An Algorithm for Merging and Aligning Ontologies: Automation and Tool Support

Natalya Fridman Noy and Mark A. Musen

Stanford Medical Informatics
Stanford University
Stanford, CA 94305-5479
{noy, musen}@smi.stanford.edu

Abstract

As researchers in the ontology-design field develop the content of a growing number of ontologies, the need for sharing and reusing this body of knowledge becomes increasingly critical. Aligning and merging existing ontologies, which is usually handled manually, often constitutes a large and tedious portion of the sharing process. We have developed SMART, an algorithm that provides a semi-automatic approach to ontology merging and alignment. SMART assists the ontology developer by performing certain tasks automatically and by guiding the developer to other tasks for which his intervention is required. SMART also determines possible inconsistencies in the state of the ontology that may result from the user's actions, and suggests ways to remedy these inconsistencies. We define the set of basic operations that are performed during merging and alignment of ontologies, and determine the effects that invocation of each of these operations has on the process. SMART is based on an extremely general knowledge model and, therefore, can be applied across various platforms.

1 Merging Versus Alignment

In recent years, researchers have developed many ontologies. These different groups of researchers are now beginning to work with one another, so they must bring together these disparate source ontologies. Two approaches are possible: (1) merging the ontologies to create a single coherent ontology, or (2) aligning the ontologies by establishing links between them and allowing them to reuse information from one another.

As an illustration of the possible processes that establish correspondence between different ontologies, we consider the ontologies that natural languages embody. A researcher trying to find common ground between two such languages may perform one of several tasks. He may create a mapping between the two languages to be used in, say, a machine-translation system. Differences in the ontologies underlying the two languages often do not allow simple one-to-one correspondence, so a mapping must account for these differences. Alternatively, Esperanto language (an international language that was constructed from words in different European languages) was created through merging: All the languages and their underlying ontologies were combined to create a single language. Aligning languages (ontologies) is a third task. Consider how we learn a new domain language that has an extensive vocabulary, such as the language of medicine. The new ontology (the vocabulary of the medical domain) needs to be linked in our minds to the knowledge that we already have (our existing ontology of the world). The creation of these links is alignment.

We consider merging and alignment in this paper.

For simplicity, throughout the discussion, we assume that only two ontologies are being merged or aligned at any given time. Figure 1 illustrates the difference between ontology merging and alignment. In **merging**, a single ontology that is a merged version of the original ontologies is created. Often, the original ontologies cover similar or overlapping domains. For example, the Unified Medical Language System (Humphreys and Lindberg 1993; UMLS 1999) is a large merged ontology that reconciles differences in terminology from various machine-readable biomedical information sources. Another example is the project that was merging the top-most levels of two general common-sense-knowledge ontologies—SENSUS (Knight and Luk 1994) and Cyc (Lenat 1995)—to create a single top-level ontology of world knowledge (Hovy 1997).

In **alignment**, the two original ontologies persist, with links established between them.[1] Alignment usually is performed when the ontologies cover domains that are complementary to each other. For example, part of the High Performance Knowledge Base (HPKB) program sponsored by the Defense Advanced Research Projects Agency (DARPA) (Cohen et al. 1999) is structured around one central ontology, the Cyc knowledge base (Lenat 1995). Several teams of researchers develop ontologies in the domain of military tactics to cover the types of military units and weapons, tasks the units can perform, constraints on the units and tasks, and so on. These developers then align these more domain-specific ontologies to Cyc by establishing links into Cyc'c upper- and middle-level ontologies. The domain-specific ontologies do not become part of the Cyc knowledge base; rather, they are separate ontologies that include Cyc and use its top-level distinctions.

[1] Most knowledge representation systems would require one ontology to be included in the other for the links to be established.

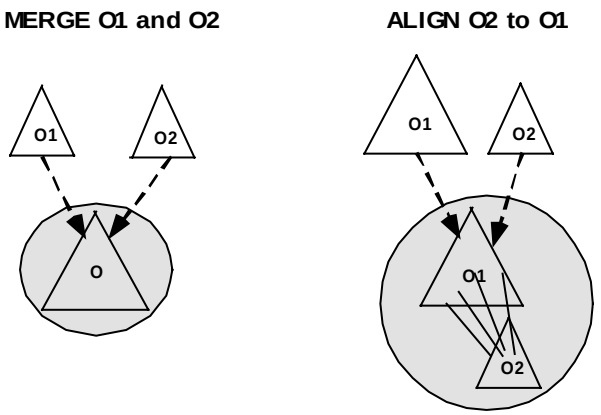

Figure 1. Merging versus alignment. O_1 and O_2 are the original ontologies, the dotted lines represent the transition that an original ontology undergoes, and the gray area is the result. The result of merging is a single ontology, O; that of alignment is persistence of the two ontologies with links (solid lines in the figure) established between them.

Until now, ontology designers have performed this complicated process of ontology merging and alignment mostly by hand, without any tools to automate the process or to provide a specialized interface. It is unrealistic to hope that merging or alignment at the semantic level could be performed completely automatically. It is, however, possible to use a hybrid approach where a system performs selected operations automatically and suggests other likely points of alignment or merging. The easiest way to automate the process partially is to look for classes that have the same names and to suggest that the user merges these classes. We developed an algorithm for ontology merging and alignment, SMART, that goes beyond simple class-name matches and looks for linguistically similar class names, studies the structure of relations in the vicinity of recently merged concepts, and matches slot names and slot value types.

We also wanted to provide specialized tools for merging and alignment. To be effective, these tools should support an ontology developer who is working simultaneously with several ontologies; should provide easy import and export of concepts among these ontologies; and should check whether consistency is maintained after the performed operations, while allowing for inconsistent intermediate states.

