(In Proc AAAI'97, pp369-376, CA:AAAI)
http://www.cs.utexas.edu/users/pclark/papers

Building Concept Representations from Reusable Components*

Peter Clark
Boeing Company
PO Box 3707, Seattle, WA 98124
clarkp@redwood.rt.cs.boeing.com

Bruce Porter
Department of Computer Sciences
University of Texas at Austin, TX 78712
porter@cs.utexas.edu

Abstract

Our goal is to build knowledge-based systems capable of answering a wide variety of questions, including questions that are unanticipated when the knowledge base is built. For systems to achieve this level of competence and generality, they require the ability to dynamically construct new concept representations, and to do so in response to the questions and tasks posed to them. Our approach to meeting this requirement is to build knowledge bases of generalized, representational components, and to develop methods for automatically composing components on demand. This work extends the normal inheritance approach used in frame-based systems, and imports ideas from several different areas of AI, in particular compositional modeling, terminological reasoning, and ontological engineering. The contribution of this work is a novel integration of these methods that improves the efficiency of building knowledge bases and the robustness of using them.

Introduction

Our goal is the construction of knowledge-based systems capable of answering a wide range of questions, including questions unanticipated when the knowledge base was constructed. Earlier research on one large-scale project — the Botany knowledge-base project (Porter *et al.* 1988) — shows that *if* detailed, declarative representations of concepts are available, then sophisticated question-answering performance can be achieved (Lester & Porter 1997; Rickel & Porter 1997). However, manually constructing such representations is laborious, and this proved to be a major bottleneck in the project; moreover, it is simply not possible to anticipate all the concept representations that may be needed for answering questions.

This points to a fundamental requirement for the future design of such systems: they must be able to *dynamically construct* new concept representations automatically, and in response to questions posed to them. During the past two years we have been developing methods to meet this requirement, which we present in this paper. Our approach is to structure a knowledge-base as a set of *generalized, representational components*, and to develop methods for automatically composing components on demand. This work extends the normal inheritance approach used in frame-based systems, and imports ideas from several different areas of AI, in particular compositional modeling, terminological reasoning and ontological engineering. The contribution of this paper is a novel integration of these methods that improves the efficiency of building knowledge bases and the robustness of using them. In the wider context of knowledge-based systems research, our concern is with building *domain models* compositionally, as opposed to building *problem-solving methods* compositionally, eg. (Chandrasekaren 1986).

Finally, it is important to note that we are *not* proposing a new knowledge representation language; rather, we are concerned with how existing languages can be used to build representations in flexible, reusable ways. While this work has been implemented using a particular representation language called KM (Eilerts 1994), the approach could also be applied using several other existing languages, eg. Algernon (Crawford & Kuipers 1991).

A Simple Example

One of our target domains, which we use for illustration throughout this paper, is *bioremediation*: the removal of toxic waste using micro-organisms that convert pollutant into harmless bi-products. For a system to answer a variety of questions about bioremediation (requiring tasks such as description, prediction, and explanation), it needs a *concept representation* — such as the one in Figure 1 — which states the relationship of bioremediation to other concepts.

The concept representation combines properties from numerous abstract concepts. It includes a process of *conversion* (in which pollutant is converted into a

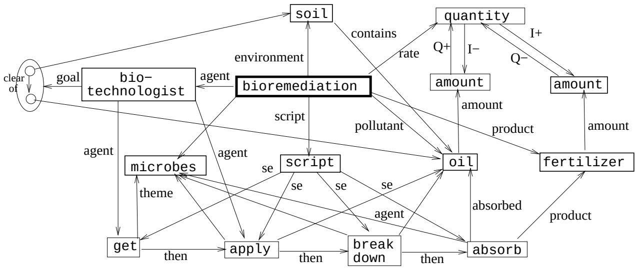


Figure 1: A Representation of Bioremediation of Soil Polluted by Oil.

fertilizer-like compound), *treatment* (in which microbes are applied to the pollutant), and *digestion* (in which microbes digest the pollutant). These three concepts are shown in Figure 2.

This paper presents a way to construct concept representations by composing their constituents, and to control the process so that only those portions of a concept representation that are needed to answer a query are built.

