A TEMPLATE-BASED APPROACH TOWARD ACQUISITION OF LOGICAL SENTENCES

Chih-Sheng Johnson Hou, Natalya F. Noy and Mark A. Musen
Stanford University, Stanford, CA 94305
{johnsonh, noy, musen} @ smi.stanford.edu

Abstract: Ontology-development languages may allow users to supplement frame-based representations with arbitrary logical sentences. In the case of the Ontolingua ontology library, only 10% of the ontologies have any user-defined axioms. We believe the phrase "writing axioms is difficult" accounts for this phenomenon; domain experts often cannot translate their thoughts into symbolic representation. We attempt to reduce this chasm in communication by identifying groups of axioms that manifest common patterns creating 'templates' that allow users to compose axioms by 'filling-in-the-blanks.' We studied axioms in two public ontology libraries, and derived 20 templates that cover 85% of all the user-defined axioms. We describe our methodology for collecting the templates and present sample templates. We also define several properties of templates that will allow users to find an appropriate template quickly. Thus, our research entails a significant simplification in the process for acquiring axioms from domain experts. We believe that this simplification will foster the introduction of axioms and constraints that are currently missing in the ontologies.

Key words: frame-based system, knowledge acquisition, knowledge representation

1. AXIOMS IN FRAME-BASED SYSTEMS

Frame-based representation systems (FRS) are a popular choice for knowledge representation [6]; their taxonomic categorization of canonical concepts often bears close resemblance to the way humans describe knowledge and is easy to understand. Because of this cognitive simplicity, FRSs serve as efficient tools for knowledge representation and acquisition.

Even though knowledge-representation constraints inherent in FRSs guarantee many advantages, they also foster several limitations across

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35602-0_35
M. A. Musen et al. (eds.), *Intelligent Information Processing*

virtually all systems. A frame-based system alone has limited expressivity: it cannot naturally represent negations, disjunctions, and existential quantification. One cannot restrict a value of one property based on another property. To overcome these limitations, some FRSs employ more expressive compensatory axiom languages, which are based on first-order logic, to encode these relationships.

With the proliferation of approachable user interfaces such as the Protégé-2000 ontology-editing environment [8] for frame-based knowledge acquisition, many domain experts participate in knowledge acquisition, often without the collaboration of knowledge engineers. Domain experts enter information about classes and properties of concepts through a convenient interface. Unfortunately, to specify additional relational information, they encounter an axiom-editing environment that has remained free-text based. The act of conceptualizing a thought in a symbolic representation is often extremely difficult for a domain expert. For example, one may not understand why representing the simple constraint "every employee has a unique ID" in an axiom in first-order logic requires the equivalent English translation of "for every two employees both of whom have IDs, if the two employees are not the same, their IDs cannot be identical." Figure 1 shows this axiom in Protégé Axiom Language (PAL), which is an axiom language for the Protégé-2000 ontology-editing environment based on Knowledge Interchange Format (KIF) [5]. Even if a domain expert understood the formulation of the axiom, he would still have to struggle with the foreign syntax (Figure 1). These factors can lead to a failure to encode critical knowledge realizable only through axioms. For example, in Ontolingua [7], only about 10% of the ontologies include manually generated axioms.

To achieve truly meaningful transfer of knowledge we must attempt to reduce the barrier between a user and a knowledge acquisition system introduced in the axiom-acquisition phase. We identified a limited number of axiom templates, which would enable domain experts to specify most of the axioms by simply filling in the blanks in sentences describing the templates. This user-interface paradigm strongly parallels the task of acquiring instances of classes in traditional frame-based systems.

We examined axioms in ontologies in the Ontolingua ontology library [7] and several private Protégé ontologies [8] and discovered that most axioms that developers have chosen to encode have repetitive meanings and structures across different domains. In fact, we discovered that 20 templates accounted for 85% of the user-defined axioms. We acknowledge that this approach will not encompass the modeling of every axiom possible, but may be sufficient for common requirements.

