferred in OWL. Therefore *is-a*$_3$ (be it an assertion or an inference) will have a corresponding *is-a* in the OWL ontology, and thus subsumption links will be preserved.

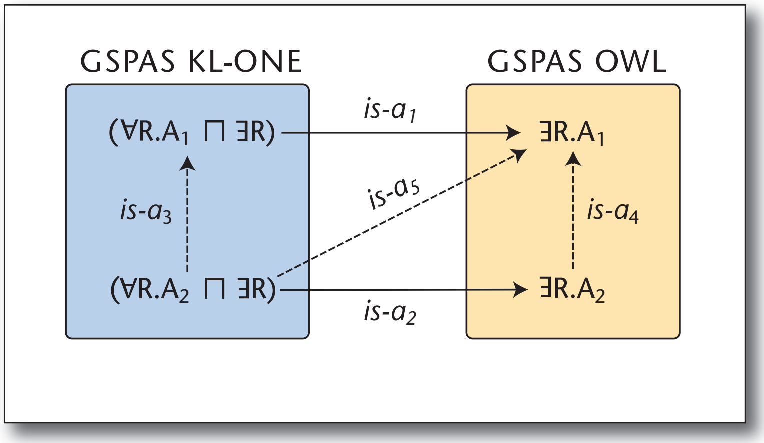


Figure 5. Translation of Value Restriction.

Next, we offer three reasons for choosing existential restrictions over value restrictions: (1) Between KL-ONE and OWL there is a paradigm shift. OWL ontologies use variants of existential restriction to model common use cases found in real-world ontologies. (2) It reduces the computational complexity of the resulting ontology. (3) It tends to reduce the number of base terms in the ontology. For example, we can model car owners in two ways. First, using value restriction a car owner is someone whose owns-car role is filled only by cars ($\forall$owns-car.Car $\sqcap$ $\exists$owns-car), and second, using existential restriction it is someone whose owns role is filled with a car ($\exists$owns.Car). For ship owners, we get ($\forall$owns-ship.Ship $\sqcap$ $\exists$owns-ship) and ($\exists$owns.Ship), respectively. The first model uses different roles to describe different owner concepts, whereas the second model uses just one role (owns) for that purpose.

In this section, we have presented a structure-preserving mapping between GSPAS and OWL primitives. Based on this mapping, the KR language of the new OWL ontology is:

$$A \sqsubseteq C \qquad A \equiv C$$
$$C \rightarrow A_1 \qquad C \rightarrow \exists R.A_2 \qquad C \rightarrow C_1 \sqcap C_2$$

Reasoning

The GSPAS classifier, a derivative of the KL-ONE classifier (Lipkis 1981), uses structure matching to compute subsumptions, 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); in fact, structure matching is complete only for a small subset of OWL-DL (Khemani 2013; Brachman and Levesque 2004); that is, for a given knowledge base, logical subsumption will find all inferences that structural subsumption can find and possibly more. Moreover, the mapping from GSPAS KR language to the new KR language preserves subsumption links. Therefore, we conclude that each subsumption link in the GSPAS ontology will have a corresponding link in the OWL ontology. Further, a GSPAS concept will be a subclass of the corresponding OWL class.

Furthermore, the new OWL ontology allows domain restriction, range restriction, and subroles:

$$\text{domain}(R) \sqsubseteq A1 \qquad \text{range}(R) \sqsubseteq A2 \qquad R \sqsubseteq S$$

where R, S are role names, and A_1, A_2 are concept names. Now, the profile of the new OWL ontology is a subset of $\mathcal{EL}^{++}$ profile (Baader, Brandt, and Lutz 2008; Motik et al. 2012), which in turn is a subset of OWL-DL profile. $\mathcal{EL}^{++}$ runs in polynomial time for common reasoning tasks. We experimented with other DL profiles and selected $\mathcal{EL}^{++}$ because it provides a good balance between expressiveness and performance for the GSPAS ontology.