Frame-Based Models of Composition

A standard and intuitive account of automatic concept construction, and the starting point for our work, is the use of frame representations with multiple inheritance. In this model, the basic representational unit is a *concept* ('frame'), and concepts are organized into a taxonomic hierarchy (lattice). In its simplest form, a concept is described by a set of *properties*, each of which is a $\langle slot, value \rangle$ pair denoting the concept's relationship to another concept. All of these properties need not be explicitly encoded on the concept's frame. Rather, a concept can be composed from other concepts in two ways:

Inheritance: A concept declared to be a specialization of (possibly multiple) concepts will automatically acquire their properties. For example, from (Andersen 1996), the concept *house-boat* could be specified as a specialization of *house* and *boat*, gaining all their properties, such as *inhabited by humans* and *floats on water*.

Modifying a Base Concept: A new concept can be declared a specialization of some base concept, with one or more of its slot values filled by the modifier concept(s). For example, *apartment-dog* could be declared a specialization of *dog*, with the slot *habitat* having the value *apartment*.

This approach achieves composition efficiently, requires very little inference, and is intuitive. However, there are two concerns that it fails to address:

1. **Conceptual Systems:** While we can declare taxonomic relationships between individual concepts, there appears to be no easy way of composing *systems* of concepts. For example, the notion of *containment* can be viewed as an abstract system of relationships (or constraints) among a *container*, a *contained-object*, and a *portal* (eg. the *contained-object* is smaller than the *container*, the *portal* is a surface-part of the *container*). We would like to automatically import this abstract system to concepts in which it applies (eg. for some tasks, it might be useful to view a person as a container of food), rather than manually enumerate its axioms from scratch. This is difficult to do with inheritance without adding unintuitive axioms such as *food isa contained-object* or *most-recent-meal* is a special kind of *contained* relation; it is impossible to do when the same abstract system can be applied in multiple ways, eg. through a different mapping, we can also view a person as a container of ideas.
2. **Concept Interactions:** Standard methods of modifying a base concept do not provide an adequate means of composing two concepts by using one to modify a property of the other. This type of composition is very common and is often used in natural language to describe a new concept. An adequate description of *apartment-dog*, for example, involves more than a restriction on habitat; such a dog is fed certain food, gets less exercise, and is probably smaller than its non-urban counterpart. All these features are derived from interactions between the original features of *apartment* and *dog*, but this interaction is not well modeled in standard methods.

We address these concerns in two ways:

1. We change the organization of knowledge from one based on *individual* concepts to one based on *systems* of interacting concepts. This change is partly conceptual, in that a knowledge base is viewed as a set of composable mini-theories, and it is partly implementational, in that new mechanisms are needed to combine these mini-theories. We henceforth refer to these systems of concepts as *components*.
2. We add a computational mechanism for composing a concept description from components, which involves a repeated cycle of *classify* and *elaborate* to compute the components' relationships:
   - During classification, components pertinent to the concept description are identified.
   - During elaboration, the information supplied by those components is merged with the concept description, triggering further classification.

   Thus classification enables elaboration, and elaboration triggers further classification. As shown below, by applying this mechanism in a goal-directed

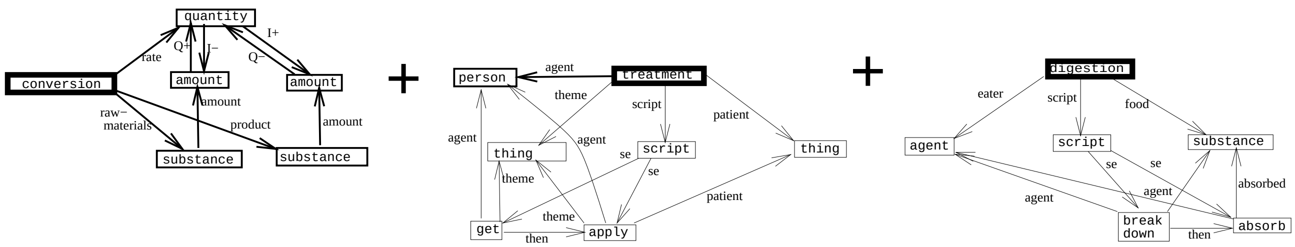


Figure 2: The concept *bioremediation* is a composition of multiple concepts, including specializations of *conversion, treatment*, and *digestion.*

way, components required for describing novel concepts can be identified and integrated automatically, thereby constructing representations of new compound concepts.