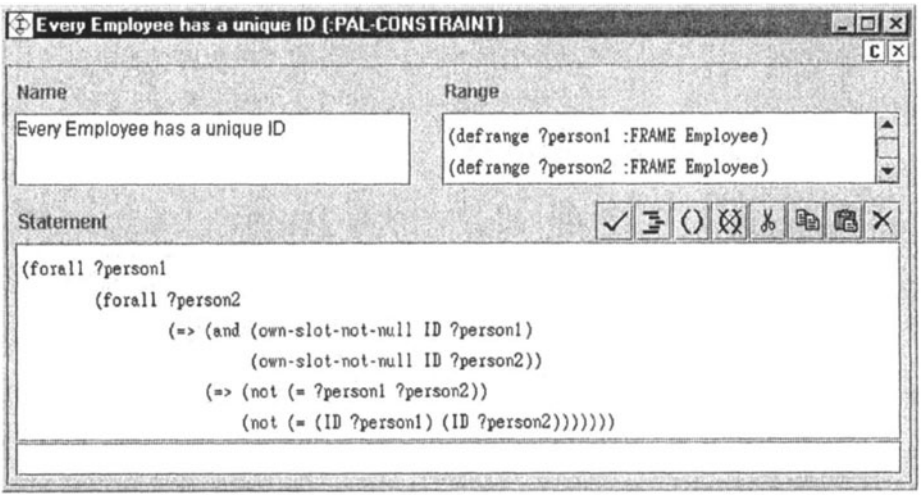

Figure 1. A Text-Based Axiom-Editing Environment in Protégé-2000. The axiom in the Protégé Axiom Language (PAL) says that every employee must have a unique ID.

2. METHODS FOR COLLECTING TEMPLATES

In order to identify axiom templates that would cover a large fraction of axioms that users have put in knowledge bases, we selected two ontology libraries that contained ontologies contributed by developers in many domains. We looked at axioms in ontologies from the Ontolingua ontology library [7] and from several Protégé ontologies [8]. These sources contain knowledge bases from diverse fields and have ample axioms. Ontolingua and Protégé have very similar knowledge models. The two systems introduce their own axiom languages, which are very similar. The axiom language of Ontolingua is KIF, a declarative language for expressing logical statements, and the Protégé Axiom Language (PAL) is a subset of KIF.

We examined 18 ontologies and 592 axioms. This set included ontologies for chemical elements, enterprise, bibliographic data, and malaria. Most have 10 to 30 axioms. We did not examine those axioms associated with upper level ontologies. These axioms seldom produce context-independent reusable patterns due to their complexity and specificity. We also did not examine computer-generated axioms, which contain information encodable in the native frame-based representation.

2.1 Knowledge Model

We chose the general frame-based knowledge model of Protégé[8]. A frame is a data structure that represents objects, abstract categorizations of objects, or concepts. Each frame has slots that contain values that describe the frame. A frame that refers to a general concept is a *class*, and a particular object of that class is an *instance* created by the process of *instantiation*. Most FRSs organize classes in a taxonomic hierarchy. A *superclass* is a superset of instances in its subclasses. Subclasses inherit slots from superclasses. We can specify attributes of slots such as the value type, cardinality, default values, and a data range. However, as we previously mentioned, we cannot usually specify relations between different slots of the same or different instances. These assumptions are consistent with the general knowledge model underlying the Open Knowledge-Base Connectivity (OKBC) protocol [3], which was designed as a means of communication among different frame-based knowledge bases.

2.2 From patterns to templates

Our process for generating generic axiom templates from the actual axioms in the ontologies consisted of the following steps:

1. We identified axioms that followed exactly the same **pattern**. They were identical except for the names of specific variables and frames.
2. We generalized similar patterns into **templates**. A template accounted for minor variations among patterns. For example, two patterns "A contains B" and "A does not contain B" give rise to one template "A contains/does not contain B."
3. We derived generic properties for categorizing the templates.