Ontology Design and Organization

The GSPAS ontology supports two use cases (figure 3): to parse build-steps written in Standard Language, and to interpret parsed build steps. As a result, there are two sets of terms in the ontology — one that describes words in the Standard Language and the other that describes build steps, parts, tools, and so on. All terms reside in a common namespace, and a term is identified by its name (label).

Ontology Organization

Each term (concept, individual, role, or attribute) in the new ontology is assigned a namespace, a label, and a unique identifier. The unique identifier[1] is generated from the namespace[2] and label. Namespaces have a hierarchical structure, which allows top-down organization of the ontology to arbitrary depth.

The new ontology is divided into subject areas, namely, language and manufacturing. Each subject area is divided into smaller areas (like verbs, parts, tools, and others), and so on to arbitrary depth. One or more namespaces are used to organize a subject area. Figure 6 shows the differences between the GSPAS ontology and the new ontology.

Ontology Design

The various concept types, role types, and modeling choices (like entity as concept versus individual, binary relation as subclass-relation versus role versus annotation property, and others) and the various hierarchies (lexical hierarchy, operations, parts, tools, and others) in the GSPAS ontology are mostly stable and are retained as such in the new ontology. We reused the working parts of the design and remodeled only the problematic cases. Here, we describe how the new ontology models three interesting problems: homonyms (one-spelling, many-meanings), synonyms (many-spellings, one-meaning), and part-of-speech information.

Homonyms

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 that occurs as a lexical term, a tool, and an operation will have a single representation overloaded with three meanings. Such terms will cause interleaving of unrelated hierarchies and produce spurious inferences. For example, given that HAMMER is a TOOL and HAMMER is also an OPERATION, if POWER-HAMMER is a HAMMER, then POWER-HAMMER becomes a TOOL as well as an OPERATION. The latter inference is spurious.

Homonyms can cause incorrect descriptions; for example, a concept can be either primitive or defined; if HAMMER as a tool is a primitive concept, and as an operation it is a defined concept, then choosing either type will lead to incorrect description.

Homonyms can 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 ontology adopts the one term, one meaning (OOM) principle, where a new term will carry only one meaning. Therefore, each sense of a homonym will be independently represented. Thus, HAMMER will split into three terms, each with a single meaning and a distinct namespace.

lex:HAMMER   opr:HAMMER   tool:HAMMER

This solves the homonym problem. 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 sources, and others) are treated as synonyms (call them GSPAS synonyms). GSPAS synonyms are stored as data values in the associated term and so the classifier does not process them. The same approach is used in the new ontology where GSPAS synonyms are stored in OWL annotation property. Next, we present an alternative approach and give reasons for rejecting it.

GSPAS synonyms of classes and objects can be modeled using the predefined properties owl:equivalentClass and owl:sameAs, respectively. Now, GSPAS synonyms become logical terms and the classifier will process them. This has some side effects. First, we cannot tell apart a term and its synonym because both are first-class terms; this is not wrong, but the synonym relation goes out of sight. Second, the synonym relation is neither symmetric nor transitive, but owl:equivalentClass and owl:sameAs are both symmetric and transitive and so will induce spurious synonym relationships. Third, the GSPAS synonyms become new terms and may cause homonym problems. This can be solved at the expense of introducing spurious homonyms (matching labels but different IRIs). 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, and others) appear as concepts in the taxonomy (so words in Standard Language can specialize them), and POS tags are stored as data values in a nondefinitional attribute. In the new ontology, we model POS tags as concepts in the taxonomy. The tags stored in the attributes are remodeled into the taxonomy by creating suitable POS concepts and subsumption links.