Components

Intuitively, a component encapsulates a coherent system of concepts and their relationships; it embodies an abstract "mini-theory", or pattern of interrelationships among those concepts, which can be mapped onto a variety of situations where that pattern is deemed to hold. This view borrows heavily from work in compositional modeling, where a model fragment encapsulates a system of relationships describing some aspect of a physical phenomenon, and can be applied when that phenomenon occurs (Levy 1993; Falkenhainer & Forbus 1991). It also draws from recent work in ontological engineering, eg. (Farquhar & Gruninger 1997), in which an *ontology* encapsulates axioms describing a theory about some area (eg. substances, ceramics, sets).

More formally, we define a component as a triple $< P, A, R >$ where:

- P is a set of **participants**, denoting the objects involved in the pattern being modeled.
- A is a set of **axioms**, describing relationships among the participants.
- R is a set of **roles** with which the participants can be labeled, each role referring to a different participant. Roles are parameter names, analogous to Lisp's keywords for naming parameters.

We define the component's *interface* as the set of role-participant pairs of the component. Figure 3 shows the *conversion* component of Figure 2 expressed in this framework.

A component is *instantiated* by binding its participants to objects in the target domain. When this happens, the relationships given by the component's axioms are asserted for those objects. The roles provide a set of unique labels with which the participants can be unambiguously referenced. A component's axioms

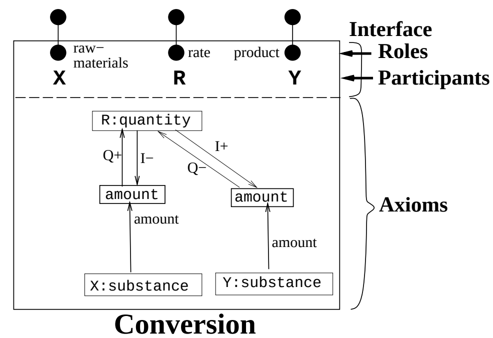


Figure 3: The Anatomy of a Component.

can be expressed in rules in some suitable knowledge representation language.

Specifying Compound Concepts

We can now specify *compound concepts* as a composition of components. A specification states how the components' interfaces "plug together", describing how their roles correspond. Figure 4 shows a specification for *bioremediation* stating that:

- it includes an instance of *conversion*, where the *raw-material* of the *conversion* is the *patient* in the *bioremediation*, etc.
- it includes an instance of *digestion*, where the *eater* in the *digestion* is the *theme* in the *bioremediation*, etc.
- it is a type of *treatment*, where all the roles map directly without renaming.
- The *theme* is *microbes*, and the *product* is *fertilizer.*

Note that in the case of *treatment* as a component of *bioremediation*, in which all the roles map directly together (ie. without renaming), composition reduces to standard inheritance. Therefore, our component-based framework can be seen as a generalization of a normal inheritance approach. Note also that the specification still includes modifiers, specializing parts of the base concept (eg. that the bioremediation *agent* is *microbes*).