We therefore undertook the following research:

- We defined a set of formalism-independent basic operations that are performed during ontology merging or alignment. Naturally, this set overlaps with the set of all operations performed during ontology development. However, none of these two sets—the set of all operations in a design of single ontology and the set of merging and alignment operations—properly subsumes the other.
- We developed SMART, an algorithm for semi-automatic ontology merging or alignment. SMART starts by automatically creating a list of initial suggestions based on class-name similarity. Then, for each operation invoked by the user, based on the classes and slots in the vicinity of that operation, SMART performs a set of actions automatically, makes suggestions to guide the user, checks for conflicts, and suggests ways to resolve these conflicts.
- We designed (and are currently implementing) a specialized tool to perform ontology merging and alignment. The tool is an extension of the Protégé ontology-development environment (Protégé is described in Section 7 and in the article by Musen and colleagues (Musen et al. 1995)).

The knowledge model underlying SMART (described in Section 2) is compatible with the Open Knowledge Base Connectivity (OKBC) protocol (Chaudhri et al. 1998). We did not make any assumptions about the underlying knowledge model that are more detailed than OKBC-compatibility, so our results are applicable to a wide range of systems.

We describe our experience in ontology alignment in Section 3. Section 4 contains an overview of SMART. Section 5 defines the set of basic operations in merging and alignment, and presents a detailed example of one such operation. We give an overview of related work in Section 6, and discuss possible future extensions to our framework in Section 7. Section 8 concludes the paper.

2 Knowledge model

The knowledge model underlying SMART is frame-based (Minsky 1981) and is designed to be compatible with OKBC (Chaudhri et al. 1998). At the top level, there are classes, slots, facets, and instances:

- **Classes** are collections of objects that have similar properties. Classes are arranged into a subclass–superclass hierarchy with multiple inheritance. Each class has slots attached to it. Slots are inherited by the subclasses.
- **Slots** are first-class objects (as in OKBC), and they are named binary relations between two class frames or between a class frame and a simple object (such as a string, number, or symbol). Slots can be single-valued or multivalued. Each slot has associated with it a name, domain, value type, and, possibly, a set of facets.
- **Facets** are frames that impose additional constraints on slots in a class frame. Facets can constrain cardinality or value type of a slot, contain documentation for a slot, and so on.
- **Instances** are individual members of classes. Any **concrete** (as opposed to **abstract**) class can have instances.

These definitions are the only restrictions that we impose on the knowledge model underlying SMART. Since such knowledge model is extremely general, and many existing knowledge-representation systems have knowledge models compatible with it, the solutions to merging and alignment produced by SMART can be applied over a variety of knowledge-representation systems.

Adopting an OKBC-compatible knowledge model also makes it possible to implement SMART as an OKBC client, where most of the work in accessing a knowledge base is performed on the server side. We translated the set of basic merging and alignment operations (Table 1) to OKBC using OKBC's scripting language. Most of the operations from this set are simple combinations of one or two operations from the OKBC specification (Chaudhri et al. 1998). Several of the operations, however, require more extensive scripts. If the OKBC front end included these more complex operations, then OKBC would provide better support for merging and alignment.

3 Experience in ontology alignment

One of our main motivations for developing algorithms and tools for ontology merging and alignment was our participation in the task of aligning ontologies that were developed as part of HPKB project (Cohen et al. 1999; HPKB 1999). This project comprises several teams designing ontologies independently and then bringing together these ontologies to solve complex reasoning problems. Therefore, merging and alignment of the ontologies developed by different teams is an important part of HPKB. The Cyc knowledge base (Lenat 1995) was adopted as the common upper-level ontology. The participating teams developed ontologies for various parts of the military domain: military units, military tasks, purposes of military actions, constraints on military units, and so on. These ontologies then were aligned with one another and with the upper level. Even though the researchers made the effort to have the ontologies aligned in the first place, often the complete alignment at the design time turned out to be impossible: Several of these ontologies were partially developed before Cyc was adopted as the upper level, and a few developers found it easier to formalize their part of the domain first and to align that part to Cyc afterward. In the end, after the ontologies were delivered, the alignment process had to take place.

Consider, for instance, the ontology that formalizes the knowledge about constraints on military units (`Unit-Constraints` ontology). The information in this ontology includes the facts of how fast a unit of a certain type can move under specific terrain and light condition, how much fuel it consumes, and how much fuel it can store, what its combat power is, and so on. Undoubtedly, this ontology overlaps both with the upper-level Cyc ontology and with domain-specific ontologies, such as the ontology of military units. The alignment process in this case included (but was not limited to) the following activities:

- We identified the content that overlapped with the already existing one, and either removed it from the `Unit-Constraints` ontology, or moved part of it to other ontologies. For instance, the developers of the `Unit-Constraints` ontology defined several military-unit specialties (such as `foot infantry`) that were not present in the ontology of military units at that time. We moved these new unit specialties to the ontology of military units.
- Concepts that were at the top level of the `Unit-Constraints` ontology became subclasses of the more general concepts in the upper level (for example, a `Constraints-On-Units` class, which was at the top level of the constraint hierarchy, became a subclass of the `Resource-Constraints` from the upper ontology).
- We established value-type links for the many slots that had value types defined in the upper ontology, such as `Speed`, `Capacity`, and `Rate`.
- We changed several concept names to conform to naming conventions adopted in other ontologies.
- We added axioms to link the information in the concepts to the information in upper level.

Based on this and other practical experiences, we developed a general approach to ontology merging and alignment that facilitates and partially automates this tedious and time-consuming process.

4 Description of the SMART algorithm

We now describe how the SMART algorithm works. The word in parentheses at the beginning of an algorithm step indicates whether a user executes the step (User) or whether SMART can perform the step automatically (SMART).

1. (User) *Setup*: Load two ontologies and custom tailor the strategy preferences described in Section 4.1.
2. (SMART) *Generate the initial list of suggestions*: Scan the class names in both ontologies to find linguistically similar names. (Linguistic similarity can be determined in many different ways, such as by synonymy, shared substrings, common prefixes, or common suffixes.) Create the initial list of suggestions to the user. If the process is `merge`, suggest the following:
   - For each pair of classes with *identical* names, either merge the classes or remove one of the classes in the pair.
   - For each pair of classes with *linguistically similar* names, establish a link between them (with a lower degree of confidence than in the first set).