Consider the following two axioms in Athena_Client, a private Protégé ontology by Samson Tu:

```
(forall ?process
    (=> (slot-not-null inhibited_by_object ?process)
        (/= "cytochalazin" (name (inhibited_by_object ?process)))))

(forall ?process
    (=> (slot-not-null inhibited_by_object ?process)
        (/= "neuraminadase" (name (inhibited_by_object ?process)))))
```

These axioms vary only in referred slot value "cytochalazin" and "neuraminadase." Thus, these two axioms give rise to the following pattern:

All instances of class ___ do not contain the value ___ in slot ___.

We then generalize these patterns further, trying to satisfy two constraints: (1) have a small number of templates in the final collection, simplifying the search for the appropriate template for the domain expert; (2) ensure that the templates do not become too general as to be incomprehensible because of the many variations that they have. Section 3 will explore the methodologies for deriving templates from similar patterns and generic properties from the templates.

Every instance of class ___ whose (slot ___ has value ___)(*N) must have (slot ___ has value ___)(*M).
Example: A student who has fewer than 180 units or who has not completed the distribution requirement cannot graduate.⇔Every instance of **Class Student** whose **Slot units-completed : Class Student** has value **< 180 OR Slot distribution-completed : Class Student** has value **false** must have **Slot is-graduating : Class Student** has value **false**.
Every instance of class ___ appears at least once in slot ___ of any instance of class ___.
Example: Every student has at least one advisor.⇔Every instance of **Class Student** appears at least once as a value of **Slot advisee : Class Professor** of any instance of **Class Professor.**
Every instance (A) of class ___ that is a value of slot ___ of instance (B) of class ___ must have the same value in its slot ___ as in slot ___ of B.
Example: Every student project that is a continuation of a previous project has the same starting data available as the final data available of the previous project. ⇔Every instance (A) of **Class Project** that is a value of **Slot continuation-of : Class Project** of instance (B) of **Class Project** must have the same value in its **Slot starting-data : Class Project** as in **Slot final-data : Class Project** of B.
For every instance of class ___, slots ___ and ___ cannot have the same value.
Example: A student does not have an activity he both likes an dislikes. ⇔ For every instance of **Class Student**, **Slot favorite-activities : Class Student** and **Slot disliked-activities : Class Student** cannot have the same value.

Table 1. A partial list showing templates as "fill-in-the-blanks" sentences and sample usages.

3. AXIOM TEMPLATES AND THEIR PROPERTIES

The most simplistic form of a template consists of a fill-in-the-blanks sentence and an example of usage. By coupling these features, we ensure that a specific example will clarify its generalized template when the user does not immediately understand the latter. Table 1 represents a partial list of the 20 templates we identified in this reduced form.

The expanded form of a template incorporates additional details about its usage. For example, many templates allow for variations: different number of conditions or resulting statements of the same type, different modifiers, and so on (Section 3.1). Thus, as mentioned before, a template gives rise to several patterns (Section 3.2) where all the possible alternatives have been

specified. Therefore, in addition to its English-language representation, each template requires the relevant information from which one can extract a single pattern and instantiate an axiom. The complete definition of a template consists of the following elements, as shown in Figure 2 (the numbers in the list correspond to the numbers in the figure):

1. The English-language representation or the "name" of the template as a "fill-in-the-blanks" sentence. For the template in Figure 2, this representation is:

   Every instance of class ___ appears at least once in slot ___ of any instance of class___