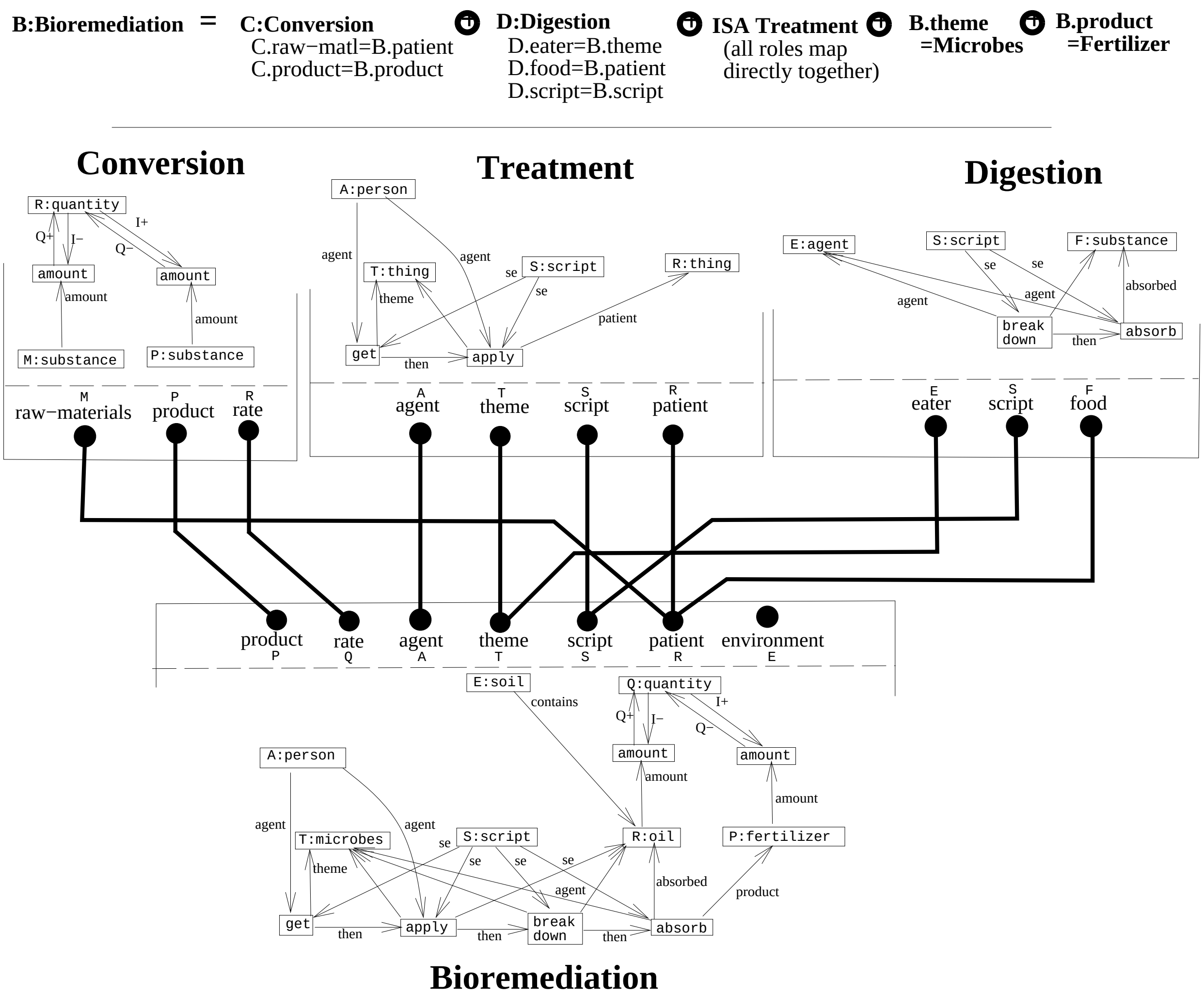


Figure 4: Bioremediation can be specified as a composition of components (shown in both text and graphical forms). By only specifying how the components' interfaces map together, the target concept can be constructed automatically.

We now describe a method for generating, in a goal-directed way, a concept representation from a specification.

Dynamic Concept Construction

A 'concept representation' is an integration of information from the concept's components, subject to the mapping given in the concept's specification. It is more than a simple union of the components' axioms, as axioms may interact to allow additional information to be inferred about the concept. In general, constructing a compound concept representation in its entirety (ie. exhaustively inferring all facts about it) is intractable, and few of the facts that can be inferred are relevant for any particular task. Therefore, by design, our concept construction method is driven by the questions posed to the knowledge-base, either from a user or an application system. We are concerned with questions that reference (at least) one concept that is not explicitly in the knowledge base, eg. "treatment of crude oil using microbes", and ask for one of its properties, eg. "how much will it cost?". To answer such questions the system first creates a *scenario*, consisting of Skolem individuals denoting parts of the compound concept (eg. *treatment001*, *oil001*, *microbes001*) plus the relationships among them. The system then elaborates the scenario to find the answer to the given question. In our work, a top level application system (eg. for diagnosis) will typically make many such queries to the knowledge-base, causing the initial scenario to become extensively elaborated over time.

The base operation in this process is answering a single question. This involves more than simply backward-chaining using the components' axioms, because as more facts are inferred about the scenario, it may become possible to refine the classification of individuals in the scenario (eg. a *tool* might be refined to be a *hammer* if it is discovered that the tool is used to affix *nails*). This refinement results in additional components (eg. for *hammer*) becoming applicable; therefore, it is essential to keep the classification of individuals in the scenario up-to-date, to ensure the system has access to all the relevant axioms. As a result, the question-answering algorithm is based on an iterating cycle of classify and elaborate, which we will describe shortly.