   If the process is `align` and one ontology (say, O_1) has been identified by the user as the less general, then the suggestions are based on the assumption that classes in the less general ontology need to be linked by subclass–superclass relations with the classes in the more general ontology (O_2). Recall, for instance, our earlier example of assigning a parent from the more general ontology (`Resource-Constraint`) to a concept from the more specific one (`Constraint-On-Units`). So, for each class `C` at the top level of O_1, the following suggestions are made with high confidence:
   - If there is a class in O_2 with the same name as `C`, merge the two classes.
   - Otherwise, find a parent for class `C`.

3. (User) *Select and perform an operation*: The operation the user selects can be, but does not have to be, from the current list of suggestions. The set of possible basic operations is defined in Section 5.1.
4. (SMART) Perform automatic updates and create new suggestions: For the concepts involved in the operation just performed in step 3, consider linguistic similarity of their class and slot names and the structure of the ontology in their vicinity. Based on the similarity and structural patterns, trigger actions in three categories:
   - Immediately execute additional changes determined automatically by the system.
   - Add to the list of conflicts (described in Section 4.2) the set of new conflicts in the state of the ontology resulting from the user action and suggestions on how to remedy each conflict.
   - Add to the list of suggestions (described in Section 4.2) the set of suggestions to the user on what other actions can be performed next.
5. Repeat steps 3 and 4 until the ontologies are fully merged or aligned.

Creation of the initial list of suggestions in step 1 is mostly syntactic: SMART makes its decisions based on *names* of frames and not on *positions* of frames in the ontology or frames' participation in specific *relations*. Step 4 includes semantic analysis as well: SMART considers the *types* of relations involving current concepts and arguments of these relations. In section 0, we give examples of the specific semantic relations that SMART considers in step 4.

In the remainder of this section, we describe how a user can custom tailor the behavior adopted by the algorithm (Section 4.1) and the structure and content of the lists that contain the current sets of suggestions (Section 4.2). One of SMART's central abilities is to determine new actions and suggestions based on the operation just performed. Thus, for each basic merging or alignment operation defined in Section 5.1 (Table 1), we must define the set of automatic actions that the operation may trigger, the conflicts that it may produce, and the new points of merging that it may create. We show how the decisions in each of these three categories are made for one such operation, *merging classes*, in Section 0.

4.1 Custom tailoring of the behavior

The user can custom tailor the behavior of the system by setting the values of the preference variables. We describe two of the preference variables (that determine the strategy and the preferred ontology) in this section.

Preferred ontology

When we align or merge ontologies, one or more ontologies often are more stable than are the others in ontology set. *Stable* here does not mean superior, or better tested, or more established. Rather, an ontology is stable if, for various reasons, making changes to it is highly discouraged and should be done only if absolutely necessary. An ontology can be designated as **stable** because, for instance, it is heavily reused already and any changes to it would affect all the systems that are reusing the ontology. A user can designate the more stable ontology as the `preferred` one. The `preferred` setting guides, for instance, the choice of a name for a concept that results from merging two differently named concepts: The name of the concept from the `preferred` ontology becomes the merged concept's name. Similarly, the default resolution of conflicts favors of the preferred ontology.

Preferred policy: `merge` or `override`

The `preferred-policy` variable guides the choices that SMART makes when conflicts arise among different slots of a concept. Concepts can get new slots as a result of operations such as `merge-classes` or `add-parent`. Some of these newly acquired slots may have the same names as the old slots, but may have different value types (or sets of facets). By setting the policy to `override`, the user requests that the value received from the `preferred` ontology automatically overrides the other ones. If the user wishes to select the overriding value himself, he sets the policy to `merge`.

For example, suppose that we are merging two concepts that describe a military unit. Suppose also that one `Military-Unit` concept comes from the `preferred` ontology and that this concept has a `rate-of-movement` slot with the value-type `Speed`; the other `Military-Unit` concept has a slot with the same name but value-type `Number`. If the policy is `override`, the concept that results from merging these two `Military-Unit` concepts will have the `rate-of-movement` slot with value-type `Speed` (the value type that the merged concept received from the `preferred` ontology). If the policy is `merge`, SMART will place the resolution of the conflict as an item on the list of conflicts.

The setting of `merge` and `override` policies is similar to merge and override composition rules (Ossher et al. 1996) for composition in subject-oriented programming (subject-oriented programming is an approach to object-oriented programming that we describe in Section 6).

4.2 Data structures

We have referred several times to the two basic data structures that SMART maintains throughout the merging and alignment process and that it uses to guide the user in the process: the list of conflicts that resulted from the user's actions and the list of suggestions for the next action. We call these structures the `Conflicts` list and the `ToDo` list, respectively. The items on the `Conflicts` list represent inconsistencies in the current state of the knowledge base that must be resolved before the knowledge base can be in a logically consistent state. For example, if we merge two slots with conflicting facet values, SMART puts the operation of removing one of the facets in the conflict on the `Conflicts` list. Before closing the knowledge base, the user must perform this operation or ask SMART to decide which facet to remove. Alternatively, if two slots of a

merged concept have similar (but not identical) names and the same value type, then merging these slots is an operation that probably should be performed. However, execution of this operation is not required for internal consistency of the knowledge base. SMART places such an operation on the ToDo list.

ToDo list

An item on the ToDo list is either a single operation or a disjunction of operations. In the latter case, the underlying semantics is that only one of the operations in the disjunction needs to be performed. SMART chooses the operations for the ToDo list from the set of basic merging and alignment operations (Table 1) and specifies their arguments (and, possibly, suggested results).