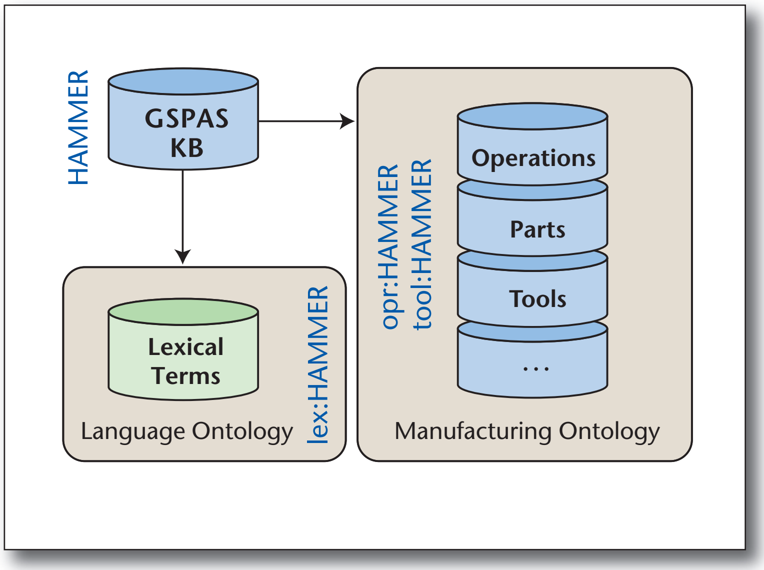


Figure 6. Reengineered Ontology.

Ontology Conversion

HAMMER has three senses: As an OPERATION it operates on an OBJECT restricted to HAMMERABLE type, and as a TOOL its SIZE is restricted to HAMMER-SIZE. In the interest of space we will ignore the lexical sense of HAMMER.

$$\begin{aligned}\text{HAMMER} \sqsubseteq\ & \text{OPERATION} \sqcap \text{TOOL} \sqcap \\ & (\forall\text{OBJECT.HAMMERABLE} \sqcap \exists\text{OBJECT}) \sqcap \\ & (\forall\text{SIZE.HAMMER-SIZE} \sqcap \exists\text{SIZE})\end{aligned}$$

Conceptually, ontology conversion takes a GSPAS term description and creates one or more new descriptions after resolving homonyms and implementing the various design choices. For the case of hammer, our goal is to split its description into two new descriptions:

$$\begin{aligned}\text{HAMMER}_{opr} \sqsubseteq\ & \text{OPERATION}_{opr} \sqcap \\ & \exists\text{OBJECT}_{opr}.\text{HAMMERABLE}_{obj} \\ \text{HAMMER}_{tool} \sqsubseteq\ & \text{TOOL}_{tool} \sqcap \\ & \exists\text{SIZE}_{tool}.\text{HAMMER-SIZE}_{tool}\end{aligned}$$

where each new term is assigned a single namespace that is denoted by its subscript, the left side of a description is a name, and the right side is an expression that refers to other term descriptions in the ontology.

Technically, the GSPAS ontology conversion reduces to the problem of assigning one or more

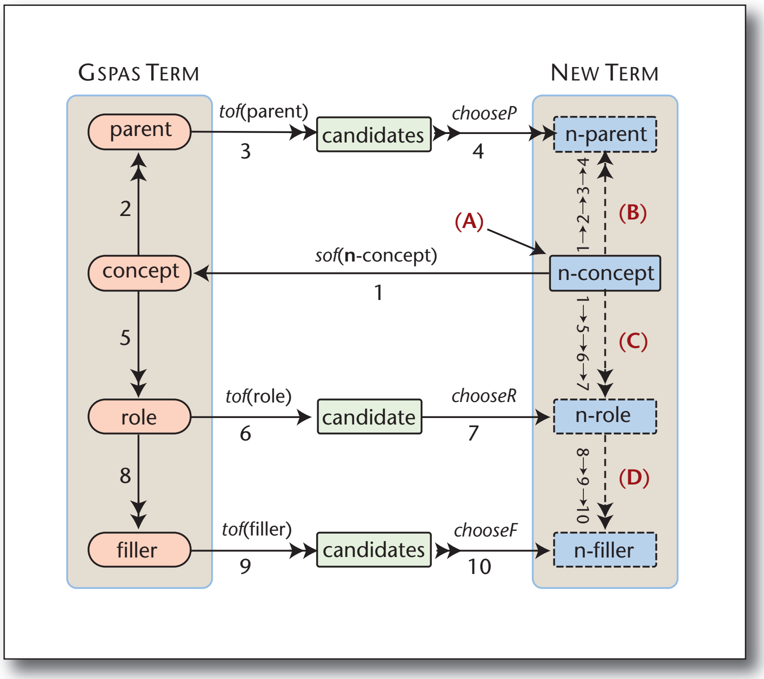


Figure 7. Conversion Work Flow.