There is a well-known trade-off to handle here, between expressivity on one hand, and completeness and tractability of inference on the other. With the exception of a few systems, eg. KRIS (Baader & Hollunder 1991), most reasoners tolerate some incompleteness in inference, so that a more expressive representation language can be used. Our algorithm similarly does this. Recent work on *access limitation* (Crawford & Kuipers 1991) has shown how tractability can be maintained through the use of *access paths*, which guide the inferences that an inference engine might make. For similar purposes, our algorithm uses access paths as the language for querying the knowledge base.

An access path is a sequence of binary predicates:

$$P_1(C, x_1), P_2(x_1, x_2), ..., P_n(x_{n-1}, x_n)$$

where C is a constant (denoting an individual in the scenario) and the x_i's are free variables.[1] An access path references the set of values for x_n (the last variable in the path) for which there exists at least one value for all the other variables in the path. For example, the access path $parent(John, x), sister(x, y)$ references "John's parents' sisters". In terms of the graphical representation of concepts used earlier, an access path corresponds to the set of paths that start at node C and traverse arcs labeled $P_1, ..., P_n$.

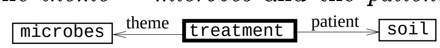
Given a query expressed as an access path (posed by a user or an application program), the composition algorithm applies the Classify-Elaborate Cycle to "follow the path". The goal is to compose just the information necessary to answer the query (ie. find all values of the last variable x_n implied by the knowledge-base). It does this by working from left to right, applying the following Classify-Elaborate Cycle for each predicate $P_i(x_{i-1}, x_i)$ in the path in turn:

[1]This is somewhat simplified, but it suffices for our purposes.

Classify: Before searching for solutions for the predicate $P_i(x_{i-1}, x_i)$, first try to refine the classification of x_{i-1} from its known class to something more specific, to find its most specific generalization(s). This, in turn, may require finding additional information about x_{i-1} to determine whether it satisfies a concept definition, hence making a recursive call to the Classify-Elaborate Cycle.

Elaborate: To find solutions for $P_i(x_{i-1}, x_i)$, search for axioms that conclude value(s) of x_i, looking in the components in which x_{i-1} participates. If any are found to apply, the concept description can be extended along the relation P_i. This similarly may call the Classify-Elaborate cycle recursively, to compute the antecedents of those axioms.

To illustrate this, consider the compound concept *microbe-soil-treatment*, referring to the treatment of soil with microbes. The concept specification consists of two modifiers of the base concept *treatment*: the *theme* = *microbes* and the *patient* = *soil*:

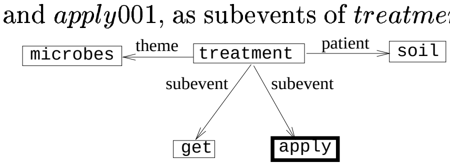


This compound concept might be needed for answering numerous questions, such as "what is the cost of the equipment required for microbe-soil-treatment?" Expressed as an access path, this query is:

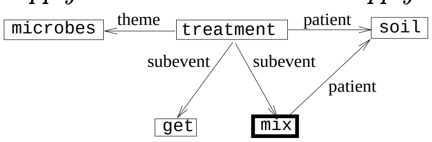
$$subevent(treatment001,S), instrument(S,I), cost(I,C)$$

denoting the cost of the instruments of the subevents of *treatment001*, a Skolem instance of the concept *microbe-soil-treatment*. The sequence of operations for constructing a representation of *treatment001* to answer the query is as follows:

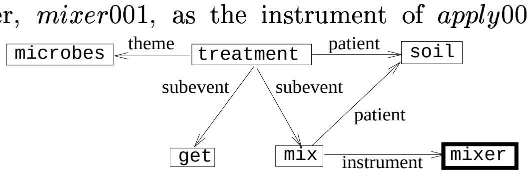
1. (Classify) Before searching for solutions for the first predicate in the path, $(subevent(treatment001, S))$, try to refine the classification of *treatment001* from *treatment* to something more specific. We will assume that none are found (no relevant concept definitions apply).
2. (Elaborate) For $subevent(treatment001, S)$, search the components that contribute to *treatment001* to find axioms that conclude about *subevent*. In this case, an axiom from *treatment* is found and evaluated. The axiom asserts the existence of (and hence generates Skolem individuals for) two subevents, *get* and *apply*. Consequently, the concept description is elaborated by adding (instances of) *get* and *apply*, say $get001$ and $apply001$, as subevents of $treatment001$.



