Practical Experiences in Trading Off Ontology Usability and Reusability

Thomas Russ, Andre Valente, Robert MacGregor and William Swartout
University of Southern California, Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA 90292, USA
e-mail: {tar,valente,macgregor,swartout}@isi.edu

University of Southern California, Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292, USA, valente@isi.edu

Abstract

This paper describes the practical application of ontologies in designing knowledge-based systems. We built a common ontology to allow knowledge sharing among several applications in the domain of air campaign planning. In trying to reuse this shared ontology, we confirmed previous findings in the knowledge acquisition literature: it is difficult to simultaneously obtain high usability (i.e., adequacy for a specific use) and reusability (i.e., adequacy for several uses). In this article, we discuss some of the reasons for this trade-off, and present some lessons learned about the limitations of current tools for ontology construction in practical applications.

1 Introduction

In the last few years, first with the ARPA Rome Planning Initiative (ARPI) program, and later with the DARPA Joint Forces Air Component Commander (JFACC) program, several groups within the University of California's Information Sciences Institute (USC/ISI) have developed applications for the domain of air campaign planning. These efforts were initially separate, since they were different projects funded by different contracts, with rather different technologies and goals. However, after a while it became apparent that it would be impractical and inefficient (particularly from a cost perspective) for these groups to develop their own knowledge bases in isolation. We decided to collaborate in developing a shared ontology that would allow us to pool resources, share experiences and best practices, and generally obtain better quality with lower development cost.

The result of this process was a relatively broad shared ontology of air campaign planning, called the *JFACC ontology*. In this paper, we relate the experience and the lessons learned in building the JFACC ontology and re-using it in several applications. In developing this ontology, we had several goals:

1. Facilitate inter-operation and communication between systems by providing a common terminology. This is important because the JFACC software was being developed by several different groups at ISI and some groups outside ISI.
2. Promote sharing of knowledge between systems. In particular, we wanted to integrate our knowledge acquisition and modeling efforts. Even in our own organization different groups were developing related knowledge bases and ontologies for their own applications. These ontologies were all in the same domain, frequently with a high degree of commonality, but also sometimes with marked modeling differences that stem from different degrees of access to experts, documentation, etc.
3. Create a repository for general knowledge about air campaign planning that could be used across the

broader research program in several applications, including traditional ones (that is, not knowledge-based).

The JFACC Ontology is represented in the knowledge representation system Loom (MacGregor, 1991). It is relatively large in size, currently comprising about 1,500 entities, of which 1,000 are concepts, the rest relations and instances.

The emphasis of this paper is to use the experience in developing the JFACC Ontology to provide a "report from the trenches" on employing the state of the art in techniques for building, using and reusing ontologies of a reasonable size and putting them to practical use. What we found is that developing a large ontology that actually needs to be used by more than one application is sometimes a painful process, and that many of the techniques proposed in the literature do not work as easily in practice as we would have hoped.

We will start with an overview of our JFACC work (Section 2), then we will discuss what problems were found in building and using the JFACC ontology, how these problems were solved, and what issues arose in the process. Specifically, we will focus on how we solved technical problems which are instances of three general problems (see Figure 1):

1. How to *translate* and import a publicly available, sharable ontology originally written in another knowledge representation formalism (Section 3.1).
2. How to *merge* the contents of two existing, independently developed knowledge bases into an ontology (Section 3.2).
3. How to *import* knowledge from an ontology into several applications, and how to *extract* part of the knowledge from the ontology to use in a knowledge-based application that needs the knowledge in another format (Section 4).

The first two problems (top of Figure 1) are instances of the general problem of *ontology reuse*; the third problem (bottom of Figure 1) is an instance of *ontology use*. We will detail what issues we found in trying to solve these problems, and emphasize the lessons we learned in the process. In Section 5 we present our conclusions.

2 Developing Applications for the Air Campaign Planning Domain

Military air campaign planners develop plans for using aircraft to achieve a variety of objectives. For a number of reasons, air campaign planning is a very complex process. First, there are a number of interacting planning concerns. In addition to figuring out how aircraft will be used, one must also create a plan for supporting the aircraft by supplying the necessary fuel, spare parts, munitions and so forth. Today, these different planning processes proceed in a fairly independent fashion, leading to serious problems when conflicts are discovered. Second, the kinds of operations in which the military may become involved are now much broader. Whereas once the military planned for large-scale battles with major opponents, now in addition to traditional military tasks, the armed forces must be prepared to deal with a wide variety of tasks that do not involve the application of force, such as non-combatant evacuation or humanitarian relief operations. Third, the nature of air campaign planning itself has changed and become more deliberate with the development of the "strategies-to-task" approach. Previously, air campaign planning was primarily bottom-up, focusing on military targets and planning operations to attack them. This approach is clearly less appropriate for non-military operations, but even for military operations a problem was that the rationale

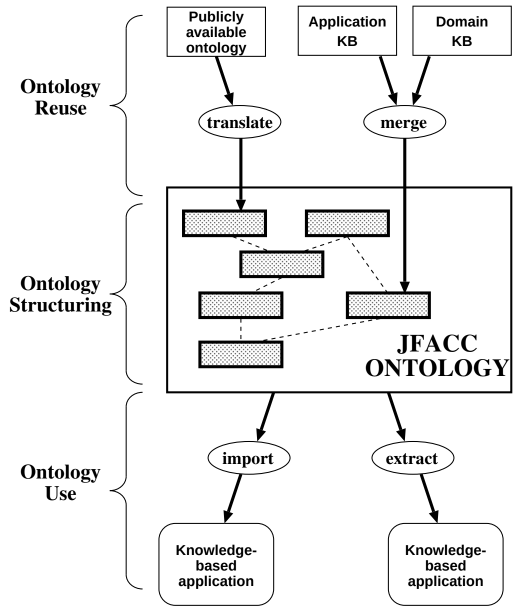

FIGURE 1: Using and reusing ontologies in the construction and employment of the JFACC Ontology.

behind the plan was often lost, so it was not clear what objectives were being served by attacking a particular target. In the strategies-to-task planning approach, high level objectives (such as "provide relief to refugees in the southern region of country A") are decomposed into increasingly more specific objectives until one has planned specific tasks for particular aircraft.

At ISI, several projects developed technology and applications to support air campaign planning:

- The Loom group, headed by Robert MacGregor, used its knowledge representation language Loom to develop the Strategy Development Assistant (SDA). The SDA is a mixed-initiative tool to help planners decompose their objectives into sub-objectives. SDA provides support for intelligent, guided plan development, following a theory of air campaign plan decomposition based on the strategies-to-task approach. SDA makes extensive use of Loom's representation and reasoning facilities. The Loom group is also developing ontologies for JFACC and also general purpose tools for ontology construction, and is the developer of the Loom and PowerLoom knowledge representation languages (MacGregor, 1991). One of the ontology development tools produced by this group is Ontosaurus (Swartout *et al.*, 1996)

- The EXPECT group, headed by Yolanda Gil, has developed a plan evaluation tool called INSPECT (Valente *et al.*, 1996). INSPECT is designed to critique air campaign plans that people have entered. Like the SDA, INSPECT's knowledge base is written in Loom.