For example, suppose that, as a result of the addition of a new parent, a Military-Unit concept acquired a combat-power slot with a name similar to that of one of its old slots, unit-combat-power, and the same value type (Number). SMART adds the following suggestion to the ToDo list:

```
(or (remove-slot Military-Unit combat-power)
  (remove-slot Military-Unit
         unit-combat-power))
```

Conflicts list

An item on the Conflicts list consists of a description of a conflict and a suggested action that would remedy the problem. It also contains a default solution that will be invoked if the user asks SMART to resolve the conflict.

For instance, using our previous example, if the two slots (that is, the one that the Military-Unit concept already had and the one that the concept inherited from the newly added parent) had the same name, combat-power, but different value types, Number and Integer, and the policy was merge, SMART would add the following record to the Conflicts list:

```
Problem:
 conflicting value types: Integer and
 Number at combat-power in Military-Unit
Solution:
 (or (remove-slot Combat-Unit
    (combat-power :value-type Integer))
   (remove-slot Combat-Unit
    (combat-power :value-type Number)))
Default solution:
 (remove-slot Combat-Unit
    (combat-power :value-type Number))
```

The default solution is based on which ontology was designated as preferred. If the user asks SMART to resolve all conflicts automatically, SMART executes the default solution.

Other attributes of items in the Conflicts and ToDo lists

Priority. SMART assigns priority to the items in the Conflicts and ToDo lists and orders the items in the lists with the higher-priority items more easily accessible. The priority value can be based on one or more of the following:

- SMART's certainty about the item: The higher the certainty, the higher the priority
- The item's age: Items triggered by the more recent operations have higher priority because they refer to the concepts that are closer to the user's current focus
- The number of references to the concepts in the operation: The more elements in the ontology would be affected if the operation is invoked, the higher the priority of the operation.

Feedback and explanation. Each item in the Conflicts and ToDo lists has references back to the triggering operation and to the rule that SMART used to put the item on the list (in the formal or natural-language form). The reference and the rule explain to the user why a particular item appears on the list. The user can also disable the rules that he considers not useful or impediments.

5 The set of basic operations in merging and alignment

In this section, we define the set of basic operations for ontology merging or alignment. Then, we describe one of these operations (merge-classes) in detail: We discuss the automatic operations and additions to the ToDo and Conflicts lists that the merge-classes operation triggers.

5.1 Overview of the basic merging and alignment operations

OKBC specifies a superset of all operations that can be performed during an ontology-design process (Chaudhri et al. 1998). This superset of operations is sufficient for the processes of ontology merging and alignment, because the latter can also be treated as the ontology design processes. Therefore, we can argue that the only task that a merging and alignment tool must do is to support the OKBC operations. However, judging from our own experience in merging and alignment, we believe that not all the operations required for general ontology design are used in the more specialized processes of ontology merging and alignment. For instance, during merging or alignment, we rarely create new classes completely from scratch (the new classes are usually modifications of the existing ones); creation of new slots or facets is even less common. Likewise, several operations are performed only during ontology merging and alignment and are usually not present at the ontology-design stage. For example, merging two concepts into one is common in ontology merging. Moving a concept from one ontology to another is often performed during alignment. Although these operations can be expressed as combinations of primitive operations from the OKBC set, the user needs to perform these sequences of basic operations in one step.

Table 1 summarizes the basic operations for ontology merging and alignment that we identified. Several of these operations are normally performed only during ontology merging, several operations are used only in ontology alignment, and most operations apply to both situations. The list in Table 1 is not exhaustive; for example, we have not represented operations dealing with updates to facets and instances (see Section 7).

We translated the operations from Table 1 to OKBC using the OKBC's scripting language. Most of the operations from this set are simple combinations of one or two operations from the OKBC specification. Several of them, however, require more extensive scripts. These operations can extend the OKBC front end to provide support for merging and alignment in OKBC.

Identification of these basic operations and of the consequences of invoking them is the central task in our approach to ontology merging and alignment. For each operation, we can specify the rules that determine the specific members of the three sets updated by the operation: (1) changes that should be performed automatically, (2) conflicts that will need to be resolved by the user, and (3) new suggestions to the user. Having specified these rules, we can have the system automatically create members of these three sets based on the arguments to a specific invocation of an operation.

Operator	**Arguments**	**Description**	**Need for the operator**	**When**
`Merge-classes`	C_1, C_2	Merge C_1 and C_2 to create a new class, C_3	Classes represent similar concepts	Merge
`Remove-class`	C_1	Remove C_1 from the ontology	The class is already represented in the other ontology	Both[1]
`Remove-parent`	C_1, C_2	Precondition: C_2 is a parent of C_1. Remove C_2 from the list of C_1's parents	*Merging:* Extra parents are the result of an earlier merging *Alignment:* A more appropriate parent is in another ontology	Both
`Add-parent`	C_1, C_2	Add C_2 to the list of parents of C_1	An appropriate parent found in another ontology	Align
`Rename`	C_1, N_1	Change a name of C_1 to N_1	Need to conform to naming conventions in another ontology	Both
`Move`	C_1, O_1, O_2	Move C_1 from O_1 to O_2	All the concepts similar to C_1 are in O_2	Align
`Remove-slot`	C_1, S_1	Remove S_1 from the list of slots of C_1	*Merging:* Extra slots are the result of an earlier merging *Alignment:* A similar slot was inherited from a new parent	Both
`Move-slot`	S_1, C_1, C_2	Precondition: S_1 is a slot in C_1. Remove S_1 from the list of slots in C_1 and add it to the list of slots in C_2	The slot is more appropriately defined in a parent	Both
`Rename-slot`	S_1, N_1	Change a name of S_1 to N_1	Need to conform to naming conventions in another ontology	
`Rename-slot-in-class`	C_1, S_1, N_1	Change a name of S_1 in C_1 to N_1[2]	Need to fix slot names in merged concept	Both
`Change-slot-range-in-class`	C_1, S_1, T_1	Change value type of S_1 in C_1 to T_1	There is a more appropriate value type in another ontology	Both

Table 1. Partial set of atomic operations in ontology merging and alignment processes. C_1 and C_2 are class frames, S_1 is a slot frame, N_1 is a string, O_1 and O_2 are ontologies, and T_1 is a value type for a slot (can be a class frame or a simple type, such as number). For each operation, we define what arguments it takes, give its natural-language description, provide a commonly encountered reason for performing this operation, and indicate the process (merging, alignment, or both) where it is applicable.