Numbers indicate flow sequence. Nodes are sets; edges are functions. A double arrowhead indicates a set valued input/output. The items to be computed are in dashed lines.

namespaces to each name in a description and then extracting new descriptions. The description of HAMMER after namespace assignment is shown below; from this, HAMMER_{opr} and HAMMER_{tool} will be extracted after resolving namespace ambiguity.

$$\text{HAMMER}_{opr,tool} \sqsubseteq \text{OPERATION}_{opr,lex} \sqcap \text{TOOL}_{tool,lex} \sqcap (\forall \text{OBJECT}_{opr}.\text{HAMMERABLE}_{obj} \sqcap \exists \text{OBJECT}_{opr}) \sqcap (\forall \text{SIZE}_{tool}.\text{HAMMER-SIZE}_{tool} \sqcap \exists \text{SIZE}_{tool})$$

In the presence of namespace ambiguity, ontology conversion becomes an inverse problem and so it has several solutions. The corresponding forward problem is to recover the GSPAS ontology from the new ontology, that is, drop the namespaces and merge the descriptions. The conversion is lossless if the GSPAS ontology can be fully recovered from the new ontology. To choose the correct description of HAMMER_{opr} and HAMMER_{tool}, we need a set of rules, also called choice functions, that will depend on the list of homonyms, list of namespaces, and the organization of GSPAS ontology.

In what follows, we describe the conversion process (figure 7) with the help of term-mapping functions and choice functions. In figure 7, *parent* denotes a named parent concept, *role* denotes a role name, and *filler* denotes a value restriction (which is a concept name). For the concept HAMMER, parents are {OPERATION, TOOL}, roles are {OBJECT, SIZE}, and filler of OBJECT is {HAMMERABLE}. The term-mapping functions track the link between GSPAS terms and new terms: *tof* (target-of) maps a GSPAS term to a set of new terms, and *sof* (source-of) maps a new term to a GSPAS term.

$$tof(\text{HAMMER}) = \{\text{HAMMER}_{opr}, \text{HAMMER}_{tool}\}$$
$$sof(\text{HAMMER}_{opr}) = \text{HAMMER}$$

The choice functions are used to resolve homonyms and select admissible terms. Given a new concept, *chooseP* takes candidate parents and returns the admissible parents; similarly, *chooseR* takes candidate roles and returns the admissible roles, and further, *chooseF* returns the admissible fillers for a new concept-role pair. Given HAMMER_{opr}, *chooseP* takes $\{\text{OPERATION}_{opr}, \text{OPERATION}_{lex}\}$ and returns $\{\text{OPERATION}_{opr}\}$, similarly, *chooseR* takes $\{\text{OBJECT}_{opr}, \text{SIZE}_{tool}\}$ and returns $\{\text{OBJECT}_{opr}\}$. Given HAMMER_{opr} and OBJECT_{opr}, *chooseF* takes $\{\text{HAMMERABLE}_{obj}\}$ and returns $\{\text{HAMMERABLE}_{obj}\}$.

Ontology conversion creates new descriptions by making several passes over the GSPAS ontology: (Step A) it first creates new terms, with empty descriptions, (Step B) then adds parents to the newly created terms, (Step C) then adds roles, (Step D) and finally role fillers (value restrictions). (See Listing 1.)

Step A.
To create a new term we need a namespace and a label. First, we identify the namespaces of the new ontology then we assign GSPAS terms to namespaces. Homonyms will show up in multiple namespaces. Now, we create one new term for each GSPAS term and its namespace combination, and we track this association using *sof* and *tof* functions (listing one). At this point we will have new terms with empty descriptions; each new term will link to one GSPAS term, and each GSPAS term will link to one or more new terms. Use *sof* and *tof* to complete the rest of the conversion process.

Step B.
To populate new parents, follow the edges 1, 2, 3, 4 in figure 7. For each new concept and its GSPAS parent, fetch the candidate parents, if a GSPAS parent is a homonym, it will return multiple candidates. Now, select the admissible parents and add them to the new concept (listing one).