- The Mastermind group, headed by Pedro Szekely, is developing an Objectives Editor that allows a user to enter air campaign objectives into an *Adaptive Form* (Frank & Szekely, 1998). Objectives are represented as structured verb clauses using a case grammar approach.

We represented the JFACC Ontology in Loom (MacGregor, 1991), a knowledge representation framework based on description logics (Patel-Schneider *et al.*, 1990). Like other description logics, Loom is based on a semantic network approach to knowledge representation. It is possible to define *concepts* in Loom. Concepts can have *roles* or slots which may be used to specify attributes of the concept. A key feature of description logic representations is that the semantics of the representation language are very precisely specified. This precise specification makes it possible to build a tool, called a classifier, that can determine whether or not one concept *subsumes* another based solely on the formal definitions of the two concepts.[1] The classifier is an important tool for building ontologies because it can be used to organize a set of Loom concepts into a hierarchy automatically, based solely on their definitions. This capability is particularly important as the ontology becomes large, since the classifier will find subsumption relations that people might overlook, as well as modeling errors that could make the knowledge base inconsistent.

2.1 Contents of the JFACC Ontology

The JFACC Ontology was created to represent a wide variety of knowledge content in the air campaign domain. However, its main function was not to describe that domain but to support of the applications described above. Therefore, its contents reflect more the specific needs we found for knowledge in these applications and less a grand vision of what air campaign planning is about. For instance, it has representations for many of the basic concepts one would expect (time, objectives, plans), but other concepts (such as action and space) are represented in a somewhat simplified (and pragmatic) manner, since it was not yet necessary to use a more complete and principled representation of these kinds of knowledge.

In constructing the JFACC ontology, an important issue was how to organize the ontology so as to make its maintenance easier and its construction more rational. Following almost standard guidelines found in the literature, we decided to modularize the ontology using the mechanism of *contexts* provided by Loom. The JFACC ontology is organized into several Loom contexts:

Systems This ontology defines general systems, their decomposition into subsystems and primitive components, their inputs and outputs, etc. It also covers the definition of networks as a special kind of system. This knowledge was very useful, for example, to model enemy systems varying from distribution to transportation networks to military systems for command and control.

Time Time is a fundamental element of any planning domain, and air campaign planning is no different. The ontology of time we used contains definitions for time points and intervals, dates (absolute and relative) and many relations between time point and intervals.

Entities This ontology contains the definition of general types of entities that are shared among many other component ontologies. It also contains extensive hierarchies of types of targets, military actions, action capabilities, and abstract characteristics of the situation or state that are used frequently when defining air campaign plan objectives.

Air campaign plans This ontology contains an overview of the basic elements that characterize air campaign plans, such as: campaign, scenario, participants, commanders, plans, phases, objectives, etc.

Weapons includes definitions and extensive information about missiles, guns, bombs, and munitions.

Fuel contains definitions and data for the main types of fuel and additives used by US military aircraft.

Aircraft This ontology, discussed in Section 3.2 below, contains knowledge about types of aircraft in the US military, including data about engines, pods and fuel tanks these aircraft can carry.

Forces contains definitions about military force units, including definitions for military bases and installations. This information is particularly relevant for reasoning about logistics.

[1]A concept A is said to subsume a concept B if all the possible entities that could be described by B are also necessarily described by A. For example, "a man who only drinks beer" subsumes "a man who only drinks imported beer." More details about subsumption can be found in (Woods, 1991).

Grammar Some of the knowledge about objectives in air campaign plans is used to produce case grammars that enable a syntax-oriented editor to define these objectives in a structured manner. This ontology contains the concepts and relations necessary to map the representation of objectives in the air campaign plans ontology (see above) into the grammar representation. This is explained in detail in Section 4.

3 Building the JFACC Ontology

In building the JFACC ontology, we tried to use as much as possible the formally described knowledge that was already available about the domain. Whenever feasible, we preferred to use an existing ontology or knowledge base instead of developing our own. We used knowledge bases from implemented applications, theories of fundamental concepts such as time and system, and ontologies with extensive data about specific elements in a domain (i.e., aircraft, weapons). In the following sections, we discuss in detail the experience we have in two of these reuse processes. First, we discuss how we imported an existing and publicly available ontology of time into the JFACC ontology. Second, we discuss how we merged two ontologies of aircraft that had been developed for different uses into a common shared ontology.

3.1 Re-using an ontology of time

Time is an essential constituent of the air campaign domain. We wanted a complete and well-founded ontology of time, but wanted to avoid the time-consuming formalization and representation process. We therefore looked for a pre-built and publicly available ontology.

After some search, we found a satisfactory ontology of time (based on Allen's theory of time (Allen, 1984)) in the Sharable Ontologies Library at Stanford.[2] It was part of a job assignment ontology developed by (Hama *et al.*, 1992). In trying to reuse this ontology for JFACC, we found several problems, all of them loosely connected to the fact that we had to use an Ontolingua translator to import the original ontology (written in Ontolingua) into our targets knowledge representation language (Loom).[3] The following discussion focuses on Ontolingua and Loom, since those are the systems that we used, but the discussion is broadly applicable to description logics other than Loom. Translating from Ontolingua into a more limited knowledge model, such as OKBC[4] would not encounter similar problems, but that is mainly because the OKBC knowledge model is not sufficiently expressive to allow the problems to surface. The Ontolingua translator was useful for producing a first draft of a Loom ontology, but the resulting translation had several problems. Therefore, we had to make an extensive manual adaptation of the translated ontology.

The problems with the translated ontology from Ontolingua were:

- The ontology that resulted from the translation process was dependent on a general KIF theory for frames (the frame ontology). This is a consequence of the fact that Ontolingua itself is based on the frame ontology, that is, the Ontolingua constructs for frames are built on top of the basic elements of KIF. One easy solution for this problem would have been to import this theory into our knowledge base as well, but we chose not to do that. Many of the frame ontology commitments already existed in

[2] URL http://www-ksl.stanford.edu/knowledge-sharing/ontologies/README.html.

[3] For details on Ontolingua and its translators, see the pages on Ontolingua at the Stanford University Knowledge Systems Laboratory http://www-ksl.stanford.edu/knowledge-sharing/ontolingua/ontolingua.html.

[4] The Open Knowledge Base Connectivity protocol. See http://www-ksl-svc.stanford.edu/okbc/.

Loom, and to take advantage of Loom's built-in reasoner, we needed to recast the Ontolingua forms in the Loom idiom (Figure 2).

;;; Produced by the Ontolingua translator.

```
(loom:defrelation meets
   :context jat-generic
   :is-primitive loom:binary-tuple
   :arity 2
   :annotations
   ((related-axioms
     '(<=>
        (meets ?tr1 ?tr2)
        (tp= (time-range.end-time ?tr1)
             (time-range.start-time ?tr2))))
    (documentation
       "a time range ?tr1 ends at the same time
a time range ?tr2 starts.")))
```

;;; Modified ("loomified") version.

```
(defrelation meets
 :is (:satisfies (?tr1 ?tr2)
       (:and
         (time-range ?tr1)
         (time-range ?tr2)
         (tp= (time-range.end-time ?tr1)
              (time-range.start-time ?tr2))))
  :arity 2
  :domain time-range
  :range time-range
  :annotations
  ((documentation "a time range ?tr1 ends at the
same time a time range ?tr2 starts.")))
```

FIGURE 2: Definitions of `meets` in the ontology translated from Loom and the modified version used in the JFACC Ontology. Notice how the contents of the `related-axioms` annotation on the definition produced by the translator (left) is almost the same as the contents of the `:satisfies` clause in the "loomified" definition (right). The expression starting with <=> can be represented in Loom, since it defines when the relation `meets` holds between two time ranges, but in order for Loom to reason with the expression, it must be part of the definition of the concept (`:is` clause).