[1] In the *merging* process, this could be considered to be a special case of `merge-class`

[2] If S_1 is attached to other classes, its copy is created, named N_1, and attached to C_1

Combat-Arms-Unit: subclass-of **Military-Unit**	
echelon:	**Military-Echelon**
combat-power:	**Integer**
rate-of-movement:	**Rate**
fuel-capacity:	**Capacity**

Ground-Maneuver-Unit: subclass-of **Modern-Military-Unit**	
echelon:	**Military-Echelon**
combat-power:	**Number**
movement-rate:	**Rate**

Figure 2. Example of classes to be merged. The words in italics are names of slots followed by value types.

In the next section, we discuss one of these operations in detail. We show how SMART creates the additions to the two running lists (`ToDo` and `Conflicts`) and what changes it makes automatically as a result of such an operation.

5.2 Example: `merge-classes`

As we defined in Table 1, the `merge-classes` operation takes two class-frame arguments and merges the two classes into one. A user would usually perform the `merge-classes` operation to merge two classes that come from two different source ontologies and that represent similar concepts. Figure 2 shows an example of two classes from two different ontologies; both cover the domain of military units: `Combat-Arms-Unit` and `Ground-Maneuver-Unit`. These two classes—`Combat-Arms-Unit` and `Ground-Maneuver-Unit`—represent the same collection of military units; therefore, they must be merged if the two source ontologies are merged. We refer to these two classes in the examples throughout this section. We assume that the `Combat-Arms-Unit` class originally is defined in the `preferred` ontology.

Naming of the new concept. If there is a `preferred` ontology, then the name of the concept that is the result of the `merge-classes` operation is the same as the one that came from the `preferred` ontology. If there is no `preferred` ontology and the classes being merged have different names, then the user must choose the name for the new concept. In our example, the merged class will be named `Combat-Arms-Unit` because this name is the one that came from the preferred ontology.

In Section 5.1, we defined three sets of actions that an operation can trigger: (1) performing automatic updates, (2) identifying conflicts, and (3) presenting suggestions for further actions. Now we describe what actions from each of these three sets are triggered specifically by the `merge-classes` operation.

5.2.1 Automatic updates

After the user has invoked the `merge-classes` operation, SMART automatically performs the following operations in addition to simply merging classes.

Updating of references. SMART updates all references to the merged classes in both ontologies to refer to the new class, and removes all the duplicate relations created by this step (such as two subclass–superclass relations between the same pair of classes). Consequently, sets of parents, children, and slots of the new class are unions of sets of parents, children, and slots of the original classes respectively. For example, the set of parents for the new `Combat-Arms-Unit` class now consists of two classes: `Military-Unit` and `Modern-Military-Unit`.

Setting of the ontology affiliation. The merged class and all its slots become part of the merged ontology.

Removal of slots with duplicate names. If the merged concept has two slots with the same name but different value types and the policy is `override`, the new concept keeps only the slot that came from the `preferred` ontology. For example, the new `Combat-Arms-Unit` class acquired two slots named `combat-power`: one with the value-type `Integer` and the other with the value-type `Number`. The merged concept keeps only the slot with the value-type `Integer` because this slot came from the `preferred` ontology and the policy was `override`.

5.2.2 Conflicts to be resolved by the user

After the merging operation, SMART determines the conflicts in the state of the ontology that the user will need to resolve.

Merging of slots with duplicate names. A conflict that a user needs to resolve arises when the merged concept has two slots with the same name but different value types and the policy is `merge`. For our example, SMART adds the following record to the `Conflicts` list:

```
Problem:
 conflicting value types: Integer and
 Number at combat-power in Military-Unit
Solution:
 (or (remove-slot Combat-Unit
    (combat-power :value-type Integer))
   (remove-slot Combat-Unit
    (combat-power :value-type Number)))
Default solution:
 (remove-slot Combat-Unit
    (combat-power :value-type Number))
```

5.2.3 Suggested actions

As a result of the `merge-classes` operation, SMART also creates a list of suggested actions based on the concepts in the structural vicinity of the merging operation.

Merging of superclasses with name matches. If the set of superclasses of the merged concept contains classes with the same or similar names, SMART places the task of merging these classes on the `ToDo` list with high priority: Not only do these concepts have similar names, but also they play the same role—superclass—for the merged concept. In our example, the set of superclasses of the merged `Combat-Arms-Unit` concept consists of the `Military-Unit` and `Modern-Military-Unit` classes. These two superclasses—`Military-Unit` and `Modern-Military-Unit`—have linguistically similar names, and

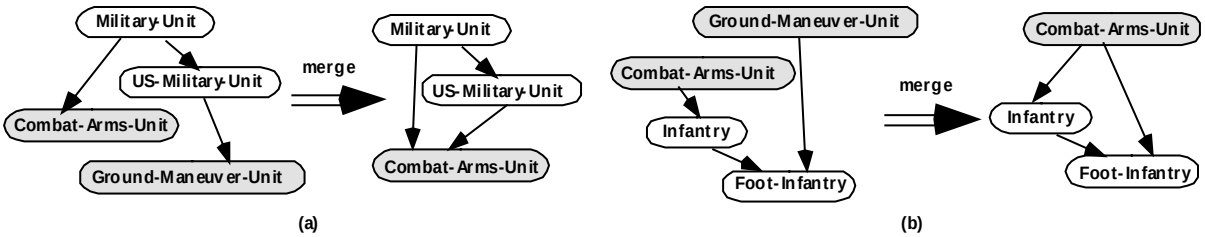

Figure 3. Redundancy resulting from merging. Single arrows represent subclass relations. The gray color indicates concepts that are directly involved in the merging operation.

they play the same role to the newly merged concept. Therefore, the task of merging them has a high priority on the `ToDo` list.