Step C.
To populate new roles, follow the edges 1, 5, 6, 7 in figure 7. For each new concept and its GSPAS role, fetch the candidate roles, which will be a singleton set because GSPAS roles have only one meaning. Now, select the admissible role, and add it to the new concept (listing one). Now, populate attributes in a similar manner.

Step D.
To populate role fillers, continue from the previous step and follow the edges 8, 9, 10 in figure 7. For a GSPAS role and its GSPAS filler, fetch the candidate fillers. Now, select the admissible fillers, and add it to the new role in new concept (listing one). Add selected fillers to new concept. Now populate attribute fillers in a similar manner.

At the end of step D, all term descriptions are complete and we have a reengineered namespace-aware ontology that is ready for lexical and syntactic translation.

In the conversion process, namespace assignment and the choice functions are two important decision points, and the remaining is routine processing. The choice functions use a set of cascading rules to disambiguate terms. Given a concept and a set of candidate parents, *chooseP* returns the parents from the concept's namespace; otherwise it returns the parents that have a preference to children from the concept's namespace, and otherwise it returns the candidate set.

For each role, its namespace and the namespaces in which it can be used are determined during the design phase. Also, its domain and range are predetermined. Given a concept and a candidate role, *chooseR* returns the role if it is admissible in that concept's namespace.

Given a concept, 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 subtypes of the role's range, next it selects fillers from the concept's namespace, and finally it selects fillers from the role range's namespace.

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

```
// Step A: Create new terms.
for each ns in Namespaces
    ns-terms = identify all terms that belong to ns
    for each term in ns-terms
        n-term = create-new-term(ns, term)
        sof (n-term) = term
        tof (term) = tof(term) ∪ {n-term}

// Step B: Populate new parents.
for each n-concept
    concept = sof(n-concept)                    // 1
    for each parent of concept                  // 2
        candidates = tof(parent)                // 3
        n-parents = chooseP(candidates)         // 4
        add n-parents to n-concept

// Step C: Populate new roles.
for each n-concept
    concept = sof(n-concept)                    // 1
    for each role of concept                    // 5
        candidates = tof(role)                  // 6
        n-role = chooseR(candidates)            // 7
        add n-role to n-concept

        // Step D: Populate new fillers.
        for each filler of role                 // 8
            candidates = tof(filler)            // 9
            n-fillers = chooseF(candidates)     // 10
            add n-fillers to n-role of n-concept
```

Listing 1.

Verification

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

At the framework level, (1) we verified the correctness of framework mapping (table 2) by first comparing the asserted hierarchies of the new and GSPAS ontologies, and then by comparing the respective inferred hierarchies. The new asserted hierarchy had four missing subsumption links (out of 12,600+ direct links); these were manually added to the OWL ontology. Next, we manually compared the inferred hierarchies; most of the hierarchy matched; there were about 20 cases where a subconcept became equivalent to its parent. These cases were manually corrected in the new ontology. (2) Further, we verified 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, (1) we verify that every GSPAS term has a representation in the new ontology and that every new term description is part of some GSPAS term description. This is done by a reverse transformation from the new ontology to GSPAS ontology, by dropping the namespaces and merging terms. We manually compared the two versions of GSPAS ontology and found no significant differences. This verification alone 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. (2) Further, we checked for the case of punning using the 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 discussed in the Deployment and Maintenance section.

Performance Testing

In the GSPAS ontology, all terms are modeled as concepts, but primitive concepts that occur as leaves in the taxonomy, and without any role restriction, qualify as individuals. To explore alternate models of GSPAS ontology, qualifying individuals in the part-of-speech hierarchy and object hierarchy were modeled as individuals.