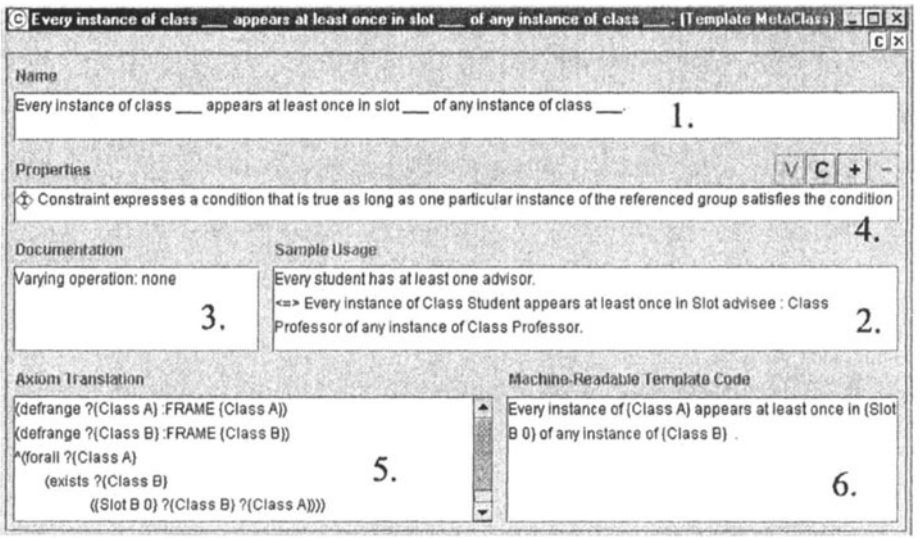

Figure 2. Elements of a complete template definition. Numbers on the figure correspond to elements in the list in Section 3. Each template definition includes a sentence with blanks to fill in, a sample usage, documentation, a list of properties, a prototypical axiom translation and a machine-interpretable expression of the template.

2. A sample usage sentence that includes a possible interpretation of the template with the blanks already filled in:

   Every student has at least one advisor ⇔ Every instance of **Class Student** appears at least once in **Slot advisee : Class Professor** of any instance of **Class Professor.**

3. Documentation: A short description of the varying operations one may perform on the templates. This field for the sample template is left blank because the template represents a single pattern rather than a group of patterns. We consider this issue in Section 3.1.
4. A set of properties describing the template. We discuss properties in Section 3.2.
5. A 'generalized' pattern that derives from the source axioms of a template. We replace specific frame and/or value references by variables and

indicate where variations (Section 3.2) are possible. Below is a translation of the example template into Protégé Axiom Language (PAL):

```
(defrange ?{Class A} :FRAME {Class A})
(defrange ?{Class B} :FRAME {Class B})
^(forall ?{Class A} (exists ?{Class B} ({Slot B 0} ?{Class B} ?{Class A})))
```

6. Machine-interpretable expression of the template: The expression contains information that enables the automatic generation of a template as an English sentence in the user interface. It also dictates what type of frame can fill in the blank or specifies additional modifiers (see section 3.1). This feature allows us to set additional user-interface controls on the types of the allowed frames that users can fill in a particular blank. Furthermore, we can use this structured internal representation to generate axioms by substituting specific values to the 'generalized' pattern.

   Every instance of {Class A} appear at least once in {Slot B 0} of any instance of class {Class B}

   The above example limits user's choice of a slot to those slots of the second class B, thus effectively preventing users from composing potentially inappropriate axioms.

3.1 Variations in Templates

Each template can have several variations. Variations can result from using multiple conditions or resulting statements, or from applying different operations to slot values.

3.1.1 Generating multiple conditions

A template can be a generalization of many patterns or a single pattern. Consider this template:

Slot ___ of instances of class ___ must contain elements that are instances of the class specified by slot ___

The sample usage of this template is:

A student (Instance of Class **Students**) can only enroll in the courses offered by the Business School (slot **Allowed-Course-Group**)

A user may only fill in the values to generate axioms but may not alter the template in any way to produce an axiom with a slightly different pattern. However, among the axioms that we examined, we found patterns with multiple conditions or multiple resulting statements. For example, consider the following axiom:

```
(exists ?current_med
    (and(drug_name ?current_med "hydrochlorothiazide")
        (> (daily_dose ?current_med) 25))
```

It states, "There exists a medication named "hydrocholorothiazide" with a daily dose greater than 25ml." Notice that there are two statements in this axiom: "medication is named "hydrocholorothiazide" and "dose is greater than 25ml." Consider another example:

```
(exists ?drug_usage
    (Drug_Class_Name ?drug_usage Beta-Blockers-Cardioselective))
```

It states "There exists a drug of the type 'cardioselective beta-blocker.'" Here we have only one statement: "drug class name is Beta-Bloker-Cardioselective." To account for this type of variation, we generalize the templates to allow an arbitrary number of conditions and resulting statements. For example, we can generalize the above pattern to:

There exists an instance of class ___ that contains (value ___ in slot ___.) (*N) (1)

The "N" modifier indicates that there can be more than one condition or statement. Therefore, this template generalizes patterns such as:

There exists an instance of class ___ that contains value ___ in slot ___ .