Merging of subclasses with name matches. The task of merging subclasses that have the same or similar names becomes a high-priority item in the `ToDo` list.

Merging of slots with similar names. If two slots attached to the merged concept have *similar* names, SMART suggests that the user removes one of them. Removing one of the two slots that have similar names and the *same* value type gets higher priority than that of removing one of the slots if they have *different* value types. For example, `Combat-Arms-Unit` now has `rate-of-movement` and `movement-rate` slots, both with `Rate` as value type. It is highly probable that the merged class needs to retain only one of these two slots.

Suggestion of actions based on structural indications. The process described so far may result in the redundancy illustrated in Figure 3: One of the direct parents of the merged concept is a (possibly indirect) subclass of another direct parent of the same concept. Then, SMART suggests that the user removes one of these two parents of the merged concept. For example, suppose that the `Military-Unit` and the `Modern-Military-Unit` concepts were merged first and `Ground-Maneuver-Unit` was a direct subclass of `US-Military-Unit` (Figure 3a). Then, after `Combat-Arms-Unit` and `Ground-Maneuver-Unit` are merged, SMART adds the following to the `ToDo` list:

```
(remove-parent Combat-Unit Military-Unit)
```

The set of subclasses of the merged concept may contain a similar redundancy: One of the children of the new concept may be a subclass of another child of the same concept (Figure 3b). Thus, for the example in the figure, SMART suggests the following action:

```
(remove-parent Foot-Infantry Combat-Unit)
```

Removing such redundant subclass–superclass links is another suggestion that results from invocation of `merge-classes` operation.

5.3 Logging of basic operations as a mechanism for periodic merging or alignment

The ontology merging or alignment process is not a one-shot exercise. After the user has merged (or aligned) two ontologies and has developed a new consistent ontology, one of the source ontologies may change. Ideally, reapplication of the merging (or alignment) process to the changed ontologies should be almost automatic. The ability to reapply the changes is called *periodic merging* (MacGregor et al. 1999).

Subject-oriented programming (Ossher and Harrison 1992), which we discuss in more detail in Section 7, solves the problem of reapplying changes to the updated sources by using scripted **extensions**: Each update to a class hierarchy is defined by a set of rules that are then applied to the hierarchy. If the hierarchy changes, this extension is reapplied to it at no additional human cost.

Having defined the set of basic operations for our system, we can maintain a log of the basic operations that were performed with their arguments and results. If the ontologies are updated at the points other than the ones explicitly used in the log, the reapplication is completely automatic. Otherwise, the user must make adjustments to the log, though. SMART can automatically identify points of adjustment. We are currently developing the specific heuristics for periodic merging and alignment.

6 Related work

Researchers in computer science have discussed automatic or tool-supported merging of ontologies (or class hierarchies, or object-oriented schemas—the specific terminology varies depending on the field). However, both automatic merging of ontologies and creation of tools that would guide the user through the process and focus his attention on the likely points for actions are in early stages. In this section, we discuss the existing merging and alignment tools in ontology design and object-oriented programming.

6.1 Ontology design

One of the few working prototypes of an ontology-merging tool is a system developed by the Knowledge Systems Laboratory (KSL) at Stanford University (Fikes et al. 1999). This system is early prototype that solves initial problems in ontology merging. It can bring together ontologies developed in different formalisms and by different authors.

The KSL's approach to merging has similarities to the one described in this paper: The KSL's algorithm generates a list of suggestions based on the operations performed by the user. The process starts by running a matching algorithm on class names in both ontologies to suggest the merging points. The matching algorithm looks for the exact

match in class names, or for a match on prefixes, suffixes, and word roots of class names. A user can then choose from these matching points, or proceed on his own initiative. After each step performed by the user, the system generates a new set of suggestions.

The KSL's approach to merging is also different from SMART's in many respects. KSL's merging algorithm does not consider slots at all. The process for name matches appears to be depth first and may cause the user to lose focus by finding matches much deeper in the class hierarchy than the current point of concentration. On the other hand, SMART does not consider disjoint classification of categories, whereas the KSL's algorithm does.

The KSL's tool employs user-interface metaphors from the Ontolingua ontology editor (Ontolingua 1995)—an advantage for the many ontology developers familiar with the ontology editor. The Ontolingua interface is not always easy to use, however, and this drawback seems to carry over to the merging tool.

Another ontology-merging and alignment project at Information Science Institute (ISI) at the University of Southern California (Chapulsky, Hovy, and Russ 1997) attempted to merge extremely large top-level ontologies: Cyc (Lenat 1995) and SENSUS (Knight and Luk 1994). The SENSUS ontology itself resulted from manual merging of the PENMAN Upper Model (Penman 1989), WordNet (Miller 1995), and several other ontologies. In the ISI approach, the creation of an initial list of alignment suggestions relies on more than just class names. The concept features that ISI scores and combines to produce the suggestion list include concepts whose names have long substrings in common; concepts whose definitions (that is, documentation) share many uncommon words; and concepts that have sufficiently high combined name-similarity scores for nearby siblings, children, and ancestors. Experiments have shown that the initial list of suggestions filters out many uninteresting concepts. However, the algorithm stops there: After this initial list is created, it was up to the ontology developer to continue the ontology alignment.

As mentioned in Section 1, medical vocabularies provide a rich field for testing of various ontology-merging paradigms. Not only is there a wide variety of large-scale sources, but also medicine is a field where standard vocabularies change constantly, updates are issued, different vocabularies need to be reconciled, and so on. Oliver and colleagues (Oliver et al. 1999) explored representation of change in medical terminologies using a frame-based knowledge-representation system. The authors compiled a list of change operations that are relevant for the domain of medical terminologies, and developed a tool to support these operations. Many of the operations are similar to the ones described in Section 5. However, the user has to do all the operations manually; there is no automated help or guidance.