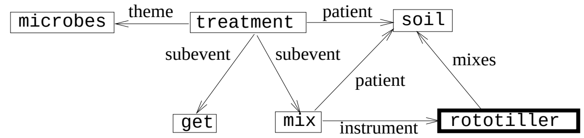
3. (Classify) Before searching for solutions for the second predicate in the path, ($instrument(apply001, I)$), try to refine the classification of *apply001*. (To simplify discussion, we omit the process of classifying $get001$, which is similar.) One potentially relevant definition (not shown here) is that a *mix* is defined as an *apply* where the patient is a *substance*. To ascertain if *apply001* satisfies this definition, the subgoal $patient(apply001, X)$ is set up. By recursively calling the Classify-Elaborate Cycle, a solution for this subgoal is found: an axiom for *treatment* states that the patient in the *apply* is the same as the patient in the *treatment*, and the patient in the *treatment* is known to be *soil* (from the initial concept specification). Thus, X is found to be (an instance of) *soil*, and, as *soil* is a substance, the definition of *mix* is satisfied. The system thus concludes that the most specific generalization of *apply001* can be refined from *apply* to *mix*.



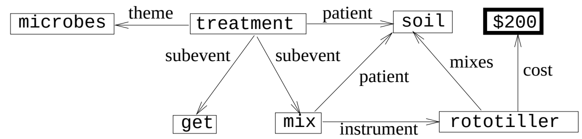
4. (Elaborate) For $instrument(apply001, I)$, search the components that contribute to *apply001* to find axioms that conclude about *instrument*. In this case, an axiom from *mix* is found and evaluated, asserting the existence of (and hence generating a Skolem individual for) an instance of *mixer*. Hence the concept description is elaborated by adding an (instance of) mixer, $mixer001$, as the instrument of $apply001$.



5. (Classify) Before searching for solutions for the third predicate in the path, ($cost(mixer001, C)$), try to refine the classification of *mixer001*. One potentially relevant definition is that a *rototiller* is a *mixer* in which the mixed is *soil*. To ascertain if *mixer001* satisfies this definition, the subgoal $mixes(mixer001, Y)$ is set up. Again, calling this procedure recursively, a solution for this subgoal is found: an axiom for *mix* states that the thing mixed by the *mixer* is the same as the patient in the *mix*, and the patient in the *mix* is known to be *soil* (from an earlier step). Thus, Y is found to be (an instance of) *soil*, and hence the definition of *rototiller* is satisfied. The system thus concludes that the most specific generalization of *mixer001* can be refined from *mixer* to *rototiller*.



6. (Elaborate) For $cost(mixer001, C)$, search the components that contribute to *mixer001* to find axioms that conclude about *cost*. In this case, an axiom from *rototiller* is found and evaluated, asserting the cost of a *rototiller* is \$200. Hence the concept description can be elaborated by adding $cost(mixer001, \$200)$ to the representation.



Hence the answer *\$200* is returned by the inference engine.[2]

If the system had started with a different specification, for example *microbe-oilSlick-treatment*, then a different chain of elaborations and classifications would occur. For example, the *apply* would have been refined to a *spread*, and the *instrument* would have been refined to a *sprayer*.

This example illustrates several distinctive features of our work. First, the concept of *microbe-soil-treatment* can be reasoned about even though it is not pre-built in the knowledge-base; instead, it is dynamically assembled from components. Second, even small differences in the initial specification of the concept may cause significantly different final representations to be constructed (eg. *microbe-soil-treatment* vs. *microbe-oilslick-treatment*), illustrating how the interaction of information from different components affects the representation built. Finally, the queries (eg. "what is the cost of the equipment required for *microbe-soil-treatment*?") control the actual representation that gets built.

Related Work

Our research focuses on two related issues: building knowledge bases of reusable components and combining components to build concept representations. We discuss related work on each of these issues.