We created five OWL ontologies from GSPAS ontology (see table 3). Each differs in the number of individuals it models. (1) LEX_1 is the language ontology where leaves are individuals. (2) ONT_1 is the full ontology where all terms are concepts. (3) ONT_2 is ONT_1 with lexical leaves as individuals. (4) ONT_3 is ONT_2 with object leaves as individuals. (5) ONT_4 is

Case	Individuals	Individuals	Classes
LEX_1	lex leaves	6,780	317
ONT_1	none	0	12,815
ONT_2	lex leaves	5,679	7,136
ONT_3	lex and obj leaves	6,898	5,917
ONT_4	lex leaves minus nominals	5,136	7,679

Table 3. Ontology Test Cases.

	Language Profile		Classification Time		
Case	OWL	DL*	FacT++	HermiT	Pellet
LEX_1	OWL 2 EL	AL	0.2	0.8	0.7
ONT_1	OWL 2 EL	ALEH	1.6	12	4
ONT_2	OWL 2 EL	ALEHO	2.3	74	564
ONT_3	OWL 2 EL	ALEHO	2.7	352	716
ONT_4	OWL 2 EL	ALEH	1.7	13	4

Table 4. Classification Time (in Seconds).

In DL profile, AL stands for attributive language, E for existential restriction, H for subrole, and O for nominals.

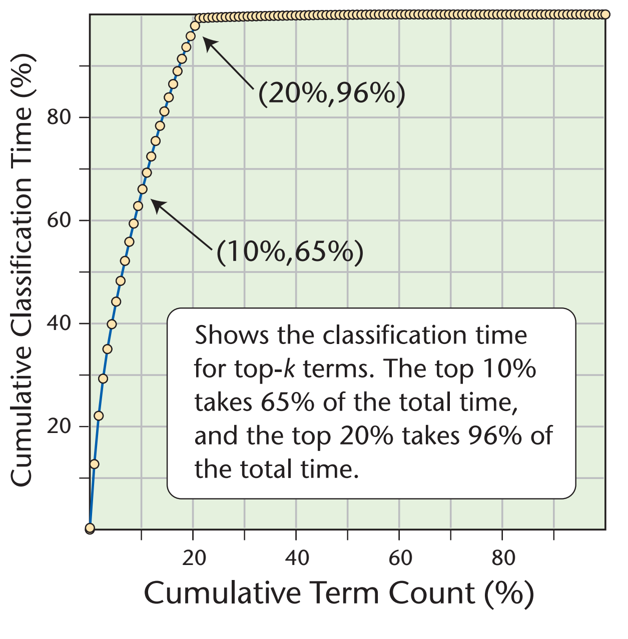


Figure 8. Pareto Chart. Time Versus Terms for ONT_3.

ONT_2 with nominals rolled back to concepts. The first four cases were created for performance testing, The fifth one was the result of performance tuning.

We tested three reasoners (FacT++ v1.6.3, Pellet v2.2.0, and HermiT v1.3.8.) on the five ontologies using Protégé v4.3.0 on Intel i7-4770 with 16 GB RAM running 64-bit Ubuntu 12.04. The execution times are given in table 4. We make the following observations: (1) Of the reasoners, FacT++ has the best overall performance, followed by HermiT and Pellet. (2) 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. (3) The performance, though within acceptable limits, begins to degrade for ONT_2 and ONT_3. HermiT and Pellet are up to two orders of magnitude slower than FacT++ for these ontologies.

To understand where the reasoner was spending time, we profiled ONT_3 using Pellet[3] and computed the classification time for each concept. Using this, a Pareto chart was prepared; see figure 8. Observe that 96 percent of the reasoner's time is spent in classifying 20 percent 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 outperforms ONT_2 and ONT_3, and has a comparable performance to ONT_1 (table 4).

From this we conclude that creation of individuals has 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 nominals.