6.2 Object-oriented programming

One of the better-developed works in the area of ontology merging comes not from the researchers in artificial intelligence, but rather from the researchers in object-oriented programming, who face problems similar to those of ontology researchers. They too may need to bring together several object-oriented class hierarchies (with the associated methods) to create a new product. Subject-oriented programming (SOP) (Harrison and Ossher 1993) supports building of object-oriented systems through composition of subjects. **Subjects** are collections of classes that represent subjective views of, possibly, the same universe. For instance, a class `Shoe` in a shoemaker universe would probably have attributes different from those of the same class in a shoe-seller universe. The class hierarchies that define shoes (and their types) from these different points of view would be the two subjects. If these subjects are to work together (for instance, if the shoemaker supplies shoes to the shoe seller), their corresponding class hierarchies must be merged.

The formal theory of subject-oriented composition (Ossher et al. 1996) defines a set of possible composition rules, these rules' semantics, and the ways that the rules work with one another. Interactive tools for subject-oriented composition are currently under development. There are important differences and similarities in the SOP approach and requirements for class-hierarchy merging and the ontology merging described here.

We begin with the description of the similarities. Merging of class hierarchies covering similar domains is the central task of both ontology merging and subject-oriented composition. The slots of the merged classes (instance variables in object-oriented terminology) must be merged, too. Simply creating union of slots leads to conflicts that must be resolved as well. In fact, our idea of preferred strategy (merge or override) was inspired by the merge and override composition rules used in specifying subject-oriented composition. In both cases, completely automatic matching (or creation of composition rules) is not possible, and interaction with the user (preferably through easy-to-use graphical tools) is needed. In Section 5.3, we introduced the notion of extensions in subject-oriented programming and described the relation of extensions to periodic ontology merging: An extension is a set of rules that defines updates to a class hierarchy. Reapplying extensions after a class hierarchy changes is similar to reapplying the set of logged merging and alignment operations after an ontology changes.

One of the major differences between SOP and our work stems from the fact that subject-oriented composition also needs to compose methods associated with classes. In an ontology, however, classes do not have methods associated with them. On the other hand, ontologies may have axioms that relate different classes or slot values. Object-oriented class hierarchies do not include axioms. Alignment (as opposed to merging) is extremely uncommon in composition of object-oriented hierarchies, whereas it is common in ontology design.

7 Future directions

We currently are implementing SMART as an extension to the Protégé ontology development environment (Musen et al. 1995)—a graphical and interactive ontology-design and knowledge-acquisition environment developed in our laboratory. Protégé helps ontology developers and domain experts to perform knowledge-management tasks. Its graphical user interface is as easy to use as is possible without significant loss in expressiveness. The ontology developer can quickly access the relevant information about classes and instances whenever he requires this information and can use direct manipulation techniques to edit an ontology. Tree controls allow quick and simple navigation through a class hierarchy. Protégé uses forms as the interface for the user to fill in slot values for instances. Protégé knowledge model is OKBC-compatible (Grosso et al. 1998).

Throughout the paper, we have mentioned possible future research. We summarize these ideas here.

7.1 Extensions to the model

We first describe possible extensions to the SMART algorithm. We then discuss how to evaluate formally an ontology-merging and alignment tool (Section 7.2)

Clusters

When a merging or alignment process is underway, at any point there may be several areas in the source ontologies where "something is happening." That is, there can be several points of contact between the ontologies that are under consideration. Each of these points is a cluster: a set of concepts closely related to one another for which alignment or merging work is needed. For example, slots for the recently merged concepts need to be matched, or conflicts that resulted from that merge need to be resolved. Figure 4 illustrates the notion of a cluster.

If we define these clusters, then we can group the actions on the ToDo and Conflicts lists based on the cluster to which they belong, and the list entries from the current cluster can be placed at the top of the lists.

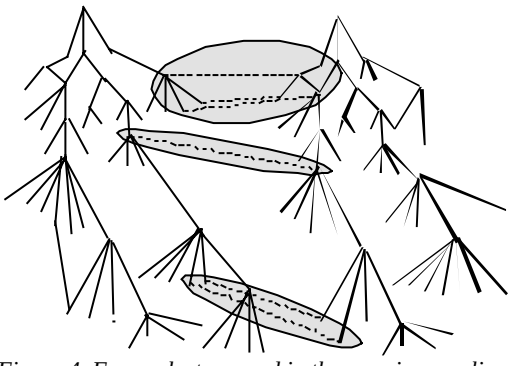

Figure 4. Focus clusters used in the merging or alignment process. Trees represent source ontologies; dotted lines represent recently established links between ontologies.

Facets and instances

Facets impose additional constraints on slots in a frame. For example, a facet can define cardinality constraints on a slot. Suppose that we merged two classes and that both classes have a slot with the same name and the same value type. In this case, SMART automatically merges the slots and adds a slot with that name and value type to the newly created concept. However, in one of the source ontologies, the slot that was merged may have had a facet that defined its maximum cardinality as 2. In the second source ontology, the otherwise-identical slot may have had a minimum cardinality of 3. These cardinality restrictions are incompatible, so the algorithm must reconcile them.

The OKBC specification (Chaudhri et al. 1998) lists standard facets for slots, such as maximum and minimum cardinality, inverse, numeric minimum and maximum, documentation, and so on. In the merging and alignment process, it is desirable to consider each of these standard facets, correlation among possible facet values, and ways of reconciling conflicting values. We can also use values of the standard facets to create additional suggestions: For example, if we merge two slots, we probably need to merge the inverses of these slots as well.

Many ontologies include definitions of not only classes (collections of similar objects), but also instances (individual objects). It would be useful to include automation and tool support for instance alignment or merging.