[2]For simplicity this example assumed that each predicate in the query was uniquely satisfied. In fact, when a predicate can be satisfied in multiple ways (eg. the subevents of *treatment001* are *apply001* and *get001*, and each subevent might have multiple instruments), all of these solutions are added to the graph by the algorithm. Combining this information to answer a query (eg. summing the costs of the instruments of the subevents) is outside the scope of the algorithm.

With regard to building knowledge bases, at a general level, we follow the principle of structuring a large theory as a set of smaller, more manageable ones, as in "ontologies" in ontolingua (Farquhar, Fikes, & Rice 1997) and TOVE (Fox & Gruninger 1994), and "microtheories" in Cyc (Blair, Guha, & Pratt 1992). However, our work differs in two important ways. First, it is not our goal to *partition* a knowledge base (eg. by topic), although this too is useful; rather, our goal is to identify *repeated patterns of axioms* in a large theory, and then abstract and reify those patterns as components in their own right (analogous to the notion of "design patterns" in object-oriented programming). A single component may be used repeatedly in a knowledge base, each time with different mappings of its interface to concepts in the knowledge base[3]. Second, we add a well-defined interface to each component, which enables reference to the objects that participate in the component's axioms. We have taken this idea from compositional modeling, eg. (Falkenhainer & Forbus 1991; Levy 1993), and software engineering, eg. (Batory & O'Malley 1992; Goguen 1986). It is also similar to the use of signatures to characterize theories in category theory (Pierce 1991), where our participants correspond to sorts in category theory.

With regard to combining components to build concept representations, our approach relates to work in description logics (DLs), such as Classic (Brachman *et al.* 1991) and Loom (MacGregor & Bates 1987). Building a concept representation is similar to the process of 'normalizing' a conjunctive DL concept description, during which information about that concept is gathered and combined. In particular, the classify-elaborate cycle is similar to normalizing a concept description in the presence of ABox rules, whereby ABox individuals are classified, the classification determines the rules that apply to those individuals, the rules are applied, the individuals are reclassified (since the rules might have added information), and the cycle repeats. However in contrast to this DL algorithm which exhaustively constructs concept representations without regard to task, our algorithm is *goal-driven*, constructing only those parts of the concept representation required to answer questions. Our trade-off is to sacrifice completeness for a language sufficiently expressive for our purposes[4]. An interesting consequence of our approach is that the concept description which is built is question-specific, containing just that information required to answer the question(s) which were posed. Thus the algorithm can also be viewed as a 'concept characterization' method, assembling a 'view' of the concept as required for a particular task. This could have useful additional benefits in explanation generation and knowledge acquisition.

While this approach appears promising, there are several additional issues which must be addressed to apply this on a large scale. First, we do not have any principled methodology for identifying generalized representational fragments when crafting a knowledge-base, apart from the general heuristic of "look for recurring patterns of axioms". A more structured methodology for identifying and deciding on the boundaries of components would be invaluable for helping guide this process. Second, we have not addressed the issue of handling components based on conflicting assumptions, or for deciding which set of assumptions is appropriate for a particular task. Recent work in compositional modeling for selecting and managing assumptions, eg. (Falkenhainer & Forbus 1991), and on the use of lifting axioms to transform axioms across contexts based on differing assumptions, eg. (Blair, Guha, & Pratt 1992), point to methods for addressing these issues. Finally, methods for ensuring and maintaining consistency of components are needed, especially as a knowledge base evolves over time. We have recently started research on this issue (Correl & Porter 1997).

Summary

The overall goal of our research is to build knowledge-based systems capable of answering a wide variety of questions, including questions that are unanticipated when the knowledge base is built. This requires that systems be able to synthesize the knowledge structures needed to answer questions when those structures are not explicitly encoded in the knowledge base. We have described a process by which these structures can be assembled from abstract, reusable components, as they are needed to answer questions and perform tasks. We have presented a way to package information about abstract concepts into components which "plug together" to build a detailed representation, and we have shown how this process can be controlled by the knowledge requirements posed by each question and task, thus allowing a knowledge base to be more easily organized in a modular fashion.

[3]as opposed to "waterfall" models in which theories are placed in a partial order, and each theory acquires all the axioms of theories upstream of it

[4]In particular, we allow existential quantifiers in components' axioms. As complete reasoning in the presence of existential quantifiers is undecidable (Donini *et al.* 1992), our algorithm only generates and classifies Skolem individuals when a query path predicate refers to them, rather than whenever an object's existence is implied.