There exists an instance of class ___ that contains value ___ in slot ___ and value ___ in slot ___.

This convenient notation allows template designers to store structurally similar patterns into one template. This notation therefore prevents users from enumerating every variation to ensure coverage of the library as if there was a one-to-one mapping from patterns to templates. We will explain the use of varying operations such as those represented inside the round brackets to generate additional patterns.

3.1.2 Varying operations

Another way to customize a template is to specify an explicit operation or to use the built-in varying operations. Consider the following further generalization of pattern (1):

There must be (at least one/exactly one/at most one) instance of class ___ that contains/does not contain value ____ in slot ____

This example provides choices of operations among "only one/ at least one/ at most one," and between "contains value ___ / does not contain value ___." Thus, the template can lead to equally valid fill-in-the-blank patterns such as:

1) Only one slot ___ of instances of class ___ contains value ___.

2) At most one slot ___ of instances of class ___ does not contain value ___.

3) At least one of the instances of class ___ contains ___ in its slot ___.

Each slot type entails a specific set of operations on the slot. For example, for a numeric slot, operations are comparison functions such as <, <=, =, >=, >, =, not =; for a slot that has other instances as its value, operations are "contains/does not contain"; for a string-valued slot, operations are about substrings, such as "present/not present/begins with/ends with". These variations are implicit in the template because we can derive them directly from the slot value type.

3.2 Properties of templates

A process that is equally important to identifying templates is devising a strategy to organize them. We believe that organizing the templates into an inheritance hierarchy is not feasible: The templates are usually so abstract that even more general "super-templates" will be very vague and incomprehensible. A template may also fall into multiple categories. However, the latter situation contributes to an alternate solution for organizing templates. First, consider the following example that shows how a template can belong to multiple categories. We identified a group of templates that is concerned with expressing a condition that is true as long as one particular instance of the referenced group satisfies the condition:

1. All instances of class ___ appear at least once in slot ___ of any instance of class ___.

2. Every instance (A) of class ___ that is a value of slot ___ of instance (B) of class ___ must have the same value in its slot ___ as in slot ___ of B.

Similarly, there is another group represented by the following characteristic: the value of an instance's slot is determined by the slots of other instances in the former instance's slots.

1. Every value (B) of slot ___ of an instance (A) of class ___ must contain the same value in slot ___ of A and the value in slot ___ of B.

2. Every instance (A) of class ___ that is a value of slot ___ of instance (B) of class ___ must have the same value in its slot ___ as in slot ___ of B.

Note that the second templates from both examples are identical. We identified several properties of templates, which are not mutually exclusive:

1. Constraint involves direct comparisons between every instance of a class.
2. Constraint expresses a condition that is true as long as one particular instance of the referenced group satisfies the condition
3. Constraint involves value(s) of a slot determined by another own slot(s)

4. Constraint involves value(s) of a slot determined by the slots of other instances(in/not in former instance's) slots
5. Constraint involves multiple constraining/constrained slots.

We distinguish these high-level descriptions of templates from the previously discussed "super-template," because the former is not an attempt to find a more structurally general representation of template groups. Rather, this type of categorization describes the templates at a very high level that assumes no knowledge about the templates' structure, but only the common "thoughts" that they express.