Performing merging and alignment in non-OKBC frameworks

We currently assume that the source ontologies conform to the OKBC knowledge model. More specifically, we rely on having a frame-based model with classes, slots, and facets as first-class objects (see Section 2). In the future, we would like to extend SMART to other knowledge models. We also pay little attention to axioms. Instead, we rely on the knowledge-representation system's built-in ability to enforce axioms. The next important step is to consider how to apply to heavily axiomatized frameworks semi-automated strategies and generation of suggestions similar to the ones described here.

7.2 Evaluation of the merging and alignment process

For any algorithm or tool, it is important to evaluate performance and to validate empirically achievement of goals. Empirical evaluation generally has not been a strong part of artificial intelligence, although some researchers have undertaken it (Cohen 1995).

We can use the standard information-retrieval metrics of precision and recall to evaluate the performance of SMART. **Recall** can be measured as the ratio of the suggestions on the `ToDo` list that the user has followed to the total number of merging and alignment operations that he performed. **Precision** can be measured as the fraction of the suggestions from the `ToDo` list that the user followed.

This metric could be augmented by a measure of how many of the conflict resolution strategies in the `Conflicts` list were satisfactory to the user (and how many he had to alter).

The number of operations from Table 1 that were invoked during the ontology-merging or alignment process also could serve as a measure of how closely related or far apart the original ontologies were before the process began. Note that the number of performed operations does not measure the *quality* of the process itself. Collecting information about operations performed during merging or alignment could also contribute to a study of the value of prior knowledge: The number of concepts that were *removed* could be used as a measure of the extra knowledge that a knowledge engineer had to create because he did not use the base ontology from the beginning. This metric would not provide a *direct* indication of the value of using the base ontology from the start, because it does not account for the extra body of knowledge could have slowed the development.

8 Conclusions

We described a general approach to ontology merging and alignment. We presented SMART—an algorithm for semi-automatic merging and alignment. We discussed strategies that SMART uses to guide a user automatically to the next possible point of merging or alignment, to suggest what operations should be performed there, and to perform certain actions automatically. We defined the set of basic operations invoked during a merging or alignment process. The more complex operations from this set could be added as front-end operations in OKBC. The strategies and algorithms described in this paper are based on a general OKBC-compliant knowledge model. Therefore, these results are applicable to a wide range of knowledge-representation and ontology-development systems.

Acknowledgments

This paper was greatly improved by the feedback, suggestions, and ideas from William Grosso and Harold Boley. We are very grateful to Lyn Dupré for her help. This work was funded in part by the High Performance Knowledge Base Project of the Defense Advanced Research Projects Agency.

References

Chapulsky, H., Hovy, E. and Russ, T. 1997. Progress on an Automatic Ontology Alignment Methodology. ANSI Ad Hoc Group on Ontology Standards; *available at* http://ksl-web.stanford.edu/onto-std/hovy/index.htm.

Chaudhri, V. K., Farquhar, A., Fikes, R., Karp, P. D. and Rice, J. P. 1998. OKBC: A Programmatic Foundation for Knowledge Base Interoperability. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*: 600-607. Madison, Wisconsin: AAAI Press/The MIT Press.

Chaudhri, V. K., Farquhar, A., Fikes, R., Karp, P. D. and Rice, J. P. 1998. Open Knowledge Base Connectivity 2.0.3, Specification document.

Cohen, P. R. 1995. *Empirical Methods for Artificial Intelligence*. Cambridge, MA: MIT Press.

Fikes, R., McGuinness, D., Rice, J., Frank, G., Sun, Y. and Qing, Z. 1999. Distributed Repositories of Highly Expressive Reusable Knowledge. Presentation at HPKB meeting, Austin, TX; *available at* http://www.teknowledge.com/HPKB/meetings/Year1.5meeting/.

Grosso, W., Gennari, J. H., Fergerson, R. and Musen, M. A. 1998. When Knowledge Models Collide (How it Happens and What to Do). In *Proceedings of the Eleventh Banff Knowledge Acquisition for Knowledge-Bases Systems Workshop*. Banff, Canada.

Harrison, W. and Ossher, H. 1993. Subject-Oriented Programming (A Critique of Pure Objects). In *Proceedings of the Conference on Object-Oriented Programming: Systems, Languages, and Applications (OOPSLA'93)*: 411-428. Washington, DC: ACM Press.

Knight, K. and Luk, S. K. 1994. Building a Large-Scale Knowledge Base for Machine Translation. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*. Seattle, Washington: AAAI Press.

Lenat, D. B. 1995. CYC: A Large-Scale Investment in Knowledge Infrastructure. *Communications of ACM* 38(11): 33-38.

MacGregor, R., Chapulsky, H., Moriarty, D. and Valente, A. 1999. Knowledge Management Tools. Presentation at HPKB meeting, Austin, TX; *available at* http://www.teknowledge.com/HPKB/meetings/Year1.5meeting/.

Miller, G. A. 1995. WordNet: A Lexical Database for English. *Communications of ACM* 38(11): 39-41.

Musen, M. A., Gennari, J. H., Eriksson, H., Tu, S. W. and Puerta, A. R. 1995. Protégé-II: Computer support for development of intelligent systems from libraries of components. In *Proceedings of the Medinfo'95*: 766-770. Vancouver, BC.

Oliver, D. E., Shahar, Y., Shorliffe, E. H. and Musen, M. A. 1999. Representation of Change in controlled medical terminologies. *Artificial Intelligence in Medicine* 15: 53-76.

Ontolingua System Reference Manual 1995: http://www-ksl-svc.stanford.edu:5915/doc/ontolingua/reference-manual.

Ossher, H. and Harrison, W. 1992. Combination of Inheritance Hierarchies. In *Proceedings of the Conference on Object-Oriented Programming: Systems, Languages, and Applications (OOPSLA'92)*: 25-40. Vancouver: ACM Press.

Ossher, H., Kaplan, M., Katz, A., Harrison, W. and Kruskal, V. 1996. Specifying Subject-Oriented Composition. *Theory and Practice of Object Systems* 2(3): 179-202.

Penman 1989. The Penman documentation, Technical report, USC/Information Sciences Institute, Marina del Rey, CA.