Acknowledgements

The authors thank Keith Williamson, Steve Woods, Ian Horrocks, Peter Patel-Schneider, Franz Baader and the anonymous reviewers for comments related to this paper. Support for this research was provided by a grant from Digital Equipment Corporation and a contract from the Air Force Office of Scientific Research (F49620-93-1-0239). This work was conducted at the University of Texas at Austin.

References

Andersen, C. 1996. A computational model of complex concept composition. Master's thesis, Dept CS, Univ Texas at Austin.

Baader, F., and Hollunder, B. 1991. A terminological knowledge representation system with complete inference algorithms. In *Proc First Int Workshop on Processing Declarative Knowledge*, volume 572 of *Lecture Notes in CS*, 67–85. Springer-Verlag.

Batory, D., and O'Malley, S. 1992. The design and implementation of hierarchical software systems with reusable components. *ACM Transactions on Software Engineering and Methodology.*

Blair, P.; Guha, R. V.; and Pratt, W. 1992. Microtheories: An ontological engineer's guide. Tech Rept CYC-050-92, MCC, Austin, TX.

Brachman, R. J.; McGuinness, D. L.; Patel-Schneider, P. F.; Resnick, L. A.; and Borgida, A. 1991. Living with CLASSIC: When and how to use a KL-ONE like language. In Sowa, J., ed., *Principles of Semantic Networks.* CA: Kaufmann.

Chandrasekaren, B. 1986. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert* 23–30.

Correl, S., and Porter, B. 1997. Iterative refinement of knowledge-bases with consistency properties. In Farquhar, A., and Gruninger, M., eds., *Proc AAAI Spring Symposium on Ontological Engineering.* AAAI (in press). 17–24.

Crawford, J. M., and Kuipers, B. J. 1991. Algernon – a tractable system for knowledge-representation. *SIGART Bulletin* 2(3):35–44.

Donini, F. M.; Hollunder, B.; Lenzerini, M.; Nardi, D.; Spaccamela, A.; and Nutt, W. 1992. The complexity of existential quantification in concept languages. *Artificial Intelligence* 53(2/3):309–327.

Eilerts, E. 1994. Kned: An interface for a frame-based knowledge representation system. Master's thesis, Dept CS, Univ Texas at Austin, USA.

Falkenhainer, B., and Forbus, K. 1991. Compositional modelling: Finding the right model for the job. *Artificial Intelligence* 51:95–143.

Farquhar, A., and Gruninger, M., eds. 1997. *Proc AAAI Spring Symposium on Ontological Engineering.* AAAI (in press).

Farquhar, A.; Fikes, R.; and Rice, J. 1997. Tools for assembling modular ontologies in ontolingua. In *Proc of the Fourteenth National Conference on Artificial Intelligence (AAAI-97).*

Fox, M. S., and Gruninger, M. 1994. Ontologies for enterprise integration. In *Proc 2nd Conference on Cooperative Information Systems.* Univ Toronto. 82–90. (Also see ontologies at http://www.ie.utoronto.ca/EIL/tove/toveont.html).

Goguen, J. A. 1986. Reusing and interconnecting software components. *Computer* 16–28.

Lester, J. C., and Porter, B. W. 1997. Developing and empirically evaluating robust explanation generators: The knight experiments. *Computational Linguistics* 22(3).

Levy, A. Y. 1993. Irrelevance reasoning in knowledge-based systems. Tech report STAN-CS-93-1482 (also KSL-93-58), Dept CS, Stanford Univ., CA. (Chapter 7).

MacGregor, R., and Bates, R. 1987. The LOOM knowledge representation language. Tech Report ISI-RS-87-188, ISI, CA.

Pierce, B. 1991. *Basic Category Theory for Computer Scientists.* MIT Press.

Porter, B. W.; Lester, J.; Murray, K.; Pittman, K.; Souther, A.; Acker, L.; and Jones, T. 1988. AI research in the context of a multifunctional knowledge base: The botany knowledge base project. Tech Report AI-88-88, Dept CS, Univ Texas at Austin.

Rickel, J. W., and Porter, B. W. 1997. Automated modeling of complex systems to answer prediction questions. *Artificial Intelligence.* (to appear).