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

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

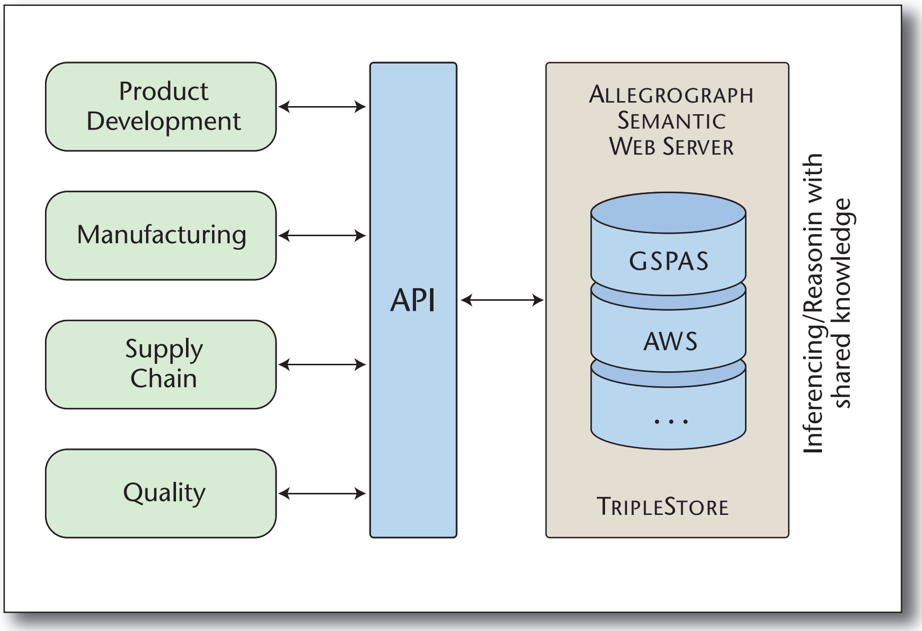


Figure 9. Ford Semantic Web Framework.

an automated set of regression tests that were run against the new OWL ontology. This is a set of more than 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. First, OWL representation was different than KL-ONE but was part of the reengineering process. In this case we adjusted the regression tests to reflect how the knowledge was represented in OWL. Second, 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. Third, 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 such as Top Braid Composer, which provides additional capability. The OWL/RDF system has proven to be easier to maintain and utilize for reusing knowledge.

The OWL ontology is also available for use through Allegrograph and is being utilized by other applications that need the information. Figure 9 shows the structure of our semantic web architecture.

Conclusions and Future Steps

In this article we described a project where Ford collaborated with the Indian Institute of Technology Madras to reengineer and convert an existing ontology into a semantic web OWL/RDF architecture. There were a number of compelling reasons that motivated the reengineering of the ontology from KL-ONE to OWL. The most important ones were based on 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 had to be rewritten several times as operating systems and hardware were being upgraded, and it was becoming a bottleneck for future ontology development. It was extremely tedious and time consuming to manually create reports and to extract knowledge from the KL-ONE ontology. 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, which 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 return on investment for this project includes a number of benefits that will pay dividends in the future. The standards and tools built around semantic technologies make our ontology easily accessible to other applications and will reduce future expenses in terms of maintenance and development costs. In addition, this project has helped us build the infrastructure needed to support semantic technology and allow for the development of other projects that could benefit from the semantic web.

Our future work will include the deployment of other ontologies into production as well as the use of semantic web tools and semantic web architecture for ontology development and maintenance. However, the real benefit will occur as we leverage semantic technology across other areas of the company and integrate this into our development and manufacturing processes.

Notes

1. International Resource Identifier (IRI).
2. International Resource Identifier (IRI).
3. In Pellet, concept classification is done by a series of subsumption tests. Pellet reports the execution time for each test, and we sum up these times to compute the classification time for a concept.

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, UK: Cambridge University Press.

Baader, F.; Brandt, S.; and Lutz, C. 2008. Pushing the EL Envelope Further. In Proceedings of the Fourth OWLED Workshop on OWL: Experiences and Directions. *Ceur Workshop Proceedings* Volume 496. Aachen, Germany: RWTH Aachen University.

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. Cambridge, MA: World Wide Web Consortium, W3C. (www.w3.org/TR/2004/REC-owl-ref-20040210).

Brachman, R., and Levesque, H. 2004. *Knowledge Representation and Reasoning.* San Francisco: Morgan Kaufmann Publishers Inc.

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

Corcho, O., and Gómez-Pérez, A. 2005. A Layered Model for Building Ontology Translation Systems. *International Journal of Semantic Web Information Systems* 1(2): 22–48.