- Even though Ontolingua is more expressive than Loom, Loom has constructs for frequently used types of definitions that allow for simpler definitions than a literal translation produces. However, since the translation software did not exploit all of Loom's capabilities, the translated ontology was unnecessarily complex.

;;; Produced by the Ontolingua translator.

```
(loom:defconcept day-number
  :context jat-generic
  :is-primitive loom:thing
  :annotations
    ((related-axioms
        '(= (i-upper-bound day-number) 31))
     (related-axioms
        '(= (i-lower-bound day-number) 1))
     (documentation
        "DAY-NUMBER denotes a day of a
month.")))
```

;;; Modified ("loomified") version.

```
(defconcept day-number
  :is (:through 1 31)
  :annotations
    ((documentation
        "DAY-NUMBER denotes a day of a
month.")))
```

FIGURE 3: Definition of `day-number` in the ontology translated from Ontolingua (right) and the modified version used in the JFACC Ontology (left). The definition of `day-number` in the ontology translated from Ontolingua is made by using the relations `i-upper-bound` and `i-lower-bound`, which relate to the upper and lower bounds of an integer interval. In contrast, the "loomified" definition (on the right) uses the built-in operator `:through` to say the same thing and allow the Loom inference engine to efficiently reason about day numbers.The resulting definition is much simpler, and also allowed us to dispense with lengthy definitions for types of intervals and relations and axioms about them.

These problems had two main causes. First, there is a mismatch in modeling styles: the way knowledge is modeled in Ontolingua is different from the way it is normally modeled in Loom. Frame systems and description logics, we found out, are not as close as they sometimes seem to be. Moreover, knowledge representation systems and languages are never completely neutral. Either by the constructs they provide or by an upper level ontology they assume, knowledge representation systems and languages assume a view of how the world is to be modeled, and even what fundamental parts there are in the world. Frequently, the constructs of the language (e.g., a `defconcept`) are shortcuts to express important relations or sets of relations in terms of the top ontology. In Loom, for example, the top ontology starts with `thing` and contains definitions for several types of numbers (or, more generally, constants), for several relations between concepts and relations, for differentiation between concepts and meta-concepts, etc. Likewise, Ontolingua is based on a top ontology that defines terms such as frames, slots, slot values, facets, etc. Therefore, any translation between two knowledge representation languages requires some kind of mapping between the top ontologies of the two languages. Constructing a translator is in general easier if the top ontologies are similar. In the case of Loom and Ontolingua, the similarity is reasonable but not very high, which explains why producing a good translation is feasible but to produce an excellent one is very difficult.

Second, and more interestingly, there seems to be an *inference engine bias* in modeling. Even when there is no bias towards tailoring the knowledge to be used by a specific application or problem solving method, knowledge is usually modeled with certain types of inferences in mind. For example, if we expect to use the Loom classifier to infer whether or not two intervals meet (that is, `(meets int1 int2)`, we need to add enough information in the definition of the relation `meets` to enable the classifier to use it. If, however, all we want to do is to *assert* that the intervals meet and use this information for other inferences, it is enough to state the range and domain of the `meets` relation.

In contrast, Ontolingua did not have an implemented reasoner. Consequently, the inference bias either reflects the use the ontology had originally (if it was produced for specific application), or tends to approximate some abstract form of theorem proving that is "natural" when one establishes logical definitions.

One interesting example of the difference in ontology development that the inference engine bias introduces can be seen in the definition of a `Rational-Number` in Ontolingua and what happens when that definition is translated into Loom. The Ontolingua definition includes the following axiom:

```
    (<=> (Rational-Number ?x)
 (and (Real-Number ?x)
      (Exists (?y)
 (and (Integer ?y) (Integer (* ?x ?y))))))
```

This axiom can be translated (manually) into its equivalent form in Loom:

```
    (defconcept ontolingua-rational-number
       :is (:satisfies (?x)
      (:and (number ?x)
    (:exists (?y)
       (:and (integer ?y) (integer (* ?x ?y)))))))
```

Unfortunately, trying to use this definition in Loom causes the following error message to be issued:

```
    Query formation error:
       Can't generate bindings for the variable ?Y.
    Possible causes are
     o  ?Y is universally quantified and all generator expressions
for ?Y are open world
     o  All generators found for ?Y are constant concepts like
NUMBER or STRING, which cannot be used.
```

Interestingly, the source of the error is not a mistake in the meaning of the definition itself, or its syntax, but the fact that Loom is trying to reason with the definition. The Ontolingua definition is a perfectly valid description of what it means to be a rational number. Indeed, one might find a similar definition in any mathematical textbook. Unfortunately, the definition does not offer a practical, constructive method for determining whether any given number is, in fact, a rational number. Consider what the definition tells you: In order to figure out if `?X` is a rational number, one merely needs to find some integer `?Y` such that `?X` times `?Y` yields an integer. Mechanistically how would a computer program (such as Loom) use this definition? First, enumerate the integers and test them, stopping when a suitable `?Y` was found. While theoretically sound, this approach suffers from efficiency problems. Loom recognizes this and knows enough not to try — thus the error message. In this particular case it is the second of the two possible causes of the problem that apply.

This type of error is not peculiar to the use of Loom. It would occur in any system that tried to use the definition of a rational number to figure out if a given object is a rational number. While Loom and other description logics provide a means to specify definitions which are not used in reasoning, one needs to know whether a given definition would encounter this problem. It is not possible to determine this from syntactic analysis alone! If `INTEGER` were a finite set, then the definition would be effectively computable.

Lessons Learned. In our work in trying to translate and import a publicly available ontology of time, we learned that translator technology is still immature. Translation of ontologies written in different formalisms is generally a difficult task. The Ontolingua-to-Loom translator is clearly problematic, but in fact creating a better one is by no means easy. We found that the result of the automatic translation is still interesting as a draft, but lots of human intervention are required to translate Ontolingua into Loom. This seems to be caused by at least two problems.

First, it is rarely recognized in constructing these translators that they must somehow bridge the gaps between the underlying models and views used in the formalisms (e.g., mapping frames and slots into concepts and relations) as well as between the "upper models" assumed in the development of the ontologies in each formalism (e.g., the frame ontology in Ontolingua and the representation of numbers in Loom's `built-in-theory`.

Second, ontologies usually contain an inference bias that makes mapping even more difficult, and importing even more time-consuming. A large part of the literature on ontologies, particularly on the formal/logical side, treats an ontology as an end in itself. They picture these kinds of biases as impurities in the ontology. However, for system builders, who use ontologies as a means to improve the engineering of knowledge-based systems (e.g., by allowing reuse), inference bias is a feature, not a bug. The challenge is not to get rid of this bias, but to create translators that allow to (re)introduce (under the user's command) in the translation process. In other words, translators need to take into account not only the "meaning" of the descriptions or definitions in the ontology, but how these constructs are going to be used. This means that there should be several types of mappings used as a basis for the translation process, and users should be able to choose the mapping most adequate to their situation.

3.2 Merging two ontologies of aircraft

Before building the JFACC ontology, we had already developed two ontologies of aircraft. The first one, called the Aircraft ontology, was a domain ontology developed from Fact Sheets published by the US Air Force. It contained extensive data about aircraft types in the US Armed Forces. The upper structure was developed by one of the authors. This ontology was developed to demonstrate the Ontosaurus browser and to showcase particular features of the Loom knowledge representation language, in particular Loom's ability to classify concepts and instances, as well as its ability to reason with and support relation (slot) hierarchies.

The second ontology for aircraft was part of the knowledge base of the INSPECT system for critiquing air campaign plans. Interestingly, these two ontologies, dealing generally with the same domain, had actually focused on different aspects. That meant that, although they were about the same domain, there was relatively little overlap between the information present at the detailed level. To construct a module with broader knowledge about aircraft for the JFACC ontology, we decided to merge the two ontologies.

The lack of overlap between the two original ontologies made the merging process easier, since it meant that many things could be merged by a simple union process. In the one area where there was significant overlap — missions and functional type of aircraft — we were forced to make choices as to which model to prefer. In our case this was relatively easy, at least in the abstract, since only one of the ontologies was developed to support a particular application. Since the INSPECT ontology was developed in part based on interviews with subject matter experts (SMEs), it was a natural candidate to prevail whenever there were conflicts in the ontologies.

In practice, however, the merging proved trickier than one might have expected. In addition to some major philosophical differences, there was also a difference in naming conventions. The Aircraft ontology used hyphens in the names of aircraft types whereas the INSPECT ontology did not. This minor annoyance motivated us to construct a tool for automatically translating between a variety of naming conventions.

A more difficult problem is that there were substantial structural differences between the two ontologies. One major example was in the method used to determine whether particular aircraft models were, for example, fighters or bombers. In the Aircraft ontology, this classification was inferred based on the missions the aircraft could perform. In INSPECT, the mission were not directly related to the aircraft types. It turned out that since in the aircraft knowledge base type memberships were all inferred by Loom's classifier, switching from one system to the other was easier that it otherwise might have been. There were no direct type assertions about these type memberships about individual aircraft in the Aircraft knowledge base, so changing to the INSPECT structure didn't require deleting any assertions in the Aircraft ontology.

Although not required by the merging process itself, we did, however, take the opportunity of doing the merge to reexamine a fundamental question in the models: Should aircraft types be modeled as concepts or as instances?

The general issue of whether to model particular items in a domain as instances or concepts can be difficult to resolve when a knowledge base terminates in abstract entities. It also depends on the particular use for which a ontology is designed (see, for example, (Brachman *et al.*, 1991)). The fact that such usage influences a fundamental modeling issue is unfortunate from the point of view of making ontologies reusable and easily applied for applications that were not envisioned at the time of the creation of the ontology.

When the items being modeled in the world are "naturally" individuals, this is less of an issue. For example, if one were creating a ontology describing Air Force aircraft, then having instances correspond to particular airplanes such as "F-16 serial number 87-0217" there is relatively little controversy. The issue arises mainly when one wishes to not represent the world at quite so detailed a level. In that case, one needs to face the choice of whether one wishes to have particular models (classes) of aircraft represented in the ontology by an instances. One could, for example, model the F-16C as an instance instead of a concept.

This was the approach taken in the Aircraft ontology. This lead to a natural definition of the instance as having particular fillers for attributes such as crew-size, range, payload, etc. An example of this hierarchy is shown in Figure 4.

The same approach was used in the INSPECT ontology, which, if left unchanged, would have made the job of merging easier. Unfortunately, this modeling choice began to exhibit some breakdowns in functionality. As one can see in Figure 4, there were multiple models of aircraft being represented as instances. As long as these were conceptually at the same level of detail (say "F16C" and "F16D") the solution still seemed to work reasonably well. The problem was that INSPECT would also refer to less specific aircraft types such as "F16" when there wasn't a need to differentiate. (The main difference between the "C" and "D" models is that the C model has one crewman and the D model two crewmen and less fuel.)

An instance representation therefore loses the relationship between these ideas, since the fact that F-16Cs

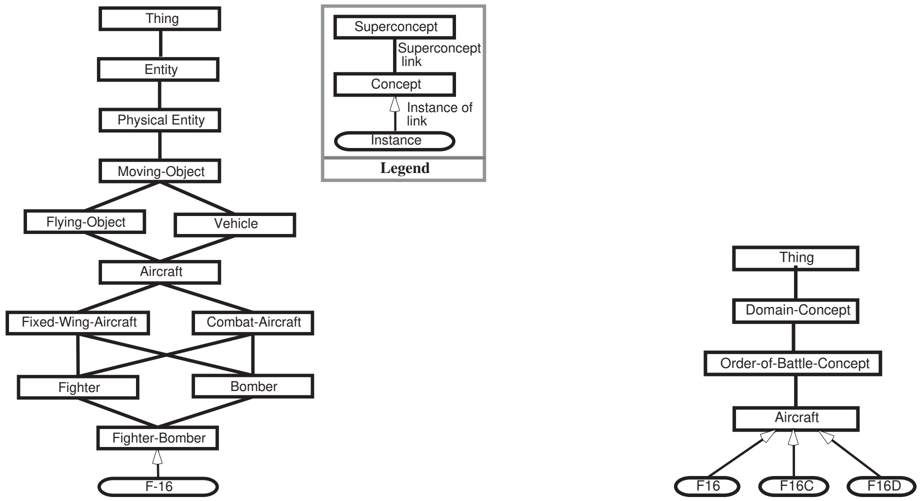

FIGURE 4: F16 Hierarchy in the Aircraft (left) and INSPECT (right) Ontology.

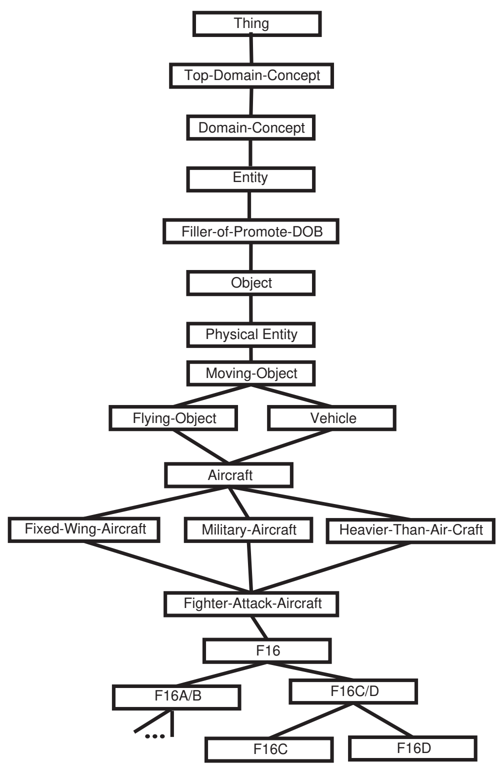

FIGURE 5: F16 Hierarchy in the JFACC Ontology.

and F-16Ds are both F-16s is lost. This indicates a need to move (at least) the "F-16" object into the concept hierarchy.

Why didn't this missing connection in the ontology hurt the INSPECT project? Because INSPECT was never required to reason about the relationship between various models of F-16. We suspect that it is the use of shortcuts like this that leads to some of the brittleness that expert systems and other computer software exhibit. It is also another example of the inference bias in ontology construction. Since this particular inference was not needed, the fact that the model did not support the inference had no immediate practical impact on the application. On the other hand, this limited the reusability of the software in other applications where the relationship between different models of F-16 is important. Since we wanted to enhance the reusability of the system, we corrected this shortcoming of the previous model.

Once that was done, it was possible to consolidate some of the common data about the aircraft into the concept definition. This then caused the difference in representation between the F-16 concept and the individual models to appear rather artificial. We therefore raised all of the aircraft class information to the concept level in the JFACC ontology. This resulted in the structure seen in Figure 5.

In switching from an instance-based to a concept-based representation, much of the data could be transferred in a relatively straightforward manner. For example, the fillers of roles like length and wingspan were easily transferred into a concept representation. Some of the information, however, was not so easily transferred. Since the instances were really describing classes, some of the assertions of fillers were really the meta-assertions about the structure of the class. For example, the "engine-count" and "engine-type" relations are descriptors about the fillers of the engine role. When moving to a concept representation, it was necessary to change these into a form that more precisely matches the description logic. In this case it was transformed into a number restriction on the "engine" role and a (disjunctive) type restriction on "engine". This change is an improvement in the model, since it more clearly represents the world in terms that the Loom classifier can reason about.

```
;; Instance
(:about F-16
     fixed-wing-aircraft

     (name "Fighting Falcon")
     (designation 'F-16C)
     (made-by General-Dynamics)
     (length 49)
     (height 16)
     (wingspan 33)
     (max-speed 1500)
     (engine-count 1)
     (engine-type F100-PW-200/220)
     (engine-type F110-GE-200))
```

```
;; Concept
(defconcept F16
  :is-primitive
     (:and fighter-attack-aircraft ...)
  :constraints
     (:and (:filled-by name
                        "Fighting Falcon")
           (:filled-by made-by
                        General-Dynamics)
           (:filled-by length 49)
           (:filled-by height 16)
           (:filled-by wingspan 33)
           (:filled-by max-speed 1500)
           (:exactly 1 engines)
           (:all engines
                 (:or F100-PW-200/220
                      F110-GE-200))))
```

One constraining factor is that although the new ontology made it possible to set up an appropriate hierarchy and thus descriptions of the instances in terms of their fillers, it made it more difficult to write queries that returned the same information that was so easy to retrieve in the INSPECT ontology. The reason the INSPECT ontology had easier retrieval is because the model had a direct encoding of the meta-information, rather than using the Loom modeling language's constructs for expressing restrictions. This reification process meant that the domain over which INSPECT reasoned was the meta-information about the aircraft.

The easier encoding of the meta-information also made it easier to be sloppy. The fact that the `engine-type` slot's fillers represents a set of potential engine types is implicit in the ontology design. Nowhere is the information available in a machine-readable form. In situations like this, the information is either absent from the ontology entirely or it is available only in a documentation string or comment.

When the meta-information was stored simply as assertions about instances, then writing a query to retrieve the information was easy. In the Aircraft ontology for example, if one wanted to get the type(s) of engine that an F-16 had the following query sufficed:

```
(retrieve ?engine (engine-type F-16 ?x))
      => (|i|F100-PW-200/220 |i|F110-GE-200)
```

When going to a concept representation, this expedient was no longer as convenient. In part this is a limitation in the Loom modeling language, since there is no convenient method by which one can make meta-assertions about the concepts *and have the classifier and definition machinery handle them automatically.* Instead we resorted to establishing a canonical place in the definitions where we added the information and then defined appropriate relations for retrieving the information as needed. This negates one of the advantages of the automatic inference in description logics—the ability to use any one of several equivalent descriptions with the inference system maintaining logical consistency.

To get similar information from the JFACC ontology becomes a lot trickier. That is not only because the information is encoded differently, but also because by using a type restriction we end up having to do more reasoning to figure out exactly what answer we want. This is because the type restriction constructed by the concept definition is no longer a concept that has a direct analog in the real world, but is instead a disjunction of such real-world engine concepts: `(:or F100-PW-200/220 F110-GE-200)`. To find the two real-world engines, we need to extract the type restriction from the `engine` role and then find the appropriate subconcepts of that restriction. In order to be able to do this mechanically we need to introduce a meta-annotation (`engine-model-class`) on the concepts that represent real world engines and then use this meta-information to filter the subconcepts of each restriction. This is a bit cumbersome, but still not too problematic.

```
(defconcept F100-PW-200/220
  :is-primitive (:and afterburning-engine turbofan-engine)
  :constraints (:filled-by made-by pratt-&-whitney)
  :defaults (:filled-by thrust 27000)
  :annotations (engine-model-class))   ;; Annotation of meta-class

(retrieve ?engine
      (:and (engine-model-class ?engine)
             (:or (role-types F16 engines ?engine)
                  (subconcepts (role-types F-16 engines) ?engine))))

    => (|C|F100-PW-200/220 |C|F110-GE-200)
```

Where this change in representation really hurts is in trying to find the aircraft which use a particular engine. Since there is no direct relation between the type in a type restriction and the concepts for which it restricts a role value, one is forced to search all aircraft models and look for those whose `engine` role has a type restriction that includes (subsumes) the engine in question. For ease of access, we can encode such a query in a relation in the knowledge-base, and thus use a defined term to simplify future queries:

```
(defrelation engine-type-can-be-used-by
    "This relation is a meta-inverse relation.  It returns concepts in the
```

```
 meta-class AIRCRAFT-MODEL-CLASS that use ENGINE-MODEL-CLASS as a type
 restriction on the relation ENGINE"
     :domain aircraft-model-class :range engine-model-class
     :is (:satisfies (?x ?y)
             (:and (aircraft-model-class ?y)
                        (engine-model-class ?x)
                    (:or (role-types ?y engine ?x)
                         (subconcepts (role-types ?y engine) ?x)))))

(retrieve ?engine (engine-type-can-be-used-by F16 ?engine))
    => (|C|F100-PW-200/220 |C|F110-GE-200)
```

Lessons Learned. In our work in trying to merge two knowledge bases into a richer ontology of aircraft, the first lesson we learned is a positive one. Merging was worth the trouble. The end result obtained was much richer than the initial products, and richer than what we would have been obtained if the product had been built from scratch. The richness comes from the fact that the two individual ontologies explored different views on the domain, and by trying to integrate these views in the merging process we obtained an ontology that incorporates both views. In fact, the result was so stimulating that we plan to do more merging of ontologies in the JFACC project.

However, we also learned that even within a specific formalisms one can use different modeling styles, and that merging needs to take them into account. Again, modeling is based on the intended use of the ontology or knowledge base, and thus even within a single representation formalism, modeling is never completely neutral.

In regard to modeling choices, there is a set of opposing forces that act on ontology designers. A decision to reify part of the model—that is, to have the domain of discourse be the meta-information instead of encoding the meta-information using Loom's description constructs—makes a program's meta-level reasoning easier to implement. The downside is that it also makes the resulting ontology harder to share with others or to reuse. That is because the reification steps represents a commitment to the level at which reasoning will be done. One methodology that suggests itself is to institute a review process into the ontology construction that seeks to identify these shortcuts and recast them in a more appropriate form. Sticking to the modeling terms defined by the modeling language and limiting the use of application-specific conventions facilitates reuse of the resulting ontology. Since this is an area in which there are competing interests, it will likely be the target of engineering tradeoff decisions.

Since one would like to be able to easily reverse the consequences of such a tradeoff decision, one would like to make modifying and updating the model simpler. In Loom, this need appears in the commonly occurring problem of transforming instances into concepts. Our conclusion is that tools for supporting the merging of two ontologies are sorely needed. While certain parts are inherently manual, the process can be made much easier if a user is able to express in general terms how the mapping should occur, e.g., this concept maps to this instance, this relation's filler are mapped into that relation's restrictions, etc. This calls for a tool that incorporates a language to talk *about* ontologies, their relations and relations among their components. Mappings should be described declaratively in this (meta-)language. To this end, concept representations like the one used in Loom can be less convenient because they are more general, but harder to manipulate. It is important to find a graphical representation for this (meta-)language, that can be used in constructing tools that are visually rich and easy to use.

4 Using the JFACC ontology

The JFACC Ontology was designed to be used by a wide range of applications. Currently, we implemented its use by three applications: INSPECT, the Mastermind Objectives Editor, and the Strategy Development Assistant (as introduced in Section 2). These applications use the JFACC Ontology in different ways. The knowledge bases of INSPECT and SDA include the JFACC ontology (or a number of its modules) and add on top of this whatever additional knowledge they need to perform their specific tasks. However, the use of the ontology by the Mastermind Editor is far from trivial. Below, we describe in some detail how the knowledge used by the Mastermind Editor (represented as a grammar) is extracted from the JFACC Ontology, and how we needed to augment the ontology to cope with two problems: the *structural mismatch* between the way the knowledge was formulated in the original ontology and the target representation (a grammar), and the lack of certain kinds of application-specific information in the ontology that required the *layering* of the JFACC ontology to allow for several different views on the same knowledge.

The Mastermind Objectives Editor and its Adaptive Form (see Section 2) uses much of the same knowledge stored in the JFACC Ontology and also used by INSPECT and SDA. This alignment was necessary to allow communication between the applications. The grammar that serves as input to the Adaptive Form contains information about, for example, which verbs can be used and what objects can be used with these verbs, the structure and contents of area, time and resource constraints, etc. Initially, both the ontology and the grammar were generated by hand, and all the usual problems started to happen: it was very hard to keep consistent, knowledge had to be entered twice, etc. Therefore, one of our goals was to develop an automatic translator that used one central source (the ontology) and produced the grammar from it.

We found two main types of problems building this translator. First, the form in which this knowledge is expressed in the grammar is rather different than in the ontology. What is expressed in Loom as concepts, relations and instances is represented in the Mastermind grammars as BNF rules — there is a *structural mismatch*.[5] There is no simple correspondence between elements in the ontology and elements in the grammar, and thus it is necessary to implement a non-trivial mapping between the two knowledge forms. Second, part of the knowledge needed by the grammar (e.g., how certain slots are ordered when mapped to elements in a BNF formula) was not present in the ontology. Conversely, not all of the knowledge had a corresponding grammar form, and should be ignored in the translation. This additional grammar-related knowledge is not an inherent part of the domain, but only guidance about how to use the knowledge. Consequently, we tried to separate it from the "core' ontology by creating an overlay that uses annotations to the ontology elements to add this information for the translation process.

Figure 6 illustrates the structural mismatch problem between the Loom concept `acp-objective` and the BNF rule that results from the mapping/translation process. One type of structural mismatch is global. The specification In Loom, concepts are generally defined by specialization of their parents. In the BNF rules, the parents explicitly list their children. This means that the hierarchical organization of the two formalizations are inverted. Part of the translation process involves expressing the implicit connection between a Loom concept and its children in explicit form in the BNF rule. Furthermore, some of the concepts are expanded in-line in the grammar. Example: In the ontology, the concept `Deploy-Forces-Template` is defined as a child of `ACP-Objective` in which the verb "Deploy" applies to objects of type "Forces". In the grammar, the rule for decomposing an objective lists " 'deploy' forces" as one of its children.

There are also several other types of mismatch. Some types of mismatch relate to the way a specific slot is mapped, others to the way the non-terminal representing that slot in the grammar is going to be produced in

[5] Indeed, one can see this difference as a consequence of the two distinct views underlying the representations in Loom and in BNF.

;; Ontology:

```
(defconcept ACP-Objective
  :is-primitive Time-Range-Bound-Concept
  :roles ((objective-statement :type String)
          (objective-issued-by :type Commander)
          (objective-supports :type ACP-Objective)
          (objective-verb :type Verb)
          (objective-role :type Entity)
          (objective-measures-of-merit :Type Measure-Of-Merit)
          (objective-area-restriction :type Area :max 1)
          (objective-time-restriction :type Time-Restriction :max 1)
          (objective-resource-specification :type Force-Package)))

(defrelation DOB-role
  :is-primitive objective-role)

(defconcept Deploy-Forces-Template
  :roles ((DOB-role :facets ((order-info 1))))
  :is (:and acp-objective
            (:filled-by objective-verb Deploy)     ;; Verb "Deploy"
            (:exactly 1 objective-role)
            (:the DOB-role Forces)))               ;;  applies to objects of type "Forces"
```

Grammar:

```
objective "Objective":
     'attrit' redObject |
     'damage' redObject |
     'deceive' redObject |
     'deny' redAction 'to' redObject |
     'deploy' forces | ...
..........
redObject "Red Object":
     objectTerminals ofRedOwner;

redOwner "Red Owner" :
     redSpecial | redCountryTerminals;

forces "Forces":
     forceType forceNumber;
..........
areaRestriction "Where":
     areaModifier area;

areaModifier "areaModifier":
     'over' | 'in' | 'at';

area "Area" :
     country | conflictSide |
     geographicRegion;
```

FIGURE 6: How the knowledge about the use of the verb deploy is expressed in the JFACC Ontology (Loom, top) and the Mastermind grammar (BNF, bottom). The concept `Deploy-template` expresses how the verb `deploy` is used — that is, which kinds of the roles it can accept (specifically, the direct object, symbolized here by `DOB-role`), and which kinds of objects can fill these roles (specifically, subtypes of the type `forces`).

the mapping process. These types of mismatch are described below.

- Some elements of the Loom ontology have no direct mapping into the Mastermind grammar. For example, the slot `objective-issued-by` has no correspondence in the grammar. This occurs because not all knowledge which is relevant for the ontology is relevant for the grammar.