We believe that these characterizations of templates can serve as the basis of implicit organization through descriptions rather than through a hierarchical organizational scheme; hence we call them "properties" of templates. We create a flat hierarchy of templates in which all templates have the superclass *template*, but each contains a section describing the properties that it satisfies. This strategy also permits much flexibility; the user can customize the templates by creating new properties. This feature also enables searching for templates by filtering out the ones that do not satisfy user's particular characterization, rather than searching through a predetermined sequence of classifications/tree-traversals unfamiliar to the intended users.

4. DISCUSSION

After examining axioms in existing ontology libraries we identified two important facts: (1) Many ontologies simply lack axioms, likely because axioms are very hard for domain experts to write. (2) Most of the axioms that do exist in the ontologies can be described by a small set of templates. We can use these templates to allow users to specify axioms by selecting an appropriate template and filling in the blanks rather than by having to conceptualise and express a complex axiom in first-order logic.

Tu and Musen [11] observe in a clinical domain that most of the axioms that domain experts need to write follow a limited set of patterns. They created a Protégé-based knowledge-acquisition tool for medical guidelines. In their approach, instead of writing an axiom in PAL, a domain expert fills in a form that looks exactly as a form for any regular instance in the knowledge base. For example, there is a check box to mark whether actions are predicated on the presence or the absence of a particular criterion; there is a box to select from the list of time periods; there is a box to select a value from the list of conditions, and so on. Once the user fills in the form, the system automatically generates an axiom in PAL. In this approach, the user can specify a large fraction of axioms simply by performing a fairly intuitive task—filling in blanks in a form. However, the templates are specific to

defining medical guideline criteria. It would be impossible to use them verbatim in another application area.

Research in classifying and representing axioms in a user-friendly way has been relatively sparse in the knowledge base community. Staab and Mädche [9] recommend an approach similar to ours: They note that many axioms that they encountered in ontology-engineering projects follow similar patterns. They abstract several classes of axioms, such as algebraic axioms (e.g., reflexivity and symmetry of properties), axioms dealing with composition of relations, axioms describing part–whole relations and so on. The representation of an axiom (the one that the user sees) is then a predicate for which a user needs only to specify arguments. The user does not need to worry about the symbolic representation of the axiom in a specific language. For the types of axioms that the authors discuss, the predicate approach works well. However, it is unclear how the same approach would work for slightly different variations of axioms and axioms with varying number of conditions. In addition, for many users a look at the predicate name does not immediately clarify what the underlying axioms are about (e.g., consider a predicate "PartonomicRolePropagation"). In the ontology libraries that we considered, most axioms did not fall in the categories identified by Staab and Mädche. However, we affirm the approach of representing axioms as objects in a knowledge base and allowing users to represent axioms at a conceptually high level of abstraction.

Researchers have applied this idea not only to specifying single axioms but also to specifying sets of axioms. In some cases, a set of axioms represents a complex knowledge-base component, such as the description of a physical process [1,2]. In other cases, a set of parameterised axioms represents a new constraint that cannot be directly expressed in the corresponding language [10]. As a result, the task of the knowledge-base designer is greatly simplified, since all he needs to do is to instantiate a particular pattern rather than to write a set of axioms from scratch. In addition, the user creates the definition in an intuitive syntax and the system can translate it into several specific languages if necessary. We not only identify a set of these recurring patterns, but we do so as a result of analysing a large set of real-world axioms that users have created in their ontologies.

The use of design patterns in object-oriented programming [4] is another approach to adopting recurring representation patterns as the building blocks of design. A design pattern records, explains, and evaluates patterns of code in object-oriented systems. Similar to our templates, design patterns encode information for structuring, creating, and managing the 'design idioms.'

Our primary goal in designing the set of axiom templates is to facilitate the task of specifying axioms for domain experts. Naturally, the primary

application of this research is to design an easy-to-use interface, which would enable domain experts to specify axioms quickly and easily.