Euzenat, J. 2001. Towards a Principled Approach to Semantic Interoperability. In Proceedings of the IJCAI Workshop on Ontologies and Information Sharing, 19–25. *Ceur Workshop Proceedings* Volume 47. Aachen, Germany: RWTH Aachen University.

Khemani, D. 2013. *A First Course in Artificial Intelligence*. Nolda, India: McGraw Hill Education.

Lipkis, T. A. 1981. A KL-ONE Classifier. In *Proceedings of the Second KL-ONE Workshop*. BBN Technical Report 4842, Cambridge, MA: BBN Laboratories.

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

Rychtyckyj, N. 1994. Classification in DLMS Utilizing a KL-ONE Representation Language. In *Proceedings of the Sixth International Conference on Tools with Artificial Intelligence,* ICTAI'94, 339–45. Los Alamitos, CA: IEEE Computer Society.

Rychtyckyj, N. 1999. DLMS: Ten Years of AI for Vehicle Assembly Process Planning. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence*, 821–828. Menlo Park, CA: AAAI Press.

Rychtyckyj, N. 2006. Standard Language at Ford Motor Company: A Case Study in Controlled Language Development and Deployment. Paper presented at the 5th International Workshop on Controlled Language Applications, Cambridge, MA, August 12.

Schmolze, J. G., and Lipkis, T. A. 1983. Classification in the KL-ONE Knowledge Representation System. In *Proceedings of the Eighth International Conference on Artificial Intelligence,* 330–332. Los Altos, CA: William Kaufmann, Inc.

Sullivan, B.; Carey, P.; and Farrell, J. 2001. *Heyde's Modapts: A Language of Work*. London: Heyde Dynamics Pty, Ltd.

Nestor Rychtyckyj is a senior analytics scientist for global data insight and analytics at Ford Motor Company in Dearborn, Michigan. His responsibilities include the application of machine learning, natural language processing, semantic computing, and machine translation for manufacturing, quality, customer interaction and cybersecurity. Previously, Rychtyckyj was responsible for the development and deployment of AI-based systems for vehicle assembly process planning and ergonomic analysis in manufacturing. He received his Ph.D. in computer science from Wayne State University in Detroit, Michigan. Rychtyckyj is a senior member of AAAI and IEEE and a member of ACM.

Venkatesh Raman is a senior data analyst for global data insight and analytics at Ford Motor Pvt. Ltd. in Chennai, India. His responsibilities include leveraging the big data platform and tools for analyzing and applying machine learning to connected vehicle data. Previously, Raman was with the Enterprise Technology Research group wherein he was researching the big data domain and evangelizing it. He received his master's degree in computer science from MS University in India.

Baskaran Sankaranarayanan is a researcher in the Department of Computer Science and Engineering, IIT Madras, India. He has more than 10 years of industry experience in designing, developing, and deploying large-scale data cleansing, data integration, and OLAP applications for retail, banking, financial services, health care, credit rating, and magazine domains. He is interested in the application of ontology to real-world problems. His long-term goal is to develop efficient data integration frameworks. He holds a master's degree in structural engineering from IIT Bombay, and a bachelor's degree in civil engineering from University of Madras.

P. Sreenivasa Kumar is a professor in the Department of Computer Science and Engineering (CSE), IIT Madras, India. He was also the head of the Computer Science and Engineering Department during the years 2013–2015. His research interests include database systems, semistructured data and XML, ontologies and semantic web, data mining, graph algorithms, and parallel computing. He earned his bachelor's degree in electronics and communication engineering from the Sri Venkateswara University College of Engineering, Tirupati, India. His master's and Ph.D. degrees are in computer science from the Indian Institute of Science, Bangalore, India.

Deepak Khemani is a professor in the Department of Computer Science and Engineering, IIT Madras, India. His long-term goal is to build articulate problem-solving systems that can interact with humans, currently looking at contract bridge. He works in memory-based reasoning, knowledge representation, planning, constraint satisfaction, and qualitative reasoning. He graduated with three degrees from IIT Bombay, including two in computer science. He is the author of *A First Course in Artificial Intelligence.*