Using Prior Knowledge: Problems and Solutions

Vinay K. Chaudhri, Mark E. Stickel, Jerome F. Thomere, Richard J. Waldinger

{chaudhri, stickel, thomere, waldinger}@ai.sri.com

Artificial Intelligence Center

SRI International

333 Ravenswood Ave, Menlo Park, CA 94025

Abstract

Encoding knowledge is time consuming and expensive. A possible solution to reduce the cost of developing a new knowledge base (KB) is to reuse existing knowledge. Previous work addressing this problem has focused on standards for representing, exchanging, and accessing knowledge (Genesereth and Fikes 1992), (Chaudhri et al. 1998), and on creating large repositories of knowledge (Lenat and Guha 1990). Results on the level of reuse achievable have been reported (Cohen et al. 1999). In this paper, we focus on the process of reuse and report a case study on constructing a KB by reusing existing knowledge. The reuse process involved the following steps*: translation, comprehension, slicing, reformulation,* and *merging*. We discuss technical problems encountered at each of these steps and explain how we solved them.

Introduction

A possible approach to reduce the cost of developing a new knowledge base (KB) is to amortize the high cost of encoding knowledge across multiple projects. To enable the reuse of knowledge across multiple projects, several complimentary approaches have been attempted. Standards for representing, exchanging, and accessing knowledge have been developed (Genesereth and Fikes 1992),(Chaudhri et al. 1998). Large repositories of knowledge have been constructed to serve as the starting point for new KB development (Lenat and Guha 1990).

This paper is about the process of knowledge reuse that builds upon the earlier work by starting from an existing repository of knowledge, and by using the standards for representing and exchanging knowledge in the development process. This process is based on an actual KB construction project, and involved the following steps*: comprehension*, *translation*, *slicing*, *reformulation*, and *merging*. We discuss technical problems encountered at each step and how we solved them.

Experimental Setup

The focus for the KB development was defined by the *Crisis Management Challenge Problem* (CMCP) (IET et al. December 1997),(Cohen et al. 1998). The CMCP defined a collection of test questions of interest to an analyst dealing with an international crisis. The objective of the KB development task was to assist the analyst in answering those questions.

The questions were specified using a question grammar. The grammar consisted of a set of parameterized questions or PQs, each of which had a large number of instantiations. An example question is, ‘‘ What {risks, rewards} would <InternationalAgent> face/expect in <InternationalActionType>?’’. An example instantiation is, ``What risks can Iran expect in attacking targets in Saudi Arabia?’’. The KB development included several test cycles with the test phase based on questions derived from the question grammar, but not previously seen by the system developers. A major test cycle was conducted at the end of the year with several small-scale tests in between. The results reported here are based on the work conducted over a period of two years.

Teams led by Teknowledge and SAIC developed the systems for answering the questions. SRI was a part of the SAIC team, and this paper concerns primarily the work conducted at SRI.

The KB was developed using the KIF syntax (Genesereth and Fikes 1992), augmented with the standard relation names derived from the OKBC knowledge model (Chaudhri et al. 1998). SNARK, **S**RI’s **N**ew **A**utomated **R**easoning **T**oolkit, a first-order theorem prover, was used as an inference tool to answer the questions (Stickel et al. 1994). The details of the KB content and the inferences performed are available elsewhere (Chaudhri et al. 2000).

The KB development started from the HPKB upper ontology (HPKB-UL) that contains roughly 3000 most general terms derived from the Cyc KB (Lenat 1997). Four members of the SAIC team -- SRI, Knowledge Systems Laboratory (KSL), Stanford, Formal Reasoning Group (FRG), Stanford, and Northwestern University (NWU) -- addressed portions of the CMCP and developed somewhat independent KBs. After the first year of the project, KSL Stanford merged the KBs from SRI, KSL Stanford, and FRG Stanford, and the resulting KB was the starting point for the development for the second year.

Problems in Reusing Prior Knowledge

The reuse process reported here is a result of the practical needs of the project. A similar process has been adopted by others (Valente et al. 1999).