In the interface, we may present a set of available templates and a list of their properties. If a user finds the size of initial list of template unwieldy for browsing, he can examine the list of properties (Section 3.2), and check the ones that are likely to be compatible with his thought to filter out unlikely candidates from the template list. A user can browse the remaining templates, each of which is accompanied by a sample usage in both English and fill-in-the-blanks representations. Once ready, the user can simply select appropriate variations, fill in the blanks, and 'instantiate' an axiom without encountering the obscure syntax of the axiom language. Note that such an interface will work for many underlying axiom languages because the selection and instantiation of a template is the same regardless of the underlying language.

We envision that advanced users will be able to extend the set of templates which we also model as an ontology in Protégé; the templates are simply classes in the knowledge base. Thus, the advanced users can create new template classes and a processing engine can incorporate them into the interface. Similarly, they may create new instances of the class *Property* to introduce additional categorization of templates.

We are currently implementing such a system to perform empirical evaluation on coverage, usability, and readability of the templates. To evaluate coverage, we will examine axioms in an ontology library that we have not yet considered and determine what fraction of axioms in that library our templates represent. To evaluate usability, we will perform user studies where we will first ask domain experts to express constraints in English that they hope to incorporate in their knowledge base but cannot encode in an axiom language. We can then measure the ratio of axioms they actually are able to encode with our template-based system compared to the number of constraints that we deem expressible in that language. The result will attest the practicality of our strategy in promoting the composition of more axioms. Finally, we will experiment with several alternatives for the English representations of the templates to derive the most intuitive statements.

ACKNOWLEDGMENTS

We are very grateful to Samson Tu, Mor Peleg and Iwei Yeh for sharing their Protégé ontologies. We would also like to thank Monica Crubézy for suggestions about template designs. This work was supported in part by a contract from the U.S. National Cancer Institute. Johnson Hou was supported by the Stanford President's Scholar Intellectual Exploration Grant for lower-division undergraduate students.

REFERENCES

1. Barker, K., Clark, P. and Porter, B. A Library of Generic Concepts for Composing Knowledge Bases. In: *1st Int Conf on Knowledge Capture (K-Cap'01)*. Victoria, BC, 2001.
2. Clark P., Thompson, J., and Porter, B. Knowledge Patterns. In *KR'2000 (Proc 7th Int Conf)*, pages 591-600, Eds: A. Cohn, F. Giunchiglia, B. Selman, CA:Kaufmann, 2000.
3. Chaudhri, V.K., Farquhar, A., Fikes, R., Karp, P.D. and Rice, J.P. OKBC: A programmatic foundation for knowledge base interoperability. In: *Fifteenth National Conference on Artificial Intelligence (AAAI-98)*. Madison, Wisconsin: AAAI Press/The MIT Press, 1998.
4. Gamma, E., Helm, R., Johnson, R. and Vlissides, J., *Design Patterns: Elements of Reusable Object Oriented Software*. Reading, MA: Addison-Wesley, 1999.
5. Genesereth, M.R. and Fikes, R.E., *Knowledge Interchange Format, Version 0.3, Reference Manual*, Knowledge Systems Laboratory, Stanford University, 1992.
6. Karp, P.D., *The design space of frame knowledge representation systems*, SRI AI Center: Menlo Park, CA, 1993.
7. Ontolingua, *Ontolingua System Reference Manual*, Knowledge Systems Lab, Stanford University, http://www-ksl-svc.stanford.edu:5915/doc/frame-editor/index.html
8. Protégé, *The Protege Project*, Stanford Medical Informatics, Stanford University, http://protege.stanford.edu, 2000.
9. Staab, S. and Mädche, A. Axioms are objects, too - Ontology Engineering Beyond the Modeling of Concepts and Relations. In: *ECAI 2000 Workshop on Ontologies and Problem-Solving Methods*. Berlin, 2000.
10. Staab, M. Erdmann, A. Maedche. Engineering ontologies using semantic patterns. In A. Preece & D. O'Leary (eds.), *Proceedings of the IJCAI-01 Workshop on E-Business & the Intelligent Web*. Seattle, 2001.
11. Tu, S.W. and Musen, M.A. Modeling Data and Knowledge in the EON Guideline Architecture. In: *MedInfo 2001*. London, 2001.