- The way the mapping is done may depend on the specific slot. Some slots, like `objective-verb`, are not mapped directly into the expansion of the non-terminal objective. Their mapping is delayed to take into account dependencies between its fillers and the fillers of other slots — specifically, `objective-role` in its subtypes, such as `DOB-role`. Other slots (like for example the relation `objective-area-restriction`, are mapped directly into the expansion of the non-terminal `objective`. The basic question is whether or not any subtypes of the concept `acp-objective` may introduce a relationship between the fillers of two roles of the parent concept. If there is no such relationship, we mapped the slots into non-terminal in the grammar rule for objective. Otherwise, we leave the mapping of the slots to be done at a subsequent rule or expansion.

- The way a mapping is constructed to build the grammar rule for a specific non-terminal is also dependent on how the knowledge about this non-terminal was structured in Loom. For example, given a specific non-terminal expressed in the Loom ontology as a concept, the right hand side of the rule corresponding to that concept may be constructed by listing instances of the concept, by listing subconcepts of that concept, or by mapping separately each of the slots (relations) associated with that concept. The same occurs for mapping non-terminals which correspond to relations in Loom.

The basic strategy we adopted to cope with these two problems is to introduce annotations in the JFACC ontology that determine how specific concepts or slots are mapped into BNF rules or elements of these rules. Figure 7 illustrates how these annotations are introduced. It shows an expanded definition of the concept `deploy-template`, as well as some additional elements of the Loom ontology and some other grammar rules that result from their translation. As we can see in the figure, the annotations are introduced in two places. First, we introduce annotations on the slots of specific concept by adding facets to it. We introduced two facets: `order-info` and `fill-by`:

- The `order-info` facet is used to overcome one of the incompleteness problems. It adds the information about the order of the slots when they are mapped into elements of the right side of a BNF rule. Why do we want this order information? We want it because it makes the English language version seem more natural when the grammar is used to create an adaptive form.

  For example, in Figure 7, the `order-info` annotations on `Deploy-object-template` specify a natural order for presentation of roles in the interface. The contents corresponding to the slot `objective-verb` are to appear first (i.e., leftmost) into the grammar rule, the contents corresponding to `DOB-role` are to appear second, etc. This annotation is also used to exclude the use of certain slots, that is, every slot or relation of the concept `deploy-object-template` that has no specific `order-info` information is not mapped into an element of the grammar rule.

- The `fill-by` facet is used to solve the structural mismatch problems for slots. It indicates how a specific slot is supposed to be mapped into an element of a grammar rule. There are three possible fillers for the `filled-by` facet:

  – The filler `instance` indicates that the slots is mapped into the specific instance (filler) of the slot in the definition of the current concept (in this case, the symbol `deploy`)

  – The filler `type` indicates that the slot is mapped into the concept specified in the type restriction for the slot in the definition of the current concept (in this case, the concept `forces`).

  – The filler `non-terminal` indicates that the slot is mapped into a non-terminal with the same name as the slots. The translator later produces a specific rule for the expansion of this non-terminal.

Second, we introduce annotations on concepts to indicate how they should be mapped in the translation process. This is done using the relation `derive-bnf-by`, with four possible fillers:

- `generate-subs` indicates that this concept is mapped into a BNF rule in which the right hand side is an OR clause connecting expansions for each of the immediate subtypes of the concept. For example, `acp-objective` is generated this way: as we see in Figure 6, its corresponding grammar rule contains the expansions for each of the "template" concepts, such as `Deploy-template`.

- `collect-instances` indicates that this concept is mapped into a BNF rule in which the right hand side is an OR clause connecting all known instances of this concept. For example, the concept `country` is mapped into its known instances (`USA`, `Canada`, `Mexico`, etc.). This kind of mapping occurs commonly for concepts (e.g., `country`, `geographic-area`) for which the fillers in a given grammar depend on the context (e.g., which specific region in the world) for which this grammar is being generated, either because no meaningful pre-compiled exhaustive list of fillers can be provided or because such a list would be unmanageably large.

- `collect-concepts` indicates that this concept is mapped into a BNF rule in which the right hand side is an OR clause connecting all known subtypes (subconcepts) of this concept. For example, the right hand side of the rule for `area` lists its subconcepts, such as `geographic-area`, `country-area`, etc.

- `generate-roles` indicates that this concept is mapped into a BNF rule in which the right hand side is an OR clause connecting expansions for each of the slots of the concept. For instance, the concept `time-restriction` is mapped into a rule whose right hand side contains a non-terminal for each of its slots — in this case two slots named `time-restriction-relation` and `time-specification`. Again, each of these slots becomes a non-terminal for which the translator will later produce a specific grammar rule.

```
(defconcept Deploy-object-template
  :roles ((objective-verb :facets ((order-info 1) (fill-by 'instance)))
          (DOB-role :facets ((order-info 2) (fill-by 'type)))
          (objective-area-restriction
               :facets ((order-info 3) (fill-by 'non-terminal)))
          (objective-time-restriction
               :facets ((order-info 4) (fill-by 'non-terminal)))
          (objective-resource-specification
               :facets ((order-info 5) (fill-by 'non-terminal))))
  :is (:and acp-objective
            (:filled-by objective-verb Deploy)
            (:exactly 1 objective-role)
            (:the DOB-role object))
  :annotations (fs-objectives-grammar))

(tellm (derive-bnf-by acp-objective 'generate-subs)
       (derive-bnf-by country 'collect-instances)
       (derive-bnf-by area 'collect-subs)
       (derive-bnf-by time-restriction 'generate-roles)
       ...
       (derive-bnf-by forces 'collect-subs))
```

FIGURE 7: Annotation of the concept `Deploy-objective-template`. Annotations for grammar generation are shown in **bold face**.

It is interesting to notice that in addition to these more semantic problems, there are also more syntactic problems derived from the different conventions for names in Loom and Mastermind. For example, for non-terminals the convention adopted in Loom is to separate different parts of the name with a hyphen (-), while this character is not allowed in a Mastermind grammar. Therefore, all hyphen characters in the names of non-terminals must be eliminated in the translation process (e.g., `time-restriction` is transformed into `timerestriction`). For terminals, the treatment of these characters is different. Because we want to obtain human-readable names, the hyphens are transformed into spaces. In addition, the names are surrounded by single quotes (') — e.g., `military-communications` is transformed into `'military communications'`. For the cases in which a translation using these simple rules of thumb is not adequate, we introduced another type of annotation using the relation `human-name`, in which an arbitrary human name (string) may be specified as a mapping for a given Loom name. This is useful for translations in which case is important (for example `southern-california` into `'Southern California'`) as well as more exceptional cases such as the relation `DOB-role`, which translates into the empty string.

Lessons Learned. The problems we found in trying to use the JFACC ontology in several applications are to some extent similar to those we found in importing an external ontology. This occurs in part because we also had to translate the ontology to a different formalism, and thus found the same mismatch and use bias problems we discussed in section Section 3.1. The gist of the solution we found for this problem (annotating the original ontology constructs to indicate their mapping into the Mastermind grammar) can thus be seen as a possible mechanism to be used in ontology translations. What the annotations allow is for a use to "drive" or "customize" to some extent the translation process to allow for a more precise mapping between source and target ontologies. However, we had an additional problem, which is that the knowledge in the original ontology was incomplete. The solution to this problem was to structure the knowledge in layers, so that the use-specific knowledge is separated as much as possible from the core (and more reusable) knowledge. While we did not intend this to be a general solution to usability problems, we believe these mechanisms can be used more generally to solve similar problems in other ontologies, provided the formalism used in those ontologies allows for layering and the use of annotations as we did here.

Also, the initial choice to use annotations was partly motivated by a need to "get the application out the door." That led us to choose a method that would be quick to implement, but that would get the job done. Upon further reflection, we noticed that the annotations could be divided into domain-specific and method-specific classes. That suggested a more principled approach to their use and specification.

An example of a domain-specific use was providing information regarding the layout of forms. The `order-info` annotation was used to make the English-like sentences in the adaptive input form read more naturally. A more better method of doing this would be to identify the case roles that slots embody and include that information in the ontology directly. For example, including in the ontology the knowledge that `objective-time-restriction` was a time restriction and that `objective-area-restriction` was a place restriction. That would allow the grammar generation routine to use a general rule such as "place restrictions should precede time restrictions" when generating the grammar.

In addition to using a more generally applicable method of marking the slots, this would enhance maintainability by eliminating the need to annotate the slots on every objective (which is what was done in the JFACC knowledge base). This type of solution presupposes that one is able to develop an appropriate model which achieves the purpose of the individual annotations. Since we haven't yet tried this, we are unable to report on whether this would work in this instance.

The method-specific parts of the annotation scheme are harder to separate from the purpose we had in generating a grammar. An example of this is the deciding which concepts can be collapsed into a single form. Relying on the ability of a description logic to discriminate between different concepts based on the value of particular fillers meant that the underlying reasoning system could differentiate between the concept "Deploy troops" and subconcepts "Deploy troops by sea" and "Deploy troops by air." A user interface which had separate forms for entering all three concepts would be unnecessarily tedious for a user. Although one might be able to construct a tool that would identify potential candidates for such treatment, this seems to be an area where the role of an interface designer is needed to determine exactly how to do the presentation.

5 Conclusions

Building the JFACC Ontology was an extremely interesting and useful enterprise. Even though we are still expanding and experimenting with the JFACC Ontology, we have already obtained in some part all the benefits we envisioned in the beginning of the process — namely, to facilitate inter-operation and communication between systems to promote sharing of knowledge between systems, to integrate our knowledge acquisition, and to offer a repository for general knowledge about air campaign planning that could be used in several applications. We are now focusing on using the ontology to bridge the gap between knowledge-based and "traditional" systems, substituting of superseding the usual database or object-oriented schema with an ontology, which offers a semantically richer model of the domain.

However, we found that actually building, using and reusing ontologies requires a lot of manual work and a lot of difficult modeling decisions, even when using some of the best and most mature knowledge representation formalisms currently available. In this paper, we described some important, general problems, and how we solved them in the JFACC ontology. First, differences in modeling styles and underlying views or upper models in different formalisms can cause problems in translating and merging. The intended use of ontologies (expressed in the inference bias) must be considered in the translation process to make the translation products more useful. Second, these problems appear even when merging several ontologies originally constructed in the same formalism. A specific formalism restricts some modeling choices, but there are still a lot of options available for the ontological engineer in how to use the constructs available. As a consequence, different people can construct quite different representations of the same knowledge. What we want to do is to be able to distinguish what is behind these choices, and what consequences they entail, so that the merging can result in a richer ontology that combines the best of both initial ontologies. Third, trying to use an ontology in several applications can be very trivial, but it can also require a translation process in many respect similar to the one used in importing an ontology, and equally difficult. We found it is difficult to structure an ontology to be used in different applications, and an interesting solution is to maintain a core representation and layer additional information necessary to make specific uses of the knowledge in this core.

Finally, while we were able to find adequate solutions for all the technical problems we found in developing the JFACC ontology, general solutions to these problems are difficult to come by. We believe, however, that practical experiences in building, using, and reusing ontologies are extremely important in directing research in ontological engineering. One research direction pointed by our experience is to focus on the *processes* performed in ontological engineering. In addition to develop techniques and tools to support ontological engineering, it would be best to aim for a *ontological engineering methodology* that can provide support and guidance to ontological engineers.

It is likely that an ontology used in a particular knowledge-based system will contain items that are idiosyncratic to the use being made of the ontology. Sometimes this will be because of pragmatic pressures that preclude the designers from stepping back and considering the larger picture. Sometimes this will be the result of a lack of a theory of how to automate a particular aspect of the application. As noted above, some of the information contained in the annotations could be added to the ontology in a more principled way. One important lesson is to find time to review ontologies after application delivery and try to identify such opportunities for improving the ontology. This will lead to better maintainability as well as to increased potential for reuse.

In summary, our experience has shown that reusability is both desirable and possible. Having a well-structured ontology of a domain provides a basis on which to build, and thus helps enormously to develop new systems on that domain. For example, the SDA was created after the JFACC Ontology was already in place, and its construction demanded much less knowledge acquisition and modeling than previous applications did. In fact, not having to worry about the basic modeling problems enabled us to spend more time on the representation issues related to SDA's problem solving methods, saving considerable time and allowing us to produce a better system.

Acknowledgments

This paper reuses significant parts of the article "Building, Using and Reusing an Ontology of Air Campaign Planning", published in IEEE Intelligent Systems, Volume 14, Number 1, pages 27–36. The work reported here relates to ongoing research sponsored by the Defense Advanced Research Projects Agency (DARPA) under Ft. Huachuca Contract DABT63-95-C-0059 and Air Force Research Laboratory Agreements F30602-97-C-0118 and F30602-97-C-0068. The views and conclusions contained in this article are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government. Many thanks to Yolanda Gil and the anonymous reviewers of the IEEE article for their comments on earlier versions of this material.

References

Allen, J. (1984). Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154.

Brachman, R. J., McGuinness, D. L., Patel-Schneider, P. F., Resnick, L. A., & Borgida, A. (1991). Living with Classic: When and how to use a KL-ONE-like language. In Sowa, J. F., (Ed.), *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 401–456. San Mateo, CA, Morgan Kaufmann.

Frank, M. & Szekely, P. (1998). Adaptive forms: An interaction paradigm for entering structured data. In *Proceedings of the ACM International Conference on Intelligent User Interfaces*, pages 153–160, San Francisco, California. ACM Press.

Hama, T., Hori, M., & Nakamura, Y. (1992). Task-specific language constructs for describing constraints in job assignment problems. Research Report RT0084, IBM.

MacGregor, R. (1991). Inside the LOOM classifier. *SIGART Bulletin*, 2(3):70–76.

Patel-Schneider, P., Owsnicki-Klewe, B., Kobsa, A., Guarino, N., MacGregor, R., Mark, W., MacGuiness, D., Nebel, B., Schmiedel, A., & Yen, J. (1990). Term subsumption languages in knowledge representation. *AI Magazine*, 11(2):16–23.

Swartout, W., Patil, R., Knight, K., & Russ, T. (1996). Toward distributed use of large-scale ontologies. In Gaines, B. & Musen, M., (Eds.), *Proceedings of the Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop*.

Valente, A., Gil, Y., & Swartout, W. (1996). INSPECT: An intelligent system for air campaign plan evaluation based on EXPECT. Technical report, USC – Information Sciences Institute.

Woods, W. A. (1991). Understanding subsumption and taxonomy: A framework for progress. In Sowa, J. F., (Ed.), *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 45–94. San Mateo, CA, Morgan Kaufmann.