Translation

Translation is the process of taking the well-formed sentences in one representation language as input and producing the equivalent well-formed sentences of another representation language as output. Our translation step was limited to syntactic transformations and constant renaming. Any further processing, for example, choosing alternative syntactic forms in the target representation language or dealing with representation differences is handled as a separate step in the reuse process. It is possible that for some sentences in the input representation language, there is no equivalent sentence in the output language. Such a situation did not arise in our application.

The HPKB-UL was available in the MELD format (a language used by Cycorp) and was not directly readable by our system. In conjunction with KSL Stanford, we developed a translator to load the HPKB-UL into any OKBC-compliant server. Since the HPKB-UL contains mostly structural information (Pease et al. 2000), this translator handles structural information in the KB. The structural information includes classes, functions, relations, class-subclass relationships, and facets. While doing this translation, we had to define equivalence between relation names in the HPKB-UL and the OKBC knowledge model. For example, the relation *#$genls* in the HPKB-UL is equivalent to the relation *subclass-of* in the OKBC knowledge model. The translation process benefited significantly by the existence of the OKBC knowledge model into which many of the representation constructs from the HPKB-UL could be mapped. We also converted the case-sensitive names from the HPKB-UL to case-insensitive names. For example, the constant name *#$performedBy* from the HPKB-UL was mapped to *performed-by*. The syntactic translation was a low-effort engineering task, and accounted for a small fraction of the KB development time.

Comprehension

Before a knowledge engineer reuses an ontology, its contents and organization must be understood. Two techniques enabled ontology comprehension. First, we used our graphical visualization tool, GKB-Editor (Paley et al. 1997), to explore the HPKB-UL. The taxonomy and the relationship browsers of the GKB-Editor were instrumental in helping us understand the interrelationships between classes and predicates of the HPKB-UL. During the KB development process, the GKB-Editor's browsing capabilities were extensively used to search for necessary concepts and to identify the location in the taxonomy for a new concept. Second, the extensive documentation accompanying the Integrated Knowledge Base Environment (IKB) clarified many design decisions and presented the KB content in many different ways. (IKB is a portion of the Cyc KB that was distributed to the participants of the project.) For example, the IKB documentation included a glossary of terms organized by domains such as time, space, geography, communication actions, etc. The IKB also included a search facility that linked the English words in Wordnet to the corresponding concept names in the KB.

Slicing

Slicing involves selecting a portion of an input ontology for use in a new application. Using all of the input ontology may not be desirable for the following reasons. First, all of the input ontology may not be needed for a new application. Second, importing all of it may make the resulting KB unnecessarily complex. Third, there may be aspects of the input ontology that the target inference tool is unable to handle, and that must be removed. Finally, some of the representation decisions made in the input ontology may not be acceptable to the target application.

Two technical problems must be solved for slicing. First, we need to decide what portions of the input ontology we need to slice. We call the portion of the input ontology that needs to be extracted the *seed*. Second, we need a computational procedure that extracts out just the right amount of terms from the input ontology.

More formally, an ontology contains a set O of sentences of the following form.

- *(class X)*, where $X \in C$, and C is a set of classes
- *(relation X)*, where $X \in R$, and R is a set of relations
- *(function X)*, where $X \in F$, and F is a set of functions
- *(individual X)*, where $X \in I$, and I is a set of individuals
- *(subclass-of X Y)*, where $X,Y \in C$
- *(instance-of X Y)*, where $X \in C \cup I, Y \in C$
- *(arity X N)*, where $X \in R \cup F$, and N is a positive integer
- *(nth-domain X N Y)*, where $X \in R \cup F$, N is a positive integer, and $Y \in C$
- *(range X Y)*, where $X \in F$, and $Y \in C$
- *(nth-domain-subclass-of X N Y)*, where $X \in R \cup F$, N is a positive integer less than the arity of R, and $Y \in C$
- *(range-subclass-of X Y)*, where $X \in F$, and $Y \in C$
- *(r X V)*, where $r \in R$
- *(template-slot-value X Y V)*, where $X \in C$, $Y \in R$, and *(arity R 2)* is in O

The relation symbols *class*, *individual*, *subclass-of*, *instance-of*, and *template-slot-value* have meanings as defined in the OKBC specification (Chaudhri et al. 1998). The sentence *(class X)* means that X is a set or a unary relation. The sentence *(individual X)* means that X is not a set. The sentence *(subclass-of X Y)* means that the class X is a subset of class Y. The sentence *(instance-of X Y)* means that individual X is a member of the set Y. The sentence *(relation X)* means that X is a relation. The sentence *(function X)* means that X is a function. Every class is a unary relation, therefore, $C \subseteq R$. The relation symbols *range*, and *range-subclass-of* specify the type restriction on the value of a function. If *(range F C)*, the value returned by F must be an instance of the class C. If *(range-subclass-*

of F C), then the value returned by *F* must be a sub class of the class *C*. The relation *arity* specifies the number of arguments of a function or relation. If *(arity R 3)*, the relation *R* can have exactly three arguments. The relation symbols *nth-domain* and *nth-domain-subclass-of* specify the type restriction on the arguments of the functions and relations. If *(nth-domain R i C)*, then the *ith* argument of the relation *R* must be an instance of the class *C*. Similarly, if *(nth-domain-subclass-of R i C)*, the *ith* argument of the relation *R* must be a subclass of *C*.

The seed *S* is a set containing sentences of the form *(class X)*, *(relation X)*, *(function X)*, and *(individual X)*, where $X \in C \cup R \cup F \cup I$. Based on the knowledge of the target application, we were able to identify *S*. At the beginning of a project, all needed terms may not be included in *S* in the initial estimate of *S*. As the KB evolves, the seed can be revised. In practice, it was sufficient to recompute the slice once every six months.

The slice *L* is a subset of *O*. We would like to compute *L* in a way that all the useful information from the source ontology is incorporated into the KB being developed. We call a slice *maximal with respect to S* if any inferences involving *S* that can be performed using *O* can be performed using *L*. We call a slice *L* that is maximal with respect to S as *minimal with respect to S* if there is no $L' \subset L$ that is maximal with respect to S.

A trivial way to compute *L* is to simply return *S*. In general, *S* is not a maximal slice of *O* with respect to *S*. Let us define an algorithm to compute the maximal slice of *O*, with respect to *S*.

Algorithm *MaximalSlice*
Input: Input ontology *O*, and seed *S*
Output: *L*, a slice of *O*, with respect to *S*

1. Let $S' = \{X \mid$ *(class X)* $\in S$, or *(relation X)* $\in S$, or *(function X)* $\in S$, or *(individual X)* $\in S\}$.
2. Set $L = S$.
3. For every $X \in S'$, if *(nth-domain X N Y)* $\in O$, add *Y* to *S'*, and add *(class Y)* and *(nth-domain X N Y)* to *L*.
4. For every $X \in S'$, if *(nth-domain-subclass-of X N Y)* $\in O$, add *Y* to *S'*, and add *(nth-domain-subclass-of X N Y)* and *(class Y)* to *L*.
5. For every $X \in S'$, if $X \in F$, and if *(range X Y)* $\in O$, add *Y* to *S'*, and add *(range X Y)* and *(class Y)* to *L*.
6. For every $X \in S'$, if $X \in F$, and if *(range-subclass-of X Y)* $\in O$, add *Y* to *S'*, and add *(range-subclass-of X Y)* and *(class Y)* to *L*.
7. For every $X \in S'$, if $X \in C$, and if *(subclass-of X Y)* $\in O$, add *Y* to *S'*, and add *(subclass-of X Y)* and *(class Y)* to *L*.
8. For every $X \in S'$, if *(instance-of X Y)* $\in O$, add *Y* to *S'*, and add *(instance-of X Y)* and *(class Y)* to *L*.
9. For every $X \in S'$, if *(r X V)* $\in O$, add *(r X V)* to *L*. If *(class V)* $\in O$, add *(class V)* to *L*, and *V* to *S'*. If *(individual V)* $\in O$, add *(individual V)* to *L*, and *V* to *S'*.
10. For every $X \in S'$, if *(template-slot-value X r V)* $\in O$, add *(template-slot-value r X V)* to *L*. If *(class V)* $\in O$, add *(class V)* to *L*, and *V* to *S'*. If *(individual V)* $\in O$, add *(individual V)* to *L*, and *V* to *S'*.
11. Repeat steps 7 through 10 until *L* does not change.
12. Return *L*.

The algorithm *MaximalSlice* works by first determining all the relevant classes, and then computing their upward closure in the graph of taxonomic relationships.

Theorem 1: The algorithm *MaximalSlice* produces a slice *L* of *O*, which is maximal with respect to *S*.
Theorem 2: The algorithm MaximalSlice is polynomial in the size of *C*.

It is possible to produce a smaller slice if one has additional knowledge about the sorts of axioms that are of interest for the target application. For example, suppose *X* is a *subclass-of Y*, and *Y* is a *subclass-of Z*, and that *X* is in the seed, but *Y* and *Z* are not. If there is no *(template-slot-value Y r V)* sentence in *O*, and if it is not used in any *nth-domain*, *nth-domain-subclass-of*, *range*, and *range-subclass-of* sentence, it may be dropped from the closure by asserting *X* as a *subclass-of Z*. This does not change any inferences of interest that can be performed about *X*. An example definition of *interestingness* follows.

Definition 1. A class *X* is of interest with respect to a seed S if one of the following holds.

- X is the root of the class-subclass graph
- The sentence *(nth-domain Y N X)* is in *O*, and *Y* is in *S*
- The sentence *(nth-domain-subclass-of Y N X)* is in *O*, and *Y* is in *S*
- The sentence *(range Y X)* is in *O*, and *Y* is in *S*
- *(range-subclass-of Y X)* is in *O*, and *Y* is in *S*
- *(template-slot-value X r V)* is in *O*

Using this definition, one can compute an interesting superclass of a class *X* as follows. For every *(subclass-of X Y)* sentence in *O*, the superclass *Y* is interesting if *Y* is of interest. If *Y* is not of interest check to see if *Z* is of interest, where *(subclass-of Y Z)* is in *O*. If *Z* is of interest, *Z* is an interesting superclass of *X*. For a rooted and connected taxonomy, this process is guaranteed to terminate. An interesting type of a class or an individual may be computed analogously.

Algorithm *MinimalMaximalSlice*
Input: Input ontology *O*, seed *S*, and slice *L* produced by *MaximalSlice*
Output: *L*, a slice of *O*, with respect to *S*

1. Steps 1 through 6 are the same as the algorithm *MaximalSlice.*
7. For every *(subclass-of X Y)* in *L,* compute interesting parent Z as in Definition 1, add *(subclass-of X Z)* and *(class Z)* to *L,* and *Z* to *S'*.
8. For every *(instance-of X Y)* in *L,* compute interesting parent Z as in Definition 1, add *(instance-of X Z)* and *(class Z)* to *L,* and *Z* to *S'*.
9. Steps 9 through 12 are the same as in the algorithm *MaximalSlice*.

Theorem 3: The algorithm *MinimalMaximalSlice* produces a slice *L* of *O,* which is minimally maximal with respect to *S* assuming the interestingness of a class as defined in Definition 1.

During the first year of the project, we used the trivial slice of the HPKB-UL, that is, we just used the constant names and documentation strings. During the second year, we used the *MaximalSlice.* The motivation for *MinimalMaximalSlice* was to argue that while reusing an ontology, it is not necessary to agree with everything in the source ontology, especially those terms and representations that can be *sliced* away. The terms that can be sliced away are like the binary code in a compiled program that never needs to be exposed to the knowledge engineer.

To consider the generality of the results of this section, let us compare the knowledge model considered here with some other well-known systems. The relation symbols taken from the OKBC knowledge model, *class, individual, subclass-of, template-slot-value,* and *instance-of,* or their equivalents are supported by a wide range of knowledge representation systems such as LOOM (MacGregor 1991), Classic (Borgida et al. 1989), and Ontolingua (Farquhar et al. 1997). Since slots are binary relations, the *nth-domain* restriction, for *n=1,* is equivalent to the *domain* restriction on a slot, and for *n=2,* is equivalent to the value type restriction. The domain and value type restrictions are commonly supported. Higher arity relations, functions, *nth-domain-subclass-of,* and *range-subclass-of* are not supported in the OKBC knowledge model and the description logic systems such as Classic. The equivalent relations, for example, *arg1Genls, resultGenls,* are in the HPKB-UL, and are used extensively in the Cyc KB. Numeric and cardinality restrictions on slot values are supported in the OKBC knowledge model, LOOM, and CLASSIC, but are not considered here. It is, straightforward to extend the slicing algorithms to include numeric and cardinality constraints. Supporting constructs such as *disjoint-with, same-values, not-same-values, etc.* remains open for future work.

Reformulation

Reformulation is the process of taking an input theory and transforming its representation. Reformulation is synonymous with morphing (Chalupsky 2000). A common reason for reformulation is that in the target system the reformulated theory may be more efficient to reason with than the original theory. We reformulated the HPKB-UL to convert every *Functional Predicate* into functions. We explain this reformulation in more detail.

HPKB-UL represents the functional relationships as predicates. For example, even though *mother* is a function, it is represented in the HPKB-UL as a relation. Such predicates are instances of the class *Functional Predicate*. There are two differences between using functions and relations for representing functional relationships.

First, functions are a more compact way to state that the relationship between two objects is single-valued and that when a function is applied to an object (or a set of objects), the value indeed exists (Bundy 1977). To assert the same information using relations, one needs to also specify that the cardinality of the relation is 1. Thus, when we represent *mother* as a function, we are guaranteed that every individual has one and only one mother. When we represent *mother* by a relation, we do not get any such guarantees unless we also assert the cardinality constraint.

Second, using functions, the paramodulation rule of inference can be applied. The benefits of using functions are enhanced while using equality reasoning. SNARK, like most theorem provers, uses the paramodulation rule of inference while reasoning with equality. The paramodulation rule, given an assertion such as *a = b*, allows us to simplify a formula such as *(R a)* to *(R b).* If we use functions instead of relations, it introduces equality in the KB. For example, *(mother sue john)* is replaced by *sue = mother (john).* SNARK is then able to use the paramodulation rule of inference for reasoning with such formulas. The paramodulation rules of inference can sometimes lead to faster and shorter proofs. But in other cases, the search space can become larger.

The implementation of this reformulation was straightforward because we were dealing with only structural information. Implementing this reformulation for general axioms remains open for future research.

Merging

Merging involves ensuring that the merged KBs use the same constant names when they mean the same thing, and that they represent the same information identically (McGuinness et al. 2000). Merging assumes the existence of two independently developed KBs that need to be combined to produce a new KB. The merging task for our KB development effort was primarily performed by KSL Stanford, and the details may be found elsewhere (McGuinness et al. 2000). The merging step was made easier by the fact that the KBs were developed using a standard syntax: ANSI KIF extended by standard relation names from the OKBC knowledge model. We, however, needed to invest some effort in resolving the conflicts

arising from the merge. We give here one example of the representational difference that arose during the merge process.

The merge process revealed that the KBs developed by SRI and KSL Stanford both represented situations in which one agent supports or opposes another. To represent an action, in which one agent supports another action, say a terrorist attack, there are two alternatives. First, one can define a class *supporting-terrorist-attack* and create an individual member of this class to represent an instance of a supporting action. Second, one can define a slot called *supports* on the class *action*, which can take an instance of *terrorist-attack* as a value. If the KBs to be merged use these different representations for supporting actions, then the merge phase should either use one representation over the other or add axioms defining equivalence between the two representations. In the current merge, a meeting was organized between KSL and SRI to resolve this difference, and the solution that defines the slot *supports* on class *action* was adopted.

In a KB development project, the merging effort can be reduced by proactive means. For example, if the development process is driven by reuse, a global catalogue for constant names is maintained, and a style guide is followed for inventing new names, the problem of the same terms meaning the same thing can be reduced.

Experimental Results on Knowledge Reuse

We have three objectives in presenting the experimental results: to characterize the KB development process, to give specific examples of knowledge reuse to highlight that the results are based on some nontrivial examples of reuse, and finally to show the level of reuse achieved.

An overview of the KB development process is shown in Figure 1. The KB development started from the HPKB-UL. We took a slice of the HPKB-UL that was extended to create the KB CMCP-98. Two other KBs were developed independently at Stanford – the *Supports* KB was developed by KSL, and the *Capability* KB was developed by the Formal Reasoning Group. At the end of the first year, these KBs were merged to produce a new KB called the *SAIC merged ontology (MO)*. The development for the second year of the project started by taking a slice of the SAIC MO, which was extended to produce CMCP-99, the KB for the second year.

While interpreting the empirical results presented next it is helpful to understand the relationship between the test questions between the two years. During the second year, many of the questions were repeated from the first year, and several new questions were introduced. In terms of the domain content, the test questions during the first year involved organizations, agents, geographical regions, products and economic measures. During the second year, several different kinds of actions, interests and historical case descriptions were introduced. Even though there was

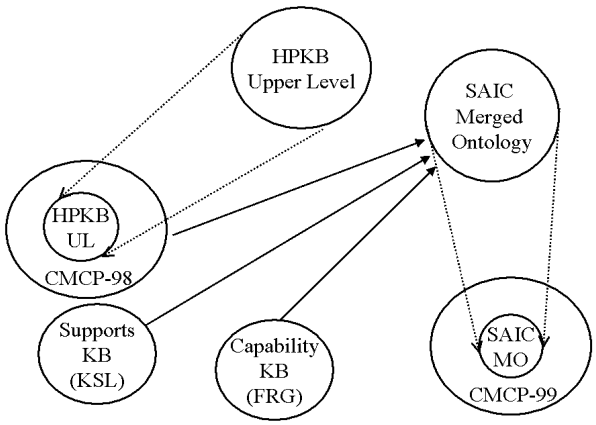

Figure 1. An overview of the development process

a substantial overlap in the content, the test problems differed significantly across the two years.

The HPKB-UL had about 16,434 axioms, and our initial slice of it contained 446 axioms. The CMCP-98 KB had 5943 axioms. The SAIC merged ontology contained 21,223 axioms. (We have excluded many ground facts from this count.) The slice of the merged ontology that was used for the development during the second year contained 5360 axioms. The CMCP-99 KB contained 22,902 axioms. The slice of HPKB-UL was recomputed three times over a period of two years. The final slice of HPKB-UL contained 2544 axioms.

To highlight the nature of reuse, we consider two representations. One of the questions answered by our system was:

Has post-Shah Iran launched ballistic missiles in wartime?

The upper ontology had a class representing *weapons*. We extended it by creating *ballistic-missiles* as a new subclass of the class representing weapons that already existed. The verb "launch" in the questions was mapped to the action *attack*, which was a subclass of an already existing class *hostile-social-action*. The class of actions in the HPKB-UL had several slots. The slot *performed-by* on an action specified the doer of the action, and *device-used* specified the tool that was used in performing that action. Finally, we specified the temporal extent of the question by defining a constant representing the time *reign-of-shah* and then using the temporal comparison *starts-after-ending-of*, from the HPKB-UL. The resulting formalization follows:

```
(and
  (attack ?act)
  (performed-by ?act Iran)
  (device-used ?act ballistic-missile)
  (starts-after-ending-of ?act reign-of-shah))
```

The formalization of this question reuses primitives for representing temporal knowledge and slots on actions from the HPKB-UL. As another example, consider the question:

What risks can Iran expect in sponsoring a terrorist attack in Saudi Arabia?

To answer questions of this type, one can use a cause effect model involving five predicates*: enables, causes, prevents, maleficiary,* and *beneficiary*. The predicates *enables, causes,* and *prevents* are based on the common sense language for reasoning about causation and rational action previously developed elsewhere (Ortiz 1999). The relation *causes* is used to represent the effects that are definitely caused by an action, *enables* to represent those actions that are made possible by an action, and *prevents* to represent actions that are prevented by an action. The relation *maleficiary* to relate an action to an agent who is harmed by that action, and *beneficiary* to relate an action to an agent who is benefited by that action. Thus, if an *?agent* performs an *?action1* that *causes* another action *?action2*, and the performer of *?action1* is *maleficiary* to *?action2*, then *?action2* is a risk in doing *?action1*. Similarly, if an *?agent* performs an *?action1* that *prevents* another action *?action2*, and the performer of *?action1* is the *beneficiary* of the *?action2*, then *?action2* is a risk in doing *?action1*. The HPKB-UL contained a predicate *cause-event-event* that was equivalent to *causes*. Adding the predicates *prevents*, and *enables* extended the HPKB-UL. The predicates *beneficiary* and *maleficiary* were reused directly from the HPKB-UL.

The empirical results that we present here are based on a reuse metric that was proposed earlier (Cohen et al. 1999). The metric can be computed for either axioms or constant names. Suppose a knowledge engineering task requires n constants and k of those can be reused from an existing KB; then, k/n measures the extent of reuse of the KB. The reuse of axioms can be computed analogously. The results are shown in Table 1.

The table first reports the constant reuse in constructing the KB CMCP-99, that is, the KB at the end of the project. We computed the reuse with respect to the HPKB-UL, CMCP-98, and the pre-evaluation KB, that is the KB that existed just before the final evaluation at the end of the second year began. Constants include any class, relation, function, or individual in the KB. The structural statements include atomic statements, which use the relations *subclass-of, instance-of, nth-domain, nth-domain-subclass-of, range-subclass-of, arity, subrelation-of,* and *disjoint-with.* The first numeric entry of 0.22 means that for encoding all the structural statements in CMCP-99, 22% of necessary constant symbols were already in the HPKB-UL. Similarly, 29% of all the necessary constants for encoding the statements containing implications were reused from the HPKB-UL.

The table next reports the level of reuse for axioms that were actually used in answering the questions. The reuse for non-ground statements is not reported, because the number of such statements in the KB was low. The reuse of the HPKB-UL in actually answering was somewhat lower than in constructing the KB. There were no implications in the HPKB-UL, and therefore, it was not meaningful to report the reuse of implications from it. The reuse of the pre-evaluation KB in answering the questions was higher than in constructing the new KB.

Table 1. Gross k/n for axioms in the KB and the axioms actually used to answer questions

With respect to KB	*Constant Reuse in Constructing the*		
	Structural	*Implications*	*Non-Implications*
HPKB-UL	0.22	0.29	0.13
Y1 KB	0.18	0.33	0.19
Pre-eval KB	0.49	0.78	0.39
	Axiom Reuse in Answering the Questions		
	Constants	*Structural*	*Implications*
HPKB-UL	0.17	0.12	N/A
Y1 KB	0.21	0.09	0.05
Pre-eval KB	0.67	0.76	0.43

The absolute value of these numbers is of less interest than the observation that the reuse of prior knowledge is indeed possible. The reuse from the HPKB-UL is especially interesting because it was not designed with the current application in mind.

Scope for Future Work

In this project, we reused only constants, and structural statements from the HPKB-UL, and did not reuse any statements containing implications and non-ground facts that were available in IKB. A hypothesis for future work would be that since our KB shares the upper structure with IKB, it would enable us to share the implications and non-ground facts from the HPKB-UL with greater ease. Exploring this hypothesis would also require us to extend our work on slicing and reformulation. The slicing techniques would need to be extended to slice out the relevant rules. The representation differences are likely to have a greater impact on rules than they had on the class-subclass structure; therefore, new reformulation techniques will need to be developed.

Apart from the technical issues associated with reuse, there are human issues. Knowledge engineers prefer their

own representations to reusing someone else's. In many cases, using different representation does not necessarily contribute to the overall system and makes it difficult to scale the scope of a KB. We reuse other people's software routinely as long as it is well packaged, has a clear functionality, and adds value to our work. Doing the same for knowledge components has been a dream for the community for a long time, and remains an open challenge for the knowledge reuse technology.

Summary

We presented a case study in reusing prior knowledge. The reuse of prior knowledge was done in the following steps: translation, comprehension, slicing, reformulation, and merging. The *translation* tools were developed to convert a subset of MELD into KIF augmented with standard relation names from the OKBC knowledge model. The *comprehension* phase used graphical visualization tools and the KB documentation. The *slicing* techniques were developed to extract a portion of the existing KB to be incorporated into the new KB. The representation was *reformulated* so that the reused KB can be efficiently reasoned with. The *merging* phase involved human intervention to resolve the representation differences. We presented several concrete examples of reuse for an application in the crisis management domain, and empirically argued that KB construction by reuse from prior knowledge is indeed feasible. These results present an advancement of the state of the art of KB construction methods that start a new development from scratch.

Acknowledgements

This work was supported by DARPA's High Performance Knowledge Bases Project. We thank the co-developers of the CMCP KBs at SAIC, KSL Stanford, FRG Stanford, and NWU. We thank the developers of the CMCP that defined the context of the KB development for the project. We thank Cycorp for making the HPKB-UL available for use in the project. We also thank the SRI staff members Charlie Ortiz, and Nina Zumel for their technical contributions to this work.

References

Borgida, A., Brachman, R. J., McGuinness, D. L., and Resnick, L. A. (1989). "CLASSIC: A Structural Data Model for Objects." Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, OR, 58-67.

Bundy, A. (1977). "Exploiting the properties of functions to control search." *D. A. I. Report No. 45*, University of Edinburgh.

Chalupsky, H. (2000). "Ontomorph: A System for Translation of Symbolic Knowledge." *Seventh International Conference on Knowledge Reprewentation and Reasoning*, Breckenridge, CO.

Chaudhri, V. K., Farquhar, A., Fikes, R., Karp, P. D., and Rice, J. P. (1998). "OKBC: A Programmatic Foundation for Knowledge Base Interoperability." Proceedings of the AAAI-98, Madison, WI.

Chaudhri, V. K., Lowrance, J. D., Stickel, M. E., Thomere, J. F., and Waldinger, R. J. (2000). "Ontology Construction Toolkit.", SRI International, Menlo Park, CA.

Cohen, P., Chaudhri, V. K., Pease, A., and Schrag, B. (1999). "Does Prior Knowledge Facilitate the Development of Knowledge-based Systems." Proceedings of the AAAI-99, 221-226.

Cohen, P., Schrag, R., Jones, E., Pease, A., Lin, A., Starr, B., Gunning, D., and Burke, M. (1998). "The DARPA High-Performance Knowledge Bases Project." *AI Magazine*, 19(4), 25-49.

Farquhar, A., Fikes, R., and Rice, J. P. (1997). "A Collaborative Tool for Ontology Construction." *International Journal of Human Computer Studies*, 46, 707--727.

Genesereth, M. R., and Fikes, R. E. (1992). "Knowledge Interchange Format, Version 3.0 Reference Manual." (Logic-92-1).

IET, Alphatech, Pacific Sierra Research, and Cohen, P. (December 1997). "HPKB year 1 end-to-end challenge problem specification, version 1.1."

Lenat, D. B. (1997). "Cyc Public Ontology." *http://www.cyc.com/cyc-2-1/index.html*.

Lenat, D. B., and Guha, R. V. (1990). *Building Large Knowledge-Based Systems*, Addison Wesley, Reading, MA.

MacGregor, R. (1991). "The evolving technology of classification-based knowledge representation systems." Principles of semantic networks, J. Sowa, ed., 385--400.

McGuinness, D., Fikes, R., Rice, J., and Wider, S. (2000). "An Environment for Merging and Testing Large Ontologies." *Seventh International Conference on Knowledge Representation and Reasoning*, Breckenridge, CO.

Ortiz, C. L. (1999). "A Commonsense Language for Reasoning about Causation and Rational Action." *Artificial Intelligence Journal*, 111(2), 73-130.

Paley, S. M., Lowrance, J. D., and Karp, P. D. (1997). "A Generic Knowledge Base Browser and Editor." Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence.

Pease, A., Chaudhri, V. K., Farquhar, A., and Lehman, F. (2000). "Practical Knowledge Representation and DARPA's High Performance Knowledge Bases Project." *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning*, Brekcenridge, Colorado.

Stickel, M., Waldinger, R., Lowry, M., Pressburger, T., and Underwood, I. (1994). "Deductive Composition of Astronomical Software from Subroutine Libraries."

Proceedings of the Twelfth International Conference on Automated Deduction (CADE-12), 341--355.

Valente, A., Russ, T., MacGregor, R., and Swartout, W. (1999). "Building and (Re)Using an Ontology of Air Campaign Planning." *IEEE Intelligent Systems*, 14(1), 